VAX Text Processing Utility Manual: Part II

Order Number: AA–PBTNA–TE

June 1990

This manual describes the elements of the VAX Text Processing Utility (VAXTPU). It is intended as a reference manual for experienced programmers.

Revision/Update Information: This document supersedes the *VAX Text Processing Utility Manual* for VMS Version 5.2.

Software Version: VMS Version 5.4

digital equipment corporation
maynard, massachusetts

June 1990

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

© Digital Equipment Corporation 1990.

All Rights Reserved.
Printed in U.S.A.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDA	DEQNA	MicroVAX	VAX RMS
DDIF	Desktop–VMS	PrintServer 40	VAXserver
DEC	DIGITAL	Q-bus	VAXstation
DECdtm	GIGI	ReGIS	VMS
DECnet	HSC	ULTRIX	VT
DECUS	LiveLink	UNIBUS	XUI
DECwindows	LN03	VAX	
DECwriter	MASSBUS	VAXcluster	digital™

The following is a third-party trademark:

PostScript is a registered trademark of Adobe Systems Incorporated.

ZK4350

Production Note

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

Contents

PREFACE xxiii

VAXTPU TUTORIAL SECTION

CHAPTER 1 OVERVIEW OF THE VAX TEXT PROCESSING UTILITY 1–1

1.1 WHAT IS VAXTPU? 1–1

1.2 WHAT IS DECWINDOWS VAXTPU? 1–2
1.2.1 DECwindows VAXTPU and DECwindows Features 1–2
1.2.2 DECwindows VAXTPU and the DECwindows User Interface Language 1–4

1.3 WHAT IS EVE? 1–4

1.4 THE VAXTPU LANGUAGE 1–5
1.4.1 VAXTPU Data Types 1–6
1.4.2 VAXTPU Language Declarations 1–7
1.4.3 VAXTPU Language Statements 1–7
1.4.4 VAXTPU Built-In Procedures 1–7
1.4.5 User-Written Procedures 1–8

1.5 TERMINALS SUPPORTED BY VAXTPU 1–8

1.6 INVOKING VAXTPU 1–9
1.6.1 Using EDIT/TPU Command Qualifiers 1–9
1.6.2 Using Startup Files 1–10

1.7 USING JOURNAL FILES 1–11
1.7.1 Buffer Change Journal File Naming Algorithm 1–12

1.8 LEARNING MORE ABOUT VAXTPU 1–13

CHAPTER 2 VAXTPU DATA TYPES 2–1

2.1 ARRAY 2–2

2.2 BUFFER 2–3

2.3 INTEGER 2–5

2.4 KEYWORD 2–5

2.5 LEARN 2–7

2.6 MARKER 2–8

2.7 PATTERN 2–11
2.7.1 Pattern Built-In Procedures 2–13
2.7.2 Keywords That Can Be Used to Build Patterns 2–14
2.7.3 Pattern Operators 2–15
2.7.3.1 + (Pattern Concatenation Operator) • 2–15
2.7.3.2 & (Pattern Linking Operator) • 2–15
2.7.3.3 | (Pattern Alternation Operator) • 2–16
2.7.3.4 @ (Partial Pattern Assignment Operator) • 2–17
2.7.3.5 Relational Operators • 2–18
2.7.4 Pattern Compilation and Execution 2–18
2.7.5 Searching 2–18
2.7.6 Anchoring a Search 2–19

2.8 PROCESS 2–20

2.9 PROGRAM 2–21

2.10 RANGE 2–21

2.11	STRING	2–23
2.12	UNSPECIFIED	2–24
2.13	WIDGET	2–24
2.14	WINDOW	2–25
2.14.1	Window Dimensions	2–25
2.14.2	Creating Windows	2–26
2.14.3	Window Values	2–27
2.14.4	Mapping Windows	2–27
2.14.5	Removing Windows	2–28
2.14.6	Screen Manager	2–28
2.14.7	Getting Information on Windows	2–29
2.14.8	Terminals That Do Not Support Windows	2–29
CHAPTER 3	LEXICAL ELEMENTS OF THE VAXTPU LANGUAGE	3–1
3.1	OVERVIEW	3–1
3.2	CHARACTER SET	3–1
3.2.1	Entering Control Characters	3–2
3.2.2	VAXTPU Symbols	3–3
3.3	IDENTIFIERS	3–4
3.4	VARIABLES	3–4
3.5	CONSTANTS	3–5
3.6	OPERATORS	3–6
3.7	EXPRESSIONS	3–8
3.7.1	Arithmetic Expressions	3–9
3.7.2	Relational Expressions	3–10
3.7.3	Pattern Expressions	3–11
3.7.4	Boolean Expressions	3–11

3.8	**RESERVED WORDS**	**3–12**
3.8.1	**Keywords**	**3–12**
3.8.2	**Built-In Procedure Names**	**3–12**
3.8.3	**Predefined Constants**	**3–13**
3.8.4	**Declarations and Statements**	**3–13**
3.8.4.1	The Module Declaration • **3–14**	
3.8.4.2	The Procedure Declaration • **3–15**	
	3.8.4.2.1 Procedure Names • 3–16	
	3.8.4.2.2 Procedure Parameters • 3–16	
	3.8.4.2.3 Procedures That Return a Result • 3–19	
	3.8.4.2.4 Recursive Procedures • 3–19	
	3.8.4.2.5 Local Variables • 3–20	
	3.8.4.2.6 Constants • 3–20	
	3.8.4.2.7 ON_ERROR Statements • 3–21	
3.8.4.3	The Assignment Statement • **3–21**	
3.8.4.4	The Repetitive Statement • **3–21**	
3.8.4.5	The Conditional Statement • **3–22**	
3.8.4.6	The Case Statement • **3–23**	
3.8.4.7	Error Handling • **3–25**	
	3.8.4.7.1 Procedural Error Handlers • 3–26	
	3.8.4.7.2 Case-Style Error Handlers • 3–28	
	3.8.4.7.3 CTRL/C Handling • 3–31	
3.8.4.8	The RETURN Statement • **3–31**	
3.8.4.9	The ABORT Statement • **3–33**	
3.8.4.10	Miscellaneous Declarations • **3–33**	
	3.8.4.10.1 EQUIVALENCE Statement • 3–33	
	3.8.4.10.2 LOCAL • 3–34	
	3.8.4.10.3 CONSTANT • 3–35	
	3.8.4.10.4 VARIABLE • 3–36	
3.9	**LEXICAL KEYWORDS**	**3–36**
3.9.1	**Conditional Compilation**	**3–36**
3.9.2	**Specifying the Radix of Numeric Constants**	**3–37**

CHAPTER 4 VAXTPU PROGRAM DEVELOPMENT **4–1**

4.1	**CREATING VAXTPU PROGRAMS**	**4–1**
4.1.1	**Simple Programs**	**4–2**
4.1.2	**Complex Programs**	**4–2**
4.1.3	**Program Syntax**	**4–3**

4.2	**PROGRAMMING IN DECWINDOWS VAXTPU**	**4–5**
4.2.1	**Widgets Supported by DECwindows VAXTPU**	**4–5**
4.2.2	**Input Focus Support in DECwindows VAXTPU**	**4–5**
4.2.3	**Global Selection Support in DECwindows VAXTPU**	**4–6**
4.2.3.1	Difference Between Global Selection and Clipboard • **4–6**	
4.2.3.2	Handling of Multiple Global Selections • **4–6**	
4.2.3.3	Relation of Global Selection to Input Focus in DECwindows VAXTPU • **4–7**	
4.2.3.4	DECwindows VAXTPU's Response to Requests for Information About the Global Selection • **4–7**	
4.2.4	**Using Callbacks in DECwindows VAXTPU**	**4–8**
4.2.4.1	Background on DECwindows Callbacks • **4–8**	
4.2.4.2	Understanding the Difference Between VAXTPU's Internally-Defined Callback Routines and a Layered Application's Callback Routines • **4–9**	
4.2.4.3	Using Internally-Defined VAXTPU Callback Routines with UIL • **4–9**	
4.2.4.4	Using Internally-Defined VAXTPU Callback Routines with Widgets Not Defined by UIL • **4–10**	
4.2.4.5	Using Application-Level Callback Action Routines • **4–10**	
4.2.4.6	Callable Interface-Level Callback Routines • **4–10**	
4.2.5	**Using Closures in DECwindows VAXTPU**	**4–11**
4.2.6	**Specifying Values for Widget Resources in DECwindows VAXTPU**	**4–12**
4.2.6.1	VAXTPU Data Types for Specifying Resource Values • **4–12**	
4.2.6.2	Specifying a List as a Resource Value • **4–13**	
4.3	**WRITING CODE COMPATIBLE WITH DECWINDOWS EVE**	**4–14**
4.3.1	**Screen Objects in Applications Layered on DECwindows VAXTPU**	**4–14**
4.3.2	**Select Ranges in DECwindows EVE**	**4–16**
4.3.2.1	Dynamic Selection • **4–17**	
4.3.2.2	Static Selection • **4–17**	
4.3.2.3	Found Range Selection • **4–18**	
4.3.2.4	Relation of EVE Selection to DECwindows Global Selection • **4–18**	
4.4	**COMPILING VAXTPU PROGRAMS**	**4–18**
4.4.1	**Compiling on the EVE Command Line**	**4–19**
4.4.2	**Compiling in a VAXTPU Buffer**	**4–19**

4.5	**EXECUTING VAXTPU PROGRAMS**	**4–19**
4.5.1	**Interrupting Execution with CTRL/C**	**4–20**
4.5.2	**Procedure Execution**	**4–21**
4.6	**VAXTPU STARTUP FILES**	**4–21**
4.6.1	**Sequence in Which VAXTPU Processes Startup Files**	**4–22**
4.6.2	**Section Files**	**4–23**
4.6.2.1	Creating and Processing a New Section File • **4–23**	
4.6.2.2	Extending an Existing Section File • **4–24**	
4.6.2.3	A Sample Section File • **4–25**	
4.6.2.4	Recommended Conventions for Section Files • **4–28**	
	4.6.2.4.1 TPU$INIT_PROCEDURE • 4–28	
	4.6.2.4.2 TPU$LOCAL_INIT • 4–29	
	4.6.2.4.3 Special Variables • 4–29	
4.6.3	**Command Files**	**4–29**
4.6.4	**EVE Initialization Files**	**4–31**
4.6.4.1	Using an EVE Initialization File at Startup • **4–31**	
4.6.4.2	Using an EVE Initialization File During an Editing Session • **4–32**	
4.6.4.3	How an EVE Initialization File Affects Buffer Settings • **4–32**	
4.7	**DEBUGGING VAXTPU PROGRAMS**	**4–33**
4.7.1	**Invoking the VAXTPU Debugger**	**4–33**
4.7.1.1	Section Files • **4–34**	
4.7.1.2	Command Files • **4–34**	
4.7.1.3	Other VAXTPU Source Code • **4–35**	
4.7.2	**Getting Started with the VAXTPU Debugger**	**4–35**
4.7.3	**VAXTPU Debugger Commands**	**4–36**
4.8	**ERROR HANDLING**	**4–38**

CHAPTER 5 INVOKING VAXTPU **5–1**

5.1	**AVOIDING ERRORS RELATED TO VIRTUAL ADDRESS SPACE**	**5–1**
5.2	**INVOKING VAXTPU FROM A DCL COMMAND PROCEDURE**	**5–2**
5.2.1	**Setting Up a Special Editing Environment**	**5–2**
5.2.2	**Creating a Noninteractive Application**	**5–3**

5.3	INVOKING VAXTPU FROM A BATCH JOB	5–5
5.4	QUALIFIERS TO THE DCL COMMAND EDIT/TPU	5–5
5.4.1	/COMMAND	5–6
5.4.2	/CREATE	5–7
5.4.3	/DEBUG	5–8
5.4.4	/DISPLAY	5–8
5.4.5	/INITIALIZATION	5–9
5.4.6	/INTERFACE	5–10
5.4.7	/JOURNAL	5–10
5.4.8	/MODIFY	5–12
5.4.9	/OUTPUT	5–12
5.4.10	/READ_ONLY	5–13
5.4.11	/RECOVER	5–14
5.4.12	/SECTION	5–16
5.4.13	/START_POSITION	5–17
5.4.14	/WRITE	5–17
5.5	HOW EVE USES /MODIFY, /OUTPUT, /READ_ONLY, AND /WRITE	5–18
5.6	SPECIFYING A PARAMETER TO EDIT/TPU	5–19

CHAPTER 6 VAXTPU SCREEN MANAGEMENT 6–1

6.1	HOW THE SCREEN MANAGER HANDLES WINDOWS AND BUFFERS	6–1
6.1.1	Buffer Changes	6–1
6.1.2	Window Changes	6–2
6.1.2.1	Making a Window Current • 6–2	
6.1.2.2	Mapping a Window • 6–3	
6.1.2.3	Shifting a Window • 6–3	
6.1.2.4	Deleting a Window • 6–4	
6.1.2.5	How VAXTPU Window Size Affects a Terminal Emulator • 6–4	
6.1.2.6	How VAXTPU Window Size Affects the Display on a Terminal • 6–4	
6.1.2.7	How a Window Displays Insertion of Records into a Buffer • 6–5	
6.1.2.8	How a Window Displays Deletion of Records from a Buffer • 6–5	
6.1.2.9	How a Window Displays Changes to a Record in a Buffer • 6–6	

6.2	INVOKING THE SCREEN MANAGER	6–6
6.2.1	Enabling Screen Updates	6–6
6.2.2	Automatic Updates	6–7
6.2.3	Updating Windows	6–8
6.2.4	Updating the Whole Screen	6–9
6.2.5	The REFRESH Built-In	6–10
6.2.6	The SCROLL Built-In	6–10
6.3	CURSOR POSITION COMPARED TO EDITING POINT	6–10
6.4	BUILT-IN PADDING	6–11

VAXTPU REFERENCE SECTION

CHAPTER 7 VAXTPU BUILT-IN PROCEDURES 7–1

7.1	BUILT-IN PROCEDURES GROUPED ACCORDING TO FUNCTION	7–1
7.1.1	Screen Layout	7–1
7.1.2	Cursor Movement	7–2
7.1.3	Moving the Editing Position	7–3
7.1.4	Text Manipulation	7–3
7.1.5	Pattern Matching	7–5
7.1.6	Status of the Editing Context	7–6
7.1.7	Defining Keys	7–8
7.1.8	Multiple Processing	7–9
7.1.9	Program Execution	7–10
7.1.10	DECwindows VAXTPU-Specific	7–10
7.1.11	Miscellaneous	7–13
7.2	DESCRIPTIONS OF THE BUILT-IN PROCEDURES	7–15
	ABORT	7–16
	ADD_KEY_MAP	7–17
	ADJUST_WINDOW	7–19
	ANCHOR	7–24
	ANY	7–26
	APPEND_LINE	7–28
	ARB	7–30
	ASCII	7–32

ATTACH	7–35
BEGINNING_OF	7–37
BREAK	7–39
CALL_USER	7–40
CHANGE_CASE	7–44
COMPILE	7–47
CONVERT	7–50
COPY_TEXT	7–53
CREATE_ARRAY	7–55
CREATE_BUFFER	7–58
CREATE_KEY_MAP	7–63
CREATE_KEY_MAP_LIST	7–65
CREATE_PROCESS	7–67
CREATE_RANGE	7–69
CREATE_WIDGET	7–72
CREATE_WINDOW	7–77
CURRENT_BUFFER	7–80
CURRENT_CHARACTER	7–81
CURRENT_COLUMN	7–83
CURRENT_DIRECTION	7–85
CURRENT_LINE	7–86
CURRENT_OFFSET	7–88
CURRENT_ROW	7–90
CURRENT_WINDOW	7–92
CURSOR_HORIZONTAL	7–94
CURSOR_VERTICAL	7–96
DEBUG_LINE	7–99
DEFINE_KEY	7–100
DEFINE_WIDGET_CLASS	7–105
DELETE	7–107
EDIT	7–111
END_OF	7–115
ERASE	7–117
ERASE_CHARACTER	7–119
ERASE_LINE	7–121
ERROR	7–123
ERROR_LINE	7–125
ERROR_TEXT	7–127
EXECUTE	7–129
EXIT	7–133
EXPAND_NAME	7–135
FAO	7–138

FILE_PARSE	7–140
FILE_SEARCH	7–143
FILL	7–146
GET_CLIPBOARD	7–149
GET_DEFAULT	7–151
GET_GLOBAL_SELECT	7–153
GET_INFO	7–156
GET_INFO (ANY_KEYNAME)	7–162
GET_INFO (ANY_KEYWORD)	7–164
GET_INFO (ANY_VARIABLE)	7–165
GET_INFO (ARRAY)	7–166
GET_INFO (ARRAY_VARIABLE)	7–167
GET_INFO (BUFFER)	7–169
GET_INFO (BUFFER_VARIABLE)	7–170
GET_INFO (COMMAND_LINE)	7–176
GET_INFO (DEBUG)	7–179
GET_INFO (DEFINED_KEY)	7–181
GET_INFO (INTEGER_VARIABLE)	7–182
GET_INFO (KEY_MAP)	7–183
GET_INFO (KEY_MAP_LIST)	7–184
GET_INFO (MARKER_VARIABLE)	7–185
GET_INFO (MOUSE_EVENT_KEYWORD)	7–188
GET_INFO (PROCEDURES)	7–190
GET_INFO (PROCESS)	7–191
GET_INFO (PROCESS_VARIABLE)	7–192
GET_INFO (RANGE_VARIABLE)	7–193
GET_INFO (SCREEN)	7–194
GET_INFO (STRING_VARIABLE)	7–204
GET_INFO (SYSTEM)	7–206
GET_INFO (WIDGET)	7–210
GET_INFO (WIDGET_VARIABLE)	7–215
GET_INFO (WINDOW)	7–219
GET_INFO (WINDOW_VARIABLE)	7–220
HELP_TEXT	7–229
INDEX	7–231
INT	7–233
JOURNAL_CLOSE	7–235
JOURNAL_OPEN	7–236
KEY_NAME	7–239
LAST_KEY	7–243
LEARN_ABORT	7–244
LEARN_BEGIN AND LEARN_END	7–245
LENGTH	7–248
LINE_BEGIN	7–250

LINE_END	7–252
LOCATE_MOUSE	7–253
LOOKUP_KEY	7–255
MANAGE_WIDGET	7–259
MAP	7–260
MARK	7–262
MATCH	7–265
MESSAGE	7–267
MESSAGE_TEXT	7–271
MODIFY_RANGE	7–274
MOVE_HORIZONTAL	7–279
MOVE_TEXT	7–281
MOVE_VERTICAL	7–283
NOTANY	7–285
PAGE_BREAK	7–287
POSITION	7–288
QUIT	7–292
READ_CHAR	7–294
READ_CLIPBOARD	7–296
READ_FILE	7–298
READ_GLOBAL_SELECT	7–300
READ_KEY	7–302
READ_LINE	7–304
REALIZE_WIDGET	7–307
RECOVER_BUFFER	7–308
REFRESH	7–311
REMAIN	7–313
REMOVE_KEY_MAP	7–314
RETURN	7–316
SAVE	7–317
SCAN	7–320
SCANL	7–323
SCROLL	7–325
SEARCH	7–328
SEARCH_QUIETLY	7–333
SELECT	7–338
SELECT_RANGE	7–341
SEND	7–343
SEND_CLIENT_MESSAGE	7–345
SEND_EOF	7–347
SET	7–348
SET (ACTIVE_AREA)	7–351

SET (AUTO_REPEAT) 7–354
SET (BELL) 7–356
SET (CLIENT_MESSAGE) 7–358
SET (COLUMN_MOVE_VERTICAL) 7–360
SET (CROSS_WINDOW_BOUNDS) 7–362
SET (DEBUG) 7–363
SET (DEFAULT_DIRECTORY) 7–367
SET (DETACHED_ACTION) 7–368
SET (DISPLAY_VALUE) 7–371
SET (DRM_HIERARCHY) 7–372
SET (ENABLE_RESIZE) 7–373
SET (EOB_TEXT) 7–375
SET (ERASE_UNMODIFIABLE) 7–376
SET (FACILITY_NAME) 7–379
SET (FORWARD) 7–380
SET (GLOBAL_SELECT) 7–381
SET (GLOBAL_SELECT_GRAB) 7–383
SET (GLOBAL_SELECT_READ) 7–386
SET (GLOBAL_SELECT_TIME) 7–388
SET (GLOBAL_SELECT_UNGRAB) 7–390
SET (HEIGHT) 7–392
SET (ICON_NAME) 7–393
SET (ICON_PIXMAP) 7–394
SET (ICONIFY_PIXMAP) 7–396
SET (INFORMATIONAL) 7–398
SET (INPUT_FOCUS) 7–399
SET (INPUT_FOCUS_GRAB) 7–401
SET (INPUT_FOCUS_UNGRAB) 7–403
SET (INSERT) 7–405
SET (JOURNALING) 7–406
SET (KEYSTROKE_RECOVERY) 7–409
SET (KEY_MAP_LIST) 7–411
SET (LEFT_MARGIN) 7–413
SET (LEFT_MARGIN_ACTION) 7–415
SET (LINE_NUMBER) 7–417
SET (MAPPED_WHEN_MANAGED) 7–419
SET (MARGINS) 7–420
SET (MAX_LINES) 7–422
SET (MENU_POSITION) 7–423
SET (MESSAGE_ACTION_LEVEL) 7–425
SET (MESSAGE_ACTION_TYPE) 7–427
SET (MESSAGE_FLAGS) 7–428

SET (MODIFIABLE) 7–430
SET (MODIFIED) 7–432
SET (MOUSE) 7–433
SET (NO_WRITE) 7–435
SET (OUTPUT_FILE) 7–436
SET (OVERSTRIKE) 7–437
SET (PAD) 7–438
SET (PAD_OVERSTRUCK_TABS) 7–440
SET (PERMANENT) 7–442
SET (POST_KEY_PROCEDURE) 7–443
SET (PRE_KEY_PROCEDURE) 7–445
SET (PROMPT_AREA) 7–447
SET (RECORD_ATTRIBUTE) 7–449
SET (RESIZE_ACTION) 7–452
SET (REVERSE) 7–454
SET (RIGHT_MARGIN) 7–455
SET (RIGHT_MARGIN_ACTION) 7–457
SET (SCREEN_LIMITS) 7–459
SET (SCREEN_UPDATE) 7–461
SET (SCROLL_BAR) 7–463
SET (SCROLL_BAR_AUTO_THUMB) 7–466
SET (SCROLLING) 7–468
SET (SELF_INSERT) 7–471
SET (SHIFT_KEY) 7–473
SET (SPECIAL_ERROR_SYMBOL) 7–475
SET (STATUS_LINE) 7–477
SET (SUCCESS) 7–480
SET (SYSTEM) 7–481
SET (TAB_STOPS) 7–482
SET (TEXT) 7–484
SET (TIMER) 7–487
SET (TRACEBACK) 7–489
SET (UNDEFINED_KEY) 7–491
SET (VIDEO) 7–493
SET (WIDGET) 7–495
SET (WIDGET_CALL_DATA) 7–497
SET (WIDGET_CALLBACK) 7–500
SET (WIDTH) 7–502
SHIFT 7–504
SHOW 7–506
SLEEP 7–509
SPAN 7–511

SPANL 7–513
SPAWN 7–516
SPLIT_LINE 7–519
STR 7–521
SUBSTR 7–524
TRANSLATE 7–527
UNANCHOR 7–531
UNDEFINE_KEY 7–533
UNMANAGE_WIDGET 7–535
UNMAP 7–537
UPDATE 7–539
WRITE_CLIPBOARD 7–541
WRITE_FILE 7–544
WRITE_GLOBAL_SELECT 7–547

APPENDIX A SAMPLE VAXTPU PROCEDURES A–1

A.1 LINE-MODE EDITOR A–1

A.2 TRANSLATION OF CONTROL CHARACTERS A–2

A.3 RESTORING TERMINAL WIDTH BEFORE EXITING FROM VAXTPU A–5

A.4 RUNNING VAXTPU FROM A SUBPROCESS A–5

APPENDIX B SAMPLE DECWINDOWS VAXTPU PROCEDURES B–1

B.1 USING DECWINDOWS VAXTPU BUILT-INS B–1

B.2 DISPLAYING A DIALOG BOX B–1

B.3 CREATING A "MOUSE PAD" B–4

B.4 IMPLEMENTING AN EDT-STYLE APPEND COMMAND B–11

B.5 TESTING AND RETURNING A SELECT RANGE B–13

B.6 RESIZING WINDOWS B–16

B.7 UNMAPPING SAVED WINDOWS B–19

B.8 MAPPING SAVED WINDOWS B–22

B.9 HANDLING CALLBACKS FROM A SCROLL BAR WIDGET B–25

B.10 IMPLEMENTING THE COPY SELECTION OPERATION B–28

B.11 REACTIVATING A SELECT RANGE B–30

B.12 COPYING SELECTED MATERIAL FROM EVE TO ANOTHER DECWINDOWS APPLICATION B–31

APPENDIX C VAXTPU TERMINAL SUPPORT C–1

C.1 SCREEN-ORIENTED EDITING ON SUPPORTED TERMINALS C–1

C.1.1 Terminal Settings That Affect VAXTPU C–1

C.1.2 The DCL Command SET TERMINAL C–3

C.2 LINE-MODE EDITING ON UNSUPPORTED TERMINALS C–3

C.3 TERMINAL WRAP C–4

APPENDIX D VAXTPU MESSAGES D–1

APPENDIX E DEC MULTINATIONAL CHARACTER SET E–1

APPENDIX F VAXTPU FILE SUPPORT F–1

APPENDIX G EVE$BUILD MODULE G–1

G.1 HOW TO PREPARE CODE FOR USE WITH EVE$BUILD G–1
G.1.1 Module Identifiers G–2
G.1.2 Parsers G–3
G.1.3 Initialization G–4
G.1.4 Command Synonyms G–5
G.1.5 Status Line Fields G–7
G.1.6 Exit and Quit Handlers G–8
G.1.7 How to Invoke EVE$BUILD G–10

G.2 WHAT HAPPENS WHEN YOU USE EVE$BUILD G–11

INDEX

EXAMPLES

1–1 Sample User-Written Procedure 1–8
2–1 Suppressing the Addition of Padding Blanks 2–11
3–1 Global and Local Variable Declarations 3–5
3–2 Global and Local Constant Declarations 3–6
3–3 A Procedure Using Relational Operators on Markers 3–11
3–4 Simple Procedure with Parameters 3–17
3–5 Complex Procedure with Optional Parameters 3–18
3–6 Procedure That Returns a Result 3–19
3–7 Procedure Within Another Procedure 3–19
3–8 Recursive Procedure 3–20
3–9 Procedure Using the CASE Statement 3–24
3–10 Procedure Using the ON_ERROR Statement 3–27
3–11 Procedure with a Case-Style Error Handler 3–29
3–12 Procedure That Returns a Value 3–32
3–13 Procedure Returning a Status 3–32
3–14 Using RETURN in an ON_ERROR Section 3–33

3–15	Simple Error Handler	3–33
4–1	SHOW (SUMMARY) Display	4–2
4–2	Syntax of a VAXTPU Program	4–3
4–3	Sample VAXTPU Programs	4–4
4–4	Sample Program for a Section File	4–25
4–5	Source Code for Minimal Interface	4–26
4–6	Command File for Go to Text Marker	4–30
4–7	SHOW DEFAULTS BUFFER Display	4–33
5–1	DCL Command Procedure FILENAME.COM	5–3
5–2	DCL Command Procedure FORTRAN_TS.COM	5–3
5–3	DCL Command Procedure INVISIBLE_TPU.COM	5–4
5–4	VAXTPU Command File GSR.TPU	5–4
7–1	Initialization Procedure Using Variants of the SET Built-In	7–385
B–1	EVE Procedure That Displays a Selection Dialog Box	B–2
B–2	Procedure That Creates a "Mouse Pad"	B–4
B–3	EVE Procedure That Implements a Variant of the EDT APPEND Command	B–12
B–4	EVE Procedure That Returns a Select Range	B–14
B–5	Procedure That Resizes Windows	B–17
B–6	EVE Procedure That Unmaps Saved Windows	B–20
B–7	Procedure That Maps Saved Windows	B–23
B–8	EVE Procedure That Handles Callbacks from a Scroll Bar Widget	B–26
B–9	EVE Procedure That Implements the COPY SELECTION Operation	B–29
B–10	EVE Procedure That Reactivates a Select Range	B–30
B–11	EVE Procedure That Implements COPY SELECTION	B–32
C–1	DCL Command Procedure for SET TERM/NOWRAP	C–4

FIGURES

1–1	VAXTPU as a Base for EVE	1–2
1–2	VAXTPU as a Base for User-Written Interfaces	1–5
4–1	Nomenclature of DECwindows VAXTPU Screen Objects	4–15
7–1	Screen Layout Before Using ADJUST_WINDOW	7–21
7–2	Screen Layout After Using ADJUST_WINDOW	7–22

TABLES

1–1	Qualifiers to the DCL Command EDIT/TPU	1–9
1–2	Journaling Behavior Established by EVE	1–12
2–1	Keywords Used for Key Names	2–6
3–1	VAXTPU Symbols	3–3
3–2	VAXTPU Operators	3–6
3–3	Operator Precedence	3–7
4–1	Correspondence Between VAXTPU Data Types and DECwindows Argument Data Types	4–12
4–2	Special VAXTPU Variables Requiring a Value from a Layered Application	4–29
5–1	Summary of How VAXTPU and the Application Layered on VAXTPU Relate to the Qualifiers to EDIT/TPU	5–5
7–1	CREATE_RANGE Keyword Parameters	7–69
7–2	GET_INFO Built-in Procedures by First Parameter	7–158
7–3	VAXTPU Keywords Representing Mouse Events	7–188
7–4	Detached Cursor Flag Constants	7–198
7–5	Valid Keywords for the Third Parameter When the Second Parameter is "Bottom", "Left", "Length", "Right", "Top", or "Width"	7–222
7–6	Message Flag Values	7–268
7–7	Message Flag Values	7–271
7–8	MODIFY_RANGE Keyword Parameters	7–274
7–9	VAXTPU Keywords Representing Mouse Events	7–352
7–10	Selected Built-in Actions When ERASE_UNMODIFIABLE is Turned Off	7–377
7–11	Message Codes for $PUTMSG System Service	7–428
7–12	Message Flag Values	7–428
C–1	Terminal Behavior That Affects VAXTPU's Performance	C–1
D–1	VAXTPU Messages and Their Severity Levels	D–1
E–1	DEC Multinational Character Set	E–1
F–1	VAXTPU Support of File Attributes	F–1

Preface

Intended Audience

This manual is intended for experienced programmers who know at least one computer language. Some features of VAXTPU, for example, the callable interface and the built-in procedure FILE_PARSE, are intended for system programmers who have a good understanding of VMS system concepts. Relevant documents about the VMS operating system are listed under Associated Documents.

Document Structure

This manual consists of six expository chapters, a reference section, and seven appendixes. The six chapters discuss the following topics:

- Chapter 1 contains an overview of VAXTPU.
- Chapter 2 provides detailed information on VAXTPU data types.
- Chapter 3 discusses the lexical elements of VAXTPU. These include the character set, identifiers, variables, constants, and reserved words, such as VAXTPU language statements.
- Chapter 4 describes VAXTPU program development.
- Chapter 5 describes how to invoke VAXTPU.
- Chapter 6 discusses the VAXTPU screen manager and screen management issues.

The VAXTPU Reference Section (Chapter 7) provides detailed descriptions of the VAXTPU built-in procedures.

The seven appendixes are organized as follows:

- Appendix A contains sample procedures written in VAXTPU.
- Appendix B contains sample procedures written in DECwindows VAXTPU.
- Appendix C describes terminals supported by VAXTPU.
- Appendix D lists each VAXTPU message, its abbreviation, and its severity level.
- Appendix E contains the DEC Multinational Character Set.
- Appendix F lists the file types that VAXTPU supports.
- Appendix G discusses EVE$BUILD, a tool that enables you to layer applications onto EVE or build new VAXTPU applications.

Associated Documents

To learn how to use the Extensible VAX Editor (EVE), see the *Guide to VMS Text Processing*. For reference information on EVE commands, see *VMS EVE Reference Manual*.

The *VMS Utility Routines Manual* contains a chapter presenting the VAXTPU callable interface.

The *VMS System Messages and Recovery Procedures Reference Manual* contains the VAXTPU messages, as well as an explanation and suggested user action for each message. The messages are listed alphabetically by the abbreviation for the message text.

The *Overview of VMS Documentation* briefly describes all VMS system documentation, defining the intended audience for each manual and providing a synopsis of each manual's contents.

The *VMS DCL Dictionary* describes the VMS DCL commands that help you create, copy, and print files containing VAXTPU programs.

The *VMS System Services Volume* describes system services.

The *Introduction to VMS System Routines* and *VMS Utility Routines Manual* describe utility routines.

The *VMS Run-Time Library Routines Volume* describes routines of the run-time library.

The *VMS Record Management Services Manual* describes VMS RMS services.

Conventions

The following conventions are used in this document:

mouse	The term *mouse* is used to refer 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. (The buttons can be redefined by the user.)
Return	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the Return key. (Note that the Return key is not usually shown in syntax statements or in all examples; however, assume that you must press the Return key after entering a command or responding to a prompt.)

CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
red ink	Red ink indicates information that you must enter from the keyboard or a screen object that you must choose or click on. For online versions, user input is shown in **bold**.
. . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.
{ }	Braces enclose a mandatory portion of the format of a built-in procedure or lexical element. When braces enclose a stacked list of items, you must choose one of the items. For example: { string / range }
[[ ]]	Double brackets in examples show an optional portion of the format of a built-in procedure or lexical element. When double brackets enclose an item or series of items, you can select one of the items. For example: [[string / range]]
[[, . . .]]	Double brackets enclosing a comma and horizontal ellipsis mean that you can repeat the preceding item one or more times, separating two or more items with commas. For example: `parameter [[,...]]`
[]	Delimits a case label. Single brackets do not indicate optional parameters in this manual.
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

UPPERCASE letters and special symbols	Uppercase letters and special symbols in syntax descriptions and sample procedures indicate VAXTPU reserved words and predeclared identifiers, and other user input that must be typed exactly as shown. For example:
	`PROCEDURE`
	`UNDERLINE`
	String constants are shown in lowercase to emphasize that they are strings. However, they, too, must be typed exactly as shown.

lowercase letters	Lowercase letters in syntax descriptions and sample procedures represent elements that you must replace according to the description in the text. For example, when a data type, such as buffer, is used in a syntax example, replace it with the variable name assigned to the data item when it was created. In the following assignment statement, *my_buffer_variable* is the variable name assigned to the buffer you are creating:
	
	`my_buffer_variable :=`
	`CREATE_BUFFER ('my_buf_name', 'my_file_name')`
	
	To specify a buffer as a parameter for a VAXTPU built-in procedure, use the variable for the buffer. For example, to erase the contents of the buffer created in the preceding statement, enter the following:
	
	`ERASE (my_buffer_variable)`

user_	Many of the sample procedures in this manual have the prefix *user_* as a part of the procedure name. Digital suggests that you replace the prefix *user* with your initials. This or some other convention helps to ensure that the variables and procedure names that you create do not conflict with either VAXTPU built-in procedure names, or the procedure names and variables of your editing interface.
filespec	Mnemonic for file specification.

VAXTPU Reference Section

This section contains detailed descriptions of the built-in procedures provided by the VAX Text Processing Utility.

7 VAXTPU Built-In Procedures

This chapter describes each of the VAXTPU built-in procedures. The chapter is divided into two sections.

In Section 7.1, the built-in procedures are grouped according to the functions that they perform so you can see at a glance which built-in is related to what task. In Section 7.2, the built-in procedures are listed alphabetically. Each built-in is described in detail.

Some built-in procedures do not return useful values. The descriptions of these built-ins do not show a return value in the format section. However, these built-ins return 0 when used on the right-hand side of an assignment statement.

Some entries in this chapter describe language elements or keywords that are not built-in procedures. These elements and keywords are included in this chapter because they are used in the same way built-ins are used.

7.1 Built-In Procedures Grouped According to Function

When you want to perform editing tasks, use the following lists to help you identify which built-in procedures are related to a particular task. For more information about a built-in procedure, see its individual description in Section 7.2.

7.1.1 Screen Layout

- ADJUST_WINDOW (window, integer1, integer2)
- CREATE_WINDOW (integer1, integer2, { , ON / , OFF })
- MAP ({ window, buffer / widget })
- REFRESH
- SET (DISPLAY_VALUE,window,display_value_integer)
- SET (HEIGHT, SCREEN, length)
- SET (PAD, window { , ON / , OFF })
- SET (PROMPT_AREA, integer1, integer2 { , NONE / , BOLD / , BLINK / , REVERSE / , UNDERLINE })

- SET (SCREEN_UPDATE { , ON / , OFF })
- SET (SCROLLING, window { , ON / , OFF } integer1, integer2, integer3)
- SET (STATUS_LINE, window { , NONE / , BOLD / , BLINK / , REVERSE / , SPECIAL_GRAPHICS / , UNDERLINE } , string)
- SET (TEXT, { widget, string / window, { BLANK_TABS / GRAPHIC_TABS / NO_TRANSLATE } })
- SET (VIDEO, window { , NONE / , BOLD / , BLINK / , REVERSE / , UNDERLINE })
- SET (WIDTH, { window / ALL / SCREEN }, width_int)
- SHIFT (window, integer1)
- UNMAP ({ widget / window })
- UPDATE ({ ALL / window })

7.1.2 Cursor Movement

- CURSOR_HORIZONTAL (integer1)
- CURSOR_VERTICAL (integer1)
- SCROLL (window [[,integer1]])
- SET (COLUMN_MOVE_VERTICAL { , ON / , OFF })
- SET (CROSS_WINDOW_BOUNDS { , ON / , OFF })
- SET (DETACHED_ACTION, SCREEN [[, { buffer / learn / program / range / string }]])

7.1.3 Moving the Editing Position

- MOVE_HORIZONTAL (integer)
- MOVE_VERTICAL (integer)
- POSITION ({ buffer | BUFFER_BEGIN | BUFFER_END | integer | LINE_BEGIN | LINE_END | marker | MOUSE | range | window })

7.1.4 Text Manipulation

- APPEND_LINE
- BEGINNING_OF ({ buffer | range })
- CHANGE_CASE ({ buffer | range | string }, { INVERT | LOWER | UPPER } [[, IN_PLACE | , NOT_IN_PLACE]])
- COPY_TEXT ({ buffer | range1 | string })
- CREATE_BUFFER (string1 [[,string2 [[,buffer1]] [[,string3]]]])
- CREATE_RANGE ({ marker1 | delimiting_keyword }, { marker2 | delimiting_keyword } [[, attribute_keyword]])
- EDIT ({ buffer | range | string }, keyword1[[,...]] [[,keyword2]] [[,keyword3]])
- END_OF ({ buffer | range })
- ERASE ({ buffer | range })
- ERASE_CHARACTER (integer)
- ERASE_LINE

- FILE_PARSE (filespec [[,string1 [[,string2
  [[, NODE]] [[, DEVICE]] [[, DIRECTORY]] [[, NAME]]
  [[, TYPE]] [[, VERSION]]]])
- FILE_SEARCH (filespec [[, string1 [[, string2
  [[, NODE]] [[, DEVICE]] [[, DIRECTORY]] [[, NAME]]
  [[, TYPE]] [[, VERSION]]]])
- FILL ({ buffer / range } [[, string [[, integer1 [[, integer2 [[, integer3]]]]]]])
- MARK ({ BLINK / BOLD / NONE / FREE_CURSOR / REVERSE / UNDERLINE })
- MESSAGE_TEXT ({ integer1 / keyword } [[, integer2 [[,FAO-parameter]]]])
- MODIFY_RANGE (range, { marker1 / delimiting_keyword },
  { marker2 / delimiting_keyword }
  [[, attribute_keyword]])
- MOVE_TEXT ({ buffer / range1 / string })
- READ_FILE (string1)
- SEARCH ({ ANCHOR / LINE_BEGIN / LINE_END / PAGE_BREAK / pattern / REMAIN / string / UNANCHOR } { , FORWARD / , REVERSE }
  [[{ , EXACT / , NO_EXACT / , integer } [[{ , buffer / , range1 }]]])
- SEARCH_QUIETLY ({ ANCHOR / LINE_BEGIN / LINE_END / PAGE_BREAK / pattern / REMAIN / string / UNANCHOR } { , FORWARD / , REVERSE }
  [[{ , EXACT / , NO_EXACT / , integer } [[{ , buffer / , range1 }]]])

- SELECT ({ BLINK / BOLD / NONE / REVERSE / UNDERLINE })
- SELECT_RANGE
- SET (ERASE_UNMODIFIABLE, buffer, { ON / OFF })
- SET (MODIFIABLE, buffer { , ON / , OFF })
- SET (MODIFIED, buffer, { ON / OFF })
- SPLIT_LINE
- TRANSLATE ({ buffer / range / string }, string2, string3 [[, { IN_PLACE / NOT_IN_PLACE }]])
- WRITE_FILE ({ buffer / range }, string1)

7.1.5 Pattern Matching

- ANCHOR
- ANY ({ buffer / range / string }, integer1)
- ARB (integer)
- LINE_BEGIN
- LINE_END
- MATCH ({ buffer / range / string })
- NOTANY ({ buffer / range / string }, integer1)
- PAGE_BREAK
- REMAIN
- SCAN ({ buffer / range / string } [[, { FORWARD / REVERSE }]])

- SCANL ({ buffer / range / string } [[, { FORWARD / REVERSE }]])
- SPAN ({ buffer / range / string } [[, { FORWARD / REVERSE }]])
- SPANL ({ buffer / range / string } [[, { FORWARD / REVERSE }]])
- UNANCHOR

7.1.6 Status of the Editing Context

- CURRENT_BUFFER
- CURRENT_CHARACTER
- CURRENT_COLUMN
- CURRENT_DIRECTION
- CURRENT_LINE
- CURRENT_OFFSET
- CURRENT_ROW
- CURRENT_WINDOW
- DEBUG_LINE
- ERROR
- ERROR_LINE
- ERROR_TEXT
- GET_INFO (parameter1, parameter2 [[, . . .]])
- RECOVER_BUFFER (old_buffer_name [[, journal_file_name / , template_buffer]])
- LOCATE_MOUSE (window, x_integer, y_integer)
- SET (AUTO_REPEAT { , ON / , OFF })
- SET (BELL { , ALL / , BROADCAST } { , ON / , OFF })
- SET (DEFAULT_DIRECTORY, new_default_string)
- SET (DEBUG [[, ON / , OFF / , PROGRAM]] [[, ALL / , buffer / , program / , range / , string]] [[,value]])
- SET (FACILITY_NAME, string)

- SET (FORWARD, buffer)
- SET (INFORMATIONAL { , ON / , OFF })
- SET (INSERT, buffer)
- SET (JOURNALING, buffer { ON / OFF } [[,file_name_string]])

  or

  SET (JOURNALING, integer)
- SET (KEYSTROKE_RECOVERY { ON / OFF })
- SET (LEFT_MARGIN, buffer, integer)
- SET (LEFT_MARGIN_ACTION, buffer1 [[, buffer2 / , learn_sequence / , program / , range / , string]])
- SET (LINE_NUMBER { , ON / , OFF })
- SET (MARGINS, buffer, integer1, integer2)
- SET (MAX_LINES, buffer, integer)
- SET (MESSAGE_ACTION_LEVEL, { integer / keyword })
- SET (MESSAGE_ACTION_TYPE, { NONE / BELL / REVERSE })
- SET (MESSAGE_FLAGS, integer)
- SET (MOUSE, { ON / OFF })
- SET (NO_WRITE, buffer [[, ON / , OFF]])
- SET (OUTPUT_FILE, buffer, string)
- SET (OVERSTRIKE, buffer)
- SET (PAD_OVERSTRUCK_TABS { , ON / , OFF })
- SET (RECORD_ATTRIBUTE, { mark / range / buffer }, { DISPLAY_VALUE / LEFT_MARGIN }, { display_setting_integer / margin_setting_integer })

  or

  SET (RECORD_ATTRIBUTE, [[marker, / range, / buffer,]] MODIFIABLE, { ON / OFF })

- SET (PERMANENT, buffer)
- SET (REVERSE, buffer)
- SET (RIGHT_MARGIN, buffer, integer)
- SET (RIGHT_MARGIN_ACTION, buffer1 [[, buffer2 | , learn_sequence | , program | , range | , string]])
- SET (SPECIAL_ERROR_SYMBOL, string)
- SET (SUCCESS { , ON | , OFF })
- SET (SYSTEM, buffer)
- SET (TAB_STOPS, buffer { , integer | , string })
- SET (TIMER { , ON | , OFF } [[, string]])
- SET (TRACEBACK { , ON | , OFF })
- SHOW ({ BUFFER[S] | KEY_MAP_LIST[[S]] | KEY_MAP[[S]] | KEYWORDS | PROCEDURES | SCREEN | SUMMARY | VARIABLES | WINDOW[[S]] | buffer | string | window })

7.1.7 Defining Keys

- ADD_KEY_MAP (key-map-list-name { , "first" | , "last" }, key-map-name [[,...]])
- CREATE_KEY_MAP (string1)
- CREATE_KEY_MAP_LIST (string1, string2 [[, . . .]])
- DEFINE_KEY ({ buffer | learn_sequence | program | range | string1 }, key-name [[, string2 [[,string3]]]])

- KEY_NAME ({ integer / key-name / string } [[{ , SHIFT_KEY / , SHIFT_MODIFIED / , ALT_MODIFIED / , CTRL_MODIFIED / , HELP_MODIFIED } [[, ...]]]] [[, FUNCTION / , KEYPAD]])
- LAST_KEY
- LOOKUP_KEY (key-name { , COMMENT / , KEY_MAP / , PROGRAM } [[, string1 / , string2]])
- REMOVE_KEY_MAP (string1, string2 [[, ALL]])
- SET (KEY_MAP_LIST, string [[, buffer, window]])
- SET (POST_KEY_PROCEDURE, string1 [[, buffer / , learn_sequence / , program / , range / , string2]])
- SET (PRE_KEY_PROCEDURE, string1 [[, buffer / , learn_sequence / , program / , range / , string2]])
- SET (SELF_INSERT, string, { , ON / , OFF })
- SET (SHIFT_KEY, keyword [[,string]])
- SET (UNDEFINED_KEY, string1 [[, buffer / , learn_sequence / , program / , range / , string2]])
- UNDEFINE_KEY (keyword [[, key-map-list-name / , key-map-name]])

7.1.8 Multiple Processing

- ATTACH [[({ integer / string })]]
- CREATE_PROCESS (buffer [[, string]])

- SEND ({ buffer / range / string }, process)
- SEND_EOF (process)
- SPAWN [[(string [[, ON / , OFF]])]]

7.1.9 Program Execution

- ABORT
- BREAK
- COMPILE ({ buffer / range / string })
- EXECUTE ({ buffer / key-name [[, key-map-list-name / , key-map-name]] / learn_sequence / program / range / string })
- RETURN
- SAVE (string1 [[,"NO_DEBUG_NAMES"]]
  [[,"NO_PROCEDURE_NAMES"]]
  [[,"IDENT", string2]])

7.1.10 DECwindows VAXTPU-Specific

- CREATE_WIDGET (widget_class, widget_name, { parent_widget / SCREEN }
  [[, { buffer / learn_sequence / program / range / string } [[, closure
  [[, widget_args...]]]]])
- CREATE_WIDGET (resource_manager_name, hierarchy_id,
  { parent_widget / SCREEN }
  [[, { buffer / learn_sequence / program / range / string }
  [[, closure
  [[, widget_args...]]]]])

- DEFINE_WIDGET_CLASS (class_name
  [[, creation_routine_name
  [[, creation_routine_image_name]]])
- DELETE (widget)
- GET_CLIPBOARD
- GET_DEFAULT (string1, string2)
- GET_GLOBAL_SELECT ({ PRIMARY | SECONDARY | selection_name }, selection_property_name)
- MANAGE_WIDGET (widget [[, widget...]])
- READ_CLIPBOARD
- READ_GLOBAL_SELECT ({ PRIMARY | SECONDARY | selection_name }, selection_property_name)
- REALIZE_WIDGET (widget)
- SEND_CLIENT_MESSAGE ({ STUFF_SELECTION | KILL_SELECTION })
- SET (ACTIVE_AREA, window, column, row [[, width, height]])
- SET (CLIENT_MESSAGE,SCREEN, { buffer | learn_sequence | program | range | string })
- SET (DRM_HIERARCHY, filespec [[, filespec...]])
- SET (ENABLE_RESIZE, { ON | OFF })
- SET (GLOBAL_SELECT, SCREEN, { PRIMARY | SECONDARY | selection_name })
- SET (GLOBAL_SELECT_GRAB, SCREEN
  [[, { buffer | learn_sequence | program | range | string | NONE }]])
- SET (GLOBAL_SELECT_READ, { buffer1 | SCREEN }
  [[, { buffer2 | learn_sequence | program | range | string | NONE }]])

- SET (GLOBAL_SELECT_TIME, SCREEN, { integer | string })
- SET (GLOBAL_SELECT_UNGRAB, SCREEN
  [[, { buffer | learn_sequence | program | range | string | NONE }]])
- SET (ICON_NAME, string)
- SET (ICON_PIXMAP,integer,icon_pixmap [[,widget]])

  or

  SET (ICON_PIXMAP,bitmap_file_name [[,widget]])
- SET (ICONIFY_PIXMAP,integer,icon_pixmap [[,widget]])

  or

  SET (ICONIFY_PIXMAP,bitmap_file_name [[,widget]])
- SET (INPUT_FOCUS [[, SCREEN | , widget]])
- SET (INPUT_FOCUS_GRAB, SCREEN [[, { buffer | learn_sequence | program | range | string | NONE }]])
- SET (INPUT_FOCUS_UNGRAB, SCREEN [[, { buffer | learn_sequence | program | range | string | NONE }]])
- SET (MAPPED_WHEN_MANAGED,widget, { ON | OFF })
- SET (MENU_POSITION, mouse_down_button, { widget | array | NONE })
- SET (RESIZE_ACTION [[, { buffer | learn_sequence | program | range | string | NONE }]])
- SET (SCREEN_LIMITS, array)

- SET (SCROLL_BAR, window, { HORIZONTAL, / VERTICAL, } { ON / OFF })
- SET (SCROLL_BAR_AUTO_THUMB, window, { HORIZONTAL / VERTICAL },
  { ON / OFF })
- SET (WIDGET, widget,
  { widget_args [[, widget_args...]] })
- SET (WIDGET_CALL_DATA, widget, reason_code, request_string,
  keyword [[, request_string, keyword...]])
- SET (WIDGET_CALLBACK, widget,
  { buffer, / learn_sequence, / program, / range, / string, } closure)
- UNMANAGE_WIDGET (widget [[, widget...]])
- WRITE_CLIPBOARD (clipboard_label, { buffer / range / string })
- WRITE_GLOBAL_SELECT ({ array / buffer / range / string / integer / NONE })

7.1.11 Miscellaneous

- ASCII ({ integer1 / keyword / string1 })
- CALL_USER (integer, string1)
- CONVERT ({ DECW_ROOT_WINDOW / SCREEN / window }, { CHARACTERS, / COORDINATES, }
  from_x_integer, from_y_integer,
  { DECW_ROOT_WINDOW / SCREEN / window }, { CHARACTERS, / COORDINATES, }
  to_x_integer, to_y_integer)
- CREATE_ARRAY [[(integer1 [[, integer2]])]]

- DELETE ({ array / buffer / integer / keyword / learn_sequence / marker / pattern / process / program / range / string / unspecified / window })
- EXIT
- EXPAND_NAME (string1 { , ALL / , KEYWORDS / , PROCEDURES / , VARIABLES })
- FAO (string1 [[, { integer1 / string3 } [[, ... { integer_*n* / string_*n* }]]])
- HELP_TEXT (library-file, topic { , ON / , OFF }, buffer)
- INDEX (string, substring)
- INT ({ integer1 / keyword / string })
- JOURNAL_CLOSE
- JOURNAL_OPEN (file-name)
- LEARN_ABORT
- LEARN_BEGIN ({ EXACT / NO_EXACT })
- LEARN_END
- LENGTH ({ buffer / range / string })
- MESSAGE (buffer, range [[, integer1]])
- MESSAGE ({ integer2 / keyword / string } [[, integer3 [[, FAO-parameter]]]])
- QUIT [[({ ON / OFF } [[, severity]])]]
- READ_CHAR
- READ_KEY
- READ_LINE [[(string1 [[, integer]])]]
- SET (EOB_TEXT, buffer, string)

- SLEEP ({ integer / string })
- STR (integer)
- STR ({ buffer / range } [[,string2]])
- STR ({ { buffer / range } [[, string2]] [[, ON / , OFF]] / string1 })
- SUBSTR ({ buffer / range / string1 }, integer1 [[, integer2]])

7.2 Descriptions of the Built-In Procedures

The discussion of each built-in procedure in this section is divided, as applicable, into the following parts:

- A short functional definition
- Format
- Parameters
- Description
- Signaled Errors listing the warnings and errors signaled, if applicable
- Examples

The built-in procedures are presented in alphabetical order.

ABORT

Stops any executing procedures and causes VAXTPU to wait for the next key press.

FORMAT

ABORT

PARAMETERS

None.

DESCRIPTION

ABORT returns control to VAXTPU'S main control loop. It causes an immediate exit from all invoked procedures.

Although ABORT behaves much like a built-in, it is actually a VAXTPU language element.

ABORT is evaluated for correct syntax at compile time. In contrast, VAXTPU procedures are usually evaluated for a correct parameter count and parameter types at execution time.

SIGNALED ERRORS

ABORT is a language element and has no completion codes.

EXAMPLE

```
ON_ERROR
   MESSAGE ("Aborting command because of error.");
   ABORT;
ENDON_ERROR;
```

This error handler does not try to recover from an error. Rather, it stops execution of the current procedure and returns to VAXTPU'S main loop.

ADD_KEY_MAP

Adds one or more key maps to a key map list.

FORMAT

ADD_KEY_MAP *(key-map-list-name,* { *"first",* / *"last",* } *key-map-name* [*, . . .]*)*

PARAMETERS

key-map-list-name

A string that specifies the name of the key map list.

"first"

A string directing VAXTPU to add the key map to the beginning of the key map list. In cases where a key is defined in multiple key maps, the first definition found for that key in any of the key maps in a key map list is used.

"last"

A string directing VAXTPU to add the key map to the end of the key map list. In cases where a key is defined in multiple key maps, the first definition found for that key in any of the key maps in a key map list is used.

key-map-name

A string that specifies the name of the key map to be added to the key map list. You can specify more than one key map. Key maps are added to the key map list in the order specified. The order of a key map in a key map list determines precedence among any conflicting key definitions.

DESCRIPTION

The built-in procedure ADD_KEY_MAP adds key maps to key map lists. Key maps are added, in the order specified, to either the top or the bottom of the key map list. Key map precedence in a key map list is used to resolve any conflicts between key definitions. The key definition in a preceding key map overrides any conflicting key definitions in key maps that follow in the key map list.

See the descriptions of the built-in procedures DEFINE_KEY, CREATE_KEY_MAP, and CREATE_KEY_MAP_LIST for more information on key definitions, key maps, and key map lists, respectively. Also see the description of the built-in procedure REMOVE_KEY_MAP for information on removing key maps from a key map list.

SIGNALED ERRORS

TPU$_NOKEYMAP	WARNING	Third argument is not a defined key map.
TPU$_KEYMAPNTFND	WARNING	The key map listed in the third argument is not found.
TPU$_TOOFEW	ERROR	Too few arguments passed to the ADD_KEY_MAP built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the ADD_KEY_MAP built-in.
TPU$_NOKEYMAPLIST	WARNING	Attempt to access an undefined key map list.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the ADD_KEY_MAP built-in.
TPU$_ILLREQUEST	WARNING	The position string must be either "*first*" or "*last*".
TPU$_BADREQUEST	WARNING	The position string must be either "*first*" or "*last*".

EXAMPLES

1
```
ADD_KEY_MAP ("TPU$KEY_MAP_LIST", "last", "TPU$KEY_MAP");
```

This statement adds the default key map TPU$KEY_MAP to the default key map list, TPU$KEY_MAP_LIST. Normally (except in the EVE editor) TPU$KEY_MAP is a member of the default key map list.

2
```
help_keys := CREATE_KEY_MAP ("help_keys");
ADD_KEY_MAP ("TPU$KEY_MAP_LIST", "first", help_keys);
```

These statements create a key map called HELP_KEYS and add it to the beginning of the default key map list, TPU$KEY_MAP_LIST. Key definitions in the new key map are invoked over definitions in the key maps already in the list.

ADJUST_WINDOW

Changes the size and/or screen location of a window and makes the window that you specify the current window.

FORMAT

ADJUST_WINDOW *(window, integer1, integer2)*

PARAMETERS

window

The window whose size or location you want to change. The window that you specify becomes the current window, and the buffer mapped to that window becomes the current buffer.

integer1

The signed integer value that you add to the screen line number at the top of the window.

integer2

The signed integer value that you add to the screen line number at the bottom of the window.

DESCRIPTION

If you want to check the visible size and/or location of a window before making an adjustment to it, use any of the following statements:

```
SHOW (WINDOW);

SHOW (WINDOWS);


top := GET_INFO (window, "top", VISIBLE_WINDOW);
MESSAGE (STR (top));

bottom := GET_INFO (window, "bottom", VISIBLE_WINDOW);
MESSAGE (STR (bottom));
```

There are screen line numbers at both the top and the bottom of the visible window. Adjust the size of a visible window by changing either or both of these screen line numbers. Make these changes by adding to or subtracting from the current screen line number, not by specifying the screen line number itself.

You can enlarge a window by decreasing the screen line number at the top of the window. (Specify a negative value for *integer1*.) You can also enlarge a window by increasing the screen line number at the bottom of the window. (Specify a positive value for *integer2*.) The following example adds four lines to the current window, provided that the values fall within the screen boundaries:

```
ADJUST_WINDOW (CURRENT_WINDOW, -2, +2)
```

If you specify integers that attempt to set the screen line number beyond the screen boundaries, VAXTPU issues a warning message. VAXTPU then sets the window boundary at the edge (top or bottom, as appropriate) of the screen.

You can reduce a window by increasing the screen line number at the top of the window. (Specify a positive value for *integer1*.) You can also reduce a window by decreasing the screen line number at the bottom of the window. (Specify a negative value for *integer2*.) If you attempt to make the size of the window smaller than one line (two lines if the window has a status line, three lines if the window has a status line and a horizontal scroll bar), VAXTPU issues an error message and no adjustment occurs. The following example reduces the current window by four lines:

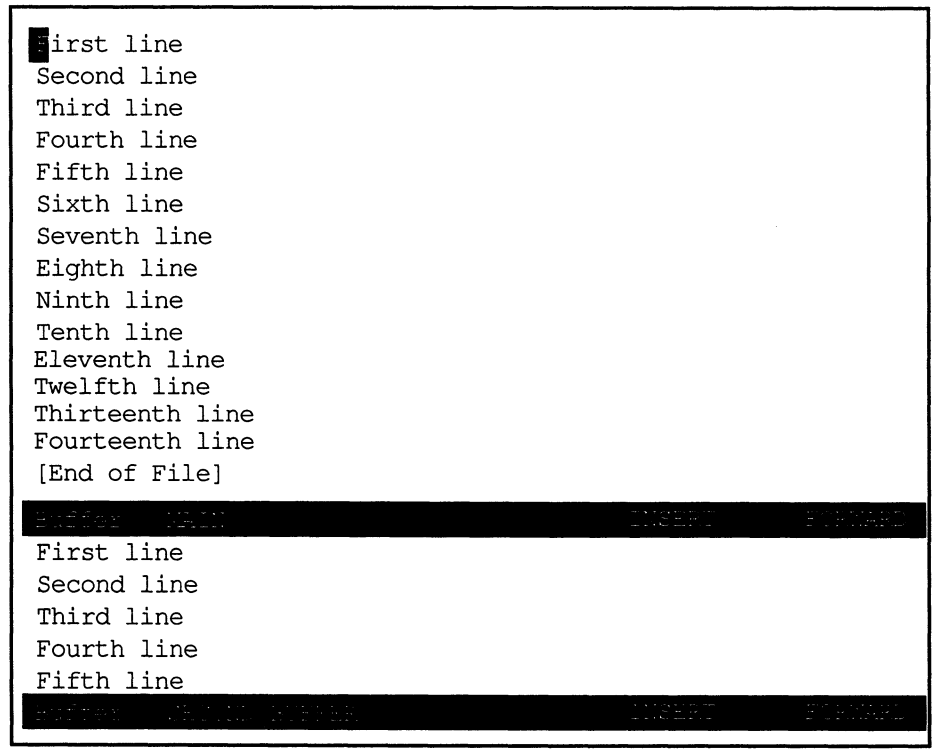
```
ADJUST_WINDOW (CURRENT_WINDOW, +2, -2)
```

You can also use ADJUST_WINDOW to change the position of the window on the screen without changing the size of the window. The following command simply moves the current window two lines higher on the screen, provided that the values fall within the screen boundaries:

```
ADJUST_WINDOW (CURRENT_WINDOW, -2, -2)
```

Figure 7–1 shows a screen layout when you invoke VAXTPU with EVE and a user-written command file. In this case, the user-written command file divides the screen into two windows. The top window has 15 text lines (including the end-of-buffer message) and a status line. The bottom window has five text lines and a status line. The two bottom lines of the screen are the command window and message window, each consisting of one line.

Figure 7–1 Screen Layout Before Using ADJUST_WINDOW

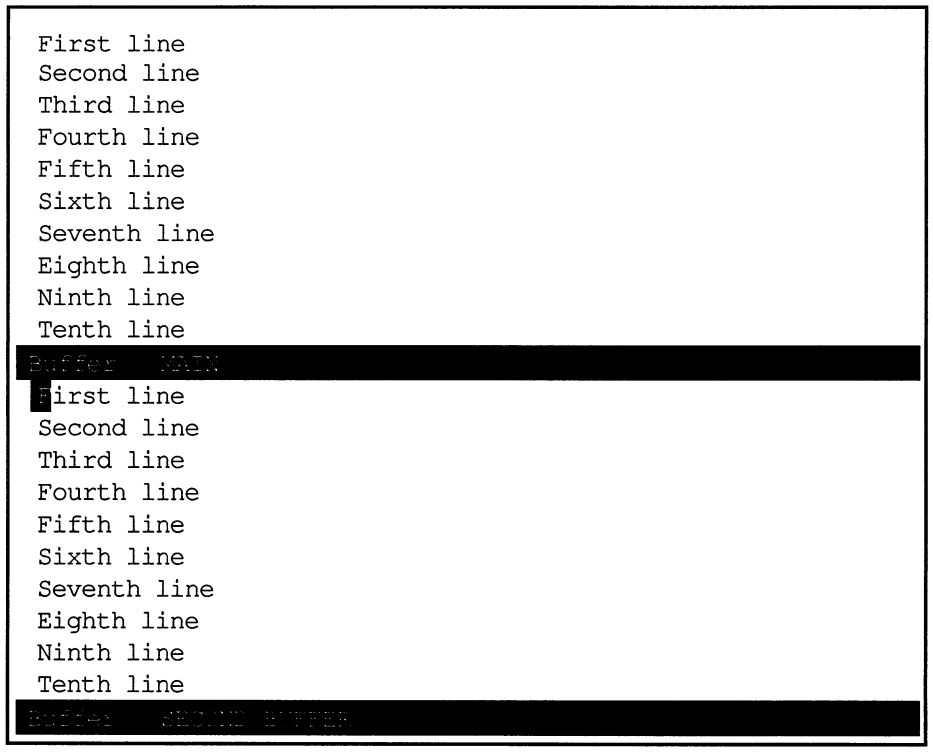


ZK–4047–GE

The user-written command file uses the variable *second_window* to identify the bottom window. Figure 7–2 shows the screen layout after the user enters ADJUST_WINDOW (second_window, –5, 0) after the appropriate prompt from EVE. Both the top and bottom windows now contain 10 lines of text and a status line. Note that the cursor is now located in the bottom window. The command and message windows still contain one line each.

ADJUST_WINDOW adds (+/–) *integer1* to the "visible_top" and (+/–) *integer2* to the "visible_bottom" of a window. The mapping of the window to its buffer is not changed. The new values for the screen line numbers become the values for the original top and original bottom. (See Chapter 2 for more information on window dimensions and window values.)

Figure 7–2 Screen Layout After Using ADJUST_WINDOW



ZK–4048–GE

Using ADJUST_WINDOW on a window makes it the current window; that is, VAXTPU puts the cursor in that window if the cursor was not already there, and VAXTPU marks that window as current in VAXTPU's internal tracking system. VAXTPU may scroll or adjust the text in the window to keep the current position visible after the adjustment occurs.

Note that both ADJUST_WINDOW and MAP may split or occlude other windows.

If you execute ADJUST_WINDOW within a procedure, the screen is not immediately updated to reflect the adjustment. The adjustment is made after the entire procedure is finished executing and control returns to the screen manager. If you want the screen to reflect the adjustment to the window before the entire procedure is executed, you can force the immediate update of a window by adding an UPDATE statement to the procedure. See the built-in procedure UPDATE for more information.

If you have defined a top or bottom scroll margin, and the window is adjusted so that the scroll margins no longer fit, TPU$_ADJSCROLLREG is signaled and the scroll margins shrink proportionally. For example, if you have a ten-line window, with an eight-line top scroll margin, shrinking the window to a five-line window also reduces the top scroll margin to four lines.

SIGNALED ERRORS

TPU$_ADJSCROLLREG	INFO	The window's scrolling region has been adjusted to fit the new window.
TPU$_BOTLINETRUNC	INFO	Bottom line cannot exceed bottom of screen.
TPU$_TOPLINETRUNC	INFO	Top line cannot exceed top of screen.
TPU$_WINDNOTMAPPED	WARNING	Cannot adjust a window that is not mapped.
TPU$_BADWINDADJUST	WARNING	Cannot adjust window to less than the minimum number of lines.
TPU$_WINDNOTVIS	WARNING	No adjustment if window is not visible.
TPU$_TOOFEW	ERROR	You specified less than three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLES

1

```
ADJUST_WINDOW (CURRENT_WINDOW, +5, 0)
```

This statement reduces the current window by removing five lines from the top of the window. If the top line of the window is screen line number 11, this statement changes the top line of the window to screen line number 16. (If the bottom line of the window is less than screen line number 16, VAXTPU signals an error.)

2

```
PROCEDURE user_display_help
   top_of_window := GET_INFO (CURRENT_WINDOW, "VISIBLE_TOP");
       !
       ! Remove the top five lines from the current window
       ! and replace them with a help window
       !
   ADJUST_WINDOW (CURRENT_WINDOW, +5, 0);
   example_window := CREATE_WINDOW (top_of_window, 5, ON);
   example_buffer := CREATE_BUFFER ("EXAMPLE",
                                    "sys$login:template.txt");
   MAP (example_window, example_buffer);
ENDPROCEDURE;
```

This procedure removes five lines from the top of a window and puts a help window in their place.

ANCHOR

Forces the next pattern element either to match immediately or else to fail.

FORMAT

ANCHOR

PARAMETERS

None.

DESCRIPTION

Normally, when SEARCH fails to find a match for a pattern, it retries the search. To try again, the SEARCH built-in moves the starting position one character forward or backward, depending upon the direction of the search. SEARCH continues this operation until it either finds a match for the pattern or reaches the end or beginning of the buffer or range being searched. If ANCHOR appears as the first element of a complex pattern, the search does not move the starting position. Instead, the search examines the next (or previous) character to determine if it matches the next character or element in the complex pattern. If the pattern does not match starting in the original position, the search fails. SEARCH does not move the starting position and retry the search.

When you build complex patterns using the + operator rather than the & operator, ANCHOR is useful only as the first element of a complex pattern. It is legal elsewhere in a pattern but has no effect.

Although ANCHOR behaves much like a built-in, it is actually a keyword.

For more information on patterns or modes of pattern searching, see Chapter 2.

SIGNALED ERRORS

ANCHOR is a keyword and has no completion codes.

EXAMPLES

```
1   pat1 := ANCHOR + "a123";
```

This assignment statement creates a pattern that matches the string *a123*. Because ANCHOR appears as the first element of the pattern, SEARCH will find *a123* only if the string appears at the starting position for the search.

```
2  PROCEDURE user_remove_comments
      LOCAL pat1,
            number_removed,
            end_mark;
      pat1 := ANCHOR + "!";
      number_removed := 0;
      end_mark := END_OF (CURRENT_BUFFER);

      POSITION (BEGINNING_OF (CURRENT_BUFFER));
      LOOP
         EXITIF MARK (NONE) = end_mark;
         r1 := SEARCH_QUIETLY (pat1, FORWARD);
         IF r1 <> 0
            THEN                        ! comment found so erase it
               ERASE_LINE;
               number_removed := number_removed + 1;
         ENDIF;
         MOVE_VERTICAL (1); ! move to the next line
      ENDLOOP;
      MESSAGE (FAO ("!ZL comment!%S removed.", number_removed));
   ENDPROCEDURE;
```

This procedure starts at the beginning of a buffer and searches forward, removing all comments that begin in column 1. The keyword ANCHOR in this example ties the search to the first character of a line (the current character). This prevents the search function from finding and removing exclamation points in the middle of a line (for example, in the FAO directive, !AS).

ANY

Returns a pattern that matches one or more characters from the set specified.

FORMAT

pattern := ANY ({ *buffer* / *range* / *string* } [, *integer1*])

PARAMETERS

buffer

An expression that evaluates to a buffer. ANY matches any of the characters in the resulting buffer.

range

An expression which evaluates to a range. ANY matches any of the characters in the resulting range.

string

An expression that evaluates to a string. ANY matches any of the characters in the resulting string.

integer1

This integer value indicates how many contiguous characters ANY matches. The default value for this integer is 1.

return value

A pattern matching one or more characters that appear in the string, buffer, or range passed as the first parameter to ANY.

DESCRIPTION

ANY is used to construct patterns.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	ANY must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	ANY requires at least one argument.
TPU$_TOOMANY	ERROR	ANY accepts no more than two arguments.
TPU$_ARGMISMATCH	ERROR	The argument you passed to the ANY built-in was of the wrong type.

TPU$_INVPARAM	ERROR	The argument you passed to the ANY built-in was of the wrong type.
TPU$_MINVALUE	WARNING	The argument you passed to the ANY built-in was less than the minimum accepted value.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of the ANY built-in.

EXAMPLES

1
```
pat1 := ANY ("hijkl")
```

This assignment statement creates a pattern that matches any one of the characters *h*, *i*, *j*, *k*, and *l*.

2
```
pat1 := any ("xy", 2);
```

This assignment statement creates a pattern that matches any of the following two-letter combinations: *xx*, *xy*, *yx*, and *yy*.

3
```
a_buf := CREATE_BUFFER ("new buffer");
POSITION (a_buf);
COPY_TEXT ("xy");
SPLIT_LINE;
COPY_TEXT ("abc");
pat1 := ANY (a_buf);
```

These statements create a pattern that matches any one of the characters *a*, *b*, *c*, *x*, and *y*.

4
```
PROCEDURE user_find_endprocedure
   LOCAL  endprocedure_pattern,
       search_range;
   endprocedure_pattern := (LINE_BEGIN + "ENDPROCEDURE") +
                           (LINE_END | ANY (";! " + ASCII (9)));
   search_range := SEARCH_QUIETLY (endprocedure_pattern, FORWARD);
   IF search_range = 0
      THEN
         MESSAGE ("Endprocedure statement not found");
      ELSE
         POSITION (END_OF (search_range));
   ENDIF;
ENDPROCEDURE;
```

This procedure finds an ENDPROCEDURE statement that starts in column 1 and moves the editing point to the end of the statement.

APPEND_LINE

Places the current line at the end of the previous line.

FORMAT

APPEND_LINE

PARAMETERS

None.

DESCRIPTION

You can use APPEND_LINE to delete line terminators.

The editing point in the line that was the current line before APPEND_LINE was executed becomes the editing point.

Using APPEND_LINE may cause VAXTPU to insert padding spaces or blank lines in the buffer. APPEND_LINE causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

SIGNALED ERRORS

TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.
TPU$_NOCACHE	ERROR	There is not enough memory to allocate a new cache.
TPU$_TOOMANY	ERROR	APPEND_LINE does not accept arguments.
TPU$_NOTMODIFIABLE	WARNING	You cannot modify an unmodifiable buffer.
TPU$_LINETOOLONG	WARNING	VAXTPU cannot append the line because the length of the resulting line would exceed VAXTPU's maximum line length.

EXAMPLES

```
1  APPEND_LINE
```

This statement adds the current line to the end of the previous line.

```
2  ! The following procedure deletes the character
   ! to the left of the cursor.  If the cursor is at the
   ! beginning of a line, it appends the current line
   ! to the end of the previous line.
   !
   PROCEDURE user_delete_char
      IF CURRENT_OFFSET = 0
         THEN
            APPEND_LINE;
         ELSE
            ERASE_CHARACTER (-1);
      ENDIF;
   ENDPROCEDURE;
```

This procedure deletes the character to the left of the cursor. If you are at the beginning of a line, the procedure appends the current line to the end of the previous line. The procedure works correctly even if the window is shifted.

You can bind this procedure to the DELETE key with the following statement:

```
DEFINE_KEY ("user_delete_char", DEL_KEY);
```

ARB

Returns a pattern that matches an arbitrary sequence of characters starting at the editing point and extending for the length you specify.

FORMAT

pattern := ARB *(integer)*

PARAMETER

integer
The number of characters in the pattern. This integer must be positive.

return value

A pattern that matches an arbitrary sequence of characters starting at the editing point and extending for the length you specify.

DESCRIPTION

ARB can be used for wildcard matches of fixed length.

For more information on patterns, see Chapter 2.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	ARB must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	ARB requires at least one argument.
TPU$_TOOMANY	ERROR	ARB accepts no more than one argument.
TPU$_INVPARAM	ERROR	The argument to ARB must be an integer.
TPU$_MINVALUE	WARNING	The argument to ARB must be positive.

EXAMPLES

1
```
pat1 := ARB (5)
```

This assignment statement creates a pattern that matches the next five characters starting at the editing point. The characters themselves are arbitrary; it is the number of characters that is important for a pattern created with ARB.

2
```
pat2 := "J" & ARB (2)
```

This assignment statement creates a pattern that matches a string beginning with a J and followed by any two other characters. Names such as "Jim," "Jan," and "Joe" match *pat2*.

```
3  PROCEDURE user_replace_prefix
      LOCAL  cur_mode,
             here,
             pat1,
            found_range;

      pat1 := (LINE_BEGIN | NOTANY ("ABCDEFGHIJKLMNOPQRSTUVWXYZ_$"))
              + ((ARB (3) + "_") @ found_range);
      here := MARK (NONE);
      cur_mode := GET_INFO (current_buffer, "mode");

      POSITION (BEGINNING_OF (CURRENT_BUFFER));
      LOOP
         found_range := 0;
         SEARCH_QUIETLY (pat1, FORWARD);
         EXITIF found_range = 0;
         ERASE (found_range);
         POSITION (END_OF (found_range));
         COPY_TEXT ("user_");
     ENDLOOP;
     POSITION (here);
     SET (cur_mode, current_buffer);
   ENDPROCEDURE;
```

This procedure replaces a prefix of any three characters followed by an underscore (xxx_) in the current buffer with the string "user_". It does not change the current position.

ASCII

Returns the ASCII value of a character or the character that has the specified ASCII value.

FORMAT

{ **integer2** / **string2** } **:= ASCII** ({ *integer1* / *keyword* / *string1* })

PARAMETERS

integer1

The decimal value of a character in the DEC Multinational Character Set.

keyword

This keyword must be a key name. If the key name is the name of a key that produces a printing character, ASCII returns that character. Otherwise it returns the character whose ASCII value is 0.

string1

The character whose ASCII value you want. If the string has a length greater than 1, the ASCII built-in returns the ASCII value of the first character in the string.

return value

The character with the specified ASCII value (if you specify an integer or keyword parameter).

The ASCII value of the string you specify (if you specify a string parameter).

DESCRIPTION

The result of this built-in depends upon its argument. If the argument is an integer then it returns a string of length 1 that represents the character of the DEC Multinational Character Set corresponding to the integer you specify. If the argument is a string then it takes the first character of the string and returns the integer corresponding to the ASCII value of that character.

If the argument to ASCII is a keyword, that keyword must be a key name. The VAXTPU built-in KEY_NAME produces key names. In addition, there are several predefined keywords that are key names. See Table 2–1 for a list of these keywords. If the keyword is a key name and the key produces a printing character, ASCII returns that character; otherwise, it returns the character whose ASCII value is 0.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	ASCII must be on the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	ASCII requires one argument.
TPU$_TOOMANY	ERROR	ASCII accepts only one argument.
TPU$_ARGMISMATCH	ERROR	The parameter you passed to ASCII is of the wrong type.
TPU$_NULLSTRING	WARNING	You passed a string of length 0 to ASCII.

EXAMPLES

1
```
my_character := ASCII(12)
```

This assignment statement assigns a string of length 1 to the variable *my_character*. This string contains the form feed character because that character has the ASCII value 12.

2
```
MESSAGE (ASCII (80))
```

This statement combines two built-in procedures and prints the ASCII character numbered 80 (whose value is P) in the message area. In this case, uppercase P is displayed.

3
```
! This procedure puts a tab character in your text
!
PROCEDURE user_tab
   COPY_TEXT (ASCII (9));
ENDPROCEDURE;
```

This procedure includes a tab character in the current buffer.

4
```
ascii_value := ASCII ("a");
```

This assignment statement assigns the integer value 97 to the variable *ascii_value*. Note that *a* is specified in quotation marks because it is a parameter of type string. For more information on specifying strings, see Chapter 2.

5
```
PROCEDURE user_test_key
   LOCAL key_struck,
         key_value;

   MESSAGE ("Press a key");
   key_struck := READ_KEY;
   key_value := ASCII (key_struck);

   IF key_value = ASCII (0)
   THEN
      MESSAGE ("That is not a typing key");
   ELSE
      MESSAGE (FAO ("That key produces the letter "!AS".", key_value));
   ENDIF;
ENDPROCEDURE;
```

This procedure prompts the user to press a key. When the user does so, the procedure reads the key. If the key is associated with a printing character, ASCII tells the user what character is produced. If the key is not associated with a printable character, ASCII informs the user of this.

ATTACH

Enables you to switch control from your current process to another process that you have previously created.

FORMAT

ATTACH *[({ integer / string })]*

PARAMETERS

integer

This integer is the process identification (PID) of the process to which terminal control is to be switched. You must use decimal numbers to specify the PID to VAXTPU.

string

A quoted string, a variable name representing a string constant, or an expression that evaluates to a string that VAXTPU interprets as a process name.

DESCRIPTION

To use the built-in procedure ATTACH, you must have previously created a subprocess. If the process you specify is not part of the current job or does not exist, an error message is displayed. For information on creating subprocesses, see the description of SPAWN in this section.

ATTACH suspends the current VAXTPU process and switches context to the process you use as a parameter. If you do not specify a parameter for ATTACH, VAXTPU switches control to the parent or owner process. A subsequent use of the DCL command ATTACH (or a logout from any process except the parent process) resumes the execution of the suspended VAXTPU process.

In all cases, VAXTPU first deassigns the terminal. If a VAXTPU process is resumed following a SPAWN or ATTACH command, VAXTPU reassigns the terminal and refreshes the screen.

If the current buffer is mapped to a visible window, the ATTACH built-in causes the screen manager to synchronize the editing point, which is a buffer location, with the cursor position, which is a window location. This may result in the insertion of padding spaces or lines into the buffer if the cursor position is before the beginning of a line, in the middle of a tab, beyond the end of a line, or after the last line in the file.

ATTACH is not a valid built-in in DECwindows VAXTPU. However, if you are running non-DECwindows VAXTPU in a DECwindows terminal emulator, ATTACH works as described in this section.

SIGNALED ERRORS

TPU$_NOPARENT	WARNING	There is no parent process to which you can attach — your current process is the top-level process.
TPU$_TOOMANY	ERROR	Too many arguments passed to the ATTACH built-in.
TPU$_SYSERROR	ERROR	Error requesting information about the process being attached to.
TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to the ATTACH built-in. Only process name strings and process IDs are allowed.
TPU$_CREATEFAIL	WARNING	Unable to attach to the process.
TPU$_REQUIRESTERM	ERROR	Feature requires a terminal.

EXAMPLES

1
```
ATTACH
```

This statement causes VAXTPU to attach to the parent process.

2
```
ATTACH (97899)
```

This statement causes VAXTPU to attach to the subprocess with the PID 97899.

3
```
ATTACH ("JONES_2")
```

This statement switches the terminal's control to the process JONES_2.

BEGINNING_OF

Returns a marker that points to the first position of a buffer or a range.

FORMAT

marker := BEGINNING_OF ({ *buffer* / *range* })

PARAMETERS

buffer

The buffer whose beginning you want to mark.

range

The range whose beginning you want to mark.

return value

A marker pointing to the first character position of the specified buffer or range.

DESCRIPTION

If you use the marker returned by this built-in procedure as a parameter for the built-in procedure POSITION, the editing point moves to the marker.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	BEGINNING_OF must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	BEGINNING_OF requires one argument.
TPU$_TOOMANY	ERROR	BEGINNING_OF accepts only one argument.
TPU$_ARGMISMATCH	ERROR	You passed something other than a range or a buffer to BEGINNING_OF.

EXAMPLES

1
```
beg_main := BEGINNING_OF (main_buffer)
```

This assignment statement stores the marker that points to the beginning of the main buffer in the variable *beg_main*.

2
```
POSITION (BEGINNING_OF (my_range))
```

This statement uses two built-in procedures to move your current character position to the beginning of *my_range*. If *my_range* is in a visible buffer in which the cursor is located, the cursor position is also moved to the beginning of *my_range*.

3
```
PROCEDURE user_top
   IF MARK (NONE) = BEGINNING_OF (CURRENT_BUFFER)
   THEN
      MESSAGE ("Already at top");
   ELSE
      POSITION (BEGINNING_OF (CURRENT_BUFFER));
   ENDIF;
ENDPROCEDURE;
```

This procedure places the cursor at the beginning of the current buffer. If you are already at the beginning of the buffer, the message "Already at top" is displayed in the message area.

4
```
PROCEDURE user_include_file
! Create scratch buffer
   b1 := CREATE_BUFFER ("Scratch Buffer");

! Map scratch buffer to main window
   MAP (main_window, b1);

! Read in file name given
   READ_FILE (READ_LINE ("File to Include:" ));

! Go to top of file
   POSITION (BEGINNING_OF (b1));
ENDPROCEDURE;
```

This procedure creates a new buffer, associates the buffer with the main window, and maps the main window to the screen. It positions to the top of the buffer, prompts the user for the name of a file to include, and reads the file into the buffer.

BREAK

Activates the debugger if VAXTPU was invoked with the /DEBUG qualifier.

FORMAT

BREAK

PARAMETERS

None.

DESCRIPTION

If VAXTPU was invoked with the /DEBUG qualifier, then execution of the BREAK statement activates the debugger. If there is no debugger, BREAK causes the following message to be displayed in the message window:

```
Breakpoint at line xxx
```

It has no other effect. Although BREAK behaves much like a built-in, it is actually a VAXTPU language element.

BREAK is evaluated for correct syntax at compile time. In contrast, VAXTPU procedures are usually evaluated for a correct parameter count and parameter types at execution time.

SIGNALED ERROR

BREAK is a language element and has no completion codes.

EXAMPLE

```
PROCEDURE user_not_quite_working
      .
      .
      .
   BREAK;
      .
      .
      .
ENDPROCEDURE;
```

This procedure contains a break statement. If the statement is executed, VAXTPU'S debugger is activated, allowing the user to debug that section of the code.

CALL_USER

Calls a program written in another language from within VAXTPU. The CALL_USER parameters are passed to the external routine exactly as you enter them; VAXTPU does not process the parameters in any way. The integer is passed by reference, and *string1* is passed by descriptor. *String2* is the value returned by the external program.

FORMAT

string2 := CALL_USER *(integer, string1)*

PARAMETERS

integer

The integer that is passed to the user-written program by reference.

string1

The string that is passed to the user-written program by descriptor.

return value

The value returned by the called program.

DESCRIPTION

In addition to returning the value *string2* to CALL_USER, the external program returns a status code that tells whether the program executed successfully. You can trap this status code in an ON_ERROR statement. An even-numbered status code (low bit in R0 clear) causes the ON_ERROR statement to be executed. The ERROR lexical element returns the status value from the program in the form of a keyword.

To use the built-in procedure CALL_USER, follow these steps:

- Write a program in whatever language you choose. The program must be a global routine called TPU$CALLUSER.
- Compile the program.
- Link the program with an options file to create a shareable image.
- Define the logical name TPU$CALLUSER to point to the file containing your routine.
- Invoke VAXTPU.
- From within a VAXTPU session, call your external program to perform its function by specifying the built-in procedure CALL_USER with the appropriate parameters. If you link your program properly, and if you define the logical name TPU$CALLUSER to point to your program, the built-in procedure CALL_USER passes the parameters you give it to the proper routine.

The CALL_USER parameters are input parameters for the external program you are calling. VAXTPU does not process the parameters in any way but passes them to the external procedure exactly as you enter them. You must supply both parameters even if the routine you are calling does not require that information be passed to it. Enter the following null parameters to indicate that you are not passing any actual values:

```
CALL_USER (0,"")
```

For information on the VAXTPU callable interface, see the *VMS Utility Routines Manual*.

SIGNALED ERRORS

TPU$_BADUSERDESC	ERROR	User-written routine incorrectly filled in the return descriptor.
TPU$_NOCALLUSER	ERROR	Could not find a routine to invoke.
TPU$_TOOFEW	ERROR	Too few arguments passed to CALL_USER.
TPU$_TOOMANY	ERROR	Too many arguments passed to CALL_USER.
TPU$_NEEDTOASSIGN	ERROR	The call to CALL_USER must be on the right-hand side of the assignment statement.
TPU$_INVPARAM	ERROR	Wrong type of data sent to CALL_USER.
TPU$_ARGMISMATCH	ERROR	Parameter is of the wrong data type.
TPU$_CALLUSERFAIL	WARNING	CALL_USER routine failed with status %X'status'. The value returned by ERROR after this type of error will be the status value reported by this message.

EXAMPLES

1
```
ret_value := CALL_USER (6, "ABC")
```

This statement calls a program that the user wrote. Before invoking VAXTPU,the user created a logical name, TPU$CALLUSER, that points to the file containing the program the user wants called by CALL_USER. VAXTPU passes the first parameter (6) by reference, and the second parameter ("ABC") by descriptor. If, for example, the user program uses an integer and a string as input values, the program processes the integer "6" and the string "ABC." If the program is designed to return a result, the result is returned in the variable *ret_value*.

2 Step-by-Step Example of Using CALL_USER

The following example shows the steps required to use the built-in procedure CALL_USER. The routine that is called to do floating-point arithmetic is written in BASIC.

1 Write a program in BASIC that does floating-point arithmetic on the values passed to it.

```
! Filename:FLOATARITH.BAS
1       sub TPU$CALLUSER ( some_integer% , input_string$ , return_string$ )
10      ! don't check some_integer% because this function only does
        ! floating-point arithmetic
20      ! parse the input string
        ! find and extract the operation
        comma_location = pos ( input_string$, ",", 1% )
        if comma_location = 0 then go to all_done
        end if
        operation$ = seg$( input_string$, 1%, comma_location - 1% )
        ! find and extract the 1st operand
        operand1_location = pos ( input_string$, ",", comma_location +1 )
        if operand1_location = 0 then go to all_done
        end if
        operand1$ = seg$( input_string$, comma_location + 1% ,  &
                          operand1_location -1 )
        ! find and extract the 2nd operand
        operand2_location = pos ( input_string$, ",", operand1_location +1 )
        if operand2_location = 0 then
                operand2_location = len( input_string$) + 1
        end if
        operand2$ = seg$( input_string$, operand1_location + 1% ,  &
                          operand2_location -1 )
        select operation$ ! do the operation
        case "+"
                result$ = sum$(  operand1$ , operand2$ )  !
        case "-"
                result$ =  dif$(  operand1$, operand2$ )  !
        case "*"
                result$ =  num1$( Val( operand1$ ) * Val( operand2$ ) )
        case "/"
                result$ =  num1$( Val( operand1$ ) / Val( operand2$ ) )
        case else
                result$ =  "unknown operation."
        end select
        return_string$ = result$
999     all_done: end sub
```

2 Compile the program with the following statement:

```
$ BASIC/LIST floatarith
```

3 Create an options file to be used by the linker when you link the BASIC program.

```
!+
!   File: FLOATARITH.OPT
!
!   Options file to link floatarith BASIC program with VAXTPU
!
!-
floatarith.obj
UNIVERSAL=TPU$CALLUSER
```

4 Link the program (using the options file) to create a shareable image.

```
$  LINK floatarith/SHARE/OPT/MAP/FULL
```

5 Define the logical name TPU$CALLUSER to point to the executable image of the BASIC program.

```
$  DEFINE TPU$CALLUSER device:[directory]floatarith.EXE
```

6 Invoke VAXTPU.

7 Write and compile the following VAXTPU procedure:

```
PROCEDURE my_call_user

! test the built-in procedure call_user

   LOCAL output,
         input;

   input := READ_LINE ("Call user >");  ! Provide a parameter for routine
   output := CALL_USER ( 0, input);     ! Value this routine returns
   MESSAGE (output);
ENDPROCEDURE;
```

8 When you call the procedure *my_call_user*, you are prompted for parameters to pass to the BASIC routine. The order of the parameters is operator, number, number. For example, if you enter "+, 3.33, 4.44" after the prompt, the result 7.77 is displayed in the message area.

CHANGE_CASE

Changes the case of all alphabetic characters in a buffer, range, or string, according to the keyword that you specify. Optionally, CHANGE_CASE returns a string, range, or buffer containing the changed text.

FORMAT

```
{ returned_buffer }
{ returned_range  } := CHANGE_CASE ( { buffer } , { INVERT }
{ returned_string }                  { range  }   { LOWER  }
                                     { string }   { UPPER  }
                                     [[ , IN_PLACE     ]] )
                                     [[ , NOT_IN_PLACE ]]
```

PARAMETERS

buffer
The buffer in which you want VAXTPU to change the case. Note that you cannot use the keyword NOT_IN_PLACE if you specify a buffer for the first parameter.

range
The range in which you want VAXTPU to change the case. Note that you cannot use the keyword NOT_IN_PLACE if you specify a range for the first parameter.

string
The string in which you want VAXTPU to change the case. If you specify IN_PLACE for the third parameter, CHANGE_CASE makes the specified change to the string specified in the first parameter. Note that if *string* is a constant, IN_PLACE has no effect.

LOWER
A keyword directing VAXTPU to change letters to all lowercase.

UPPER
A keyword directing VAXTPU to change letters to all uppercase.

INVERT
A keyword directing VAXTPU to change uppercase letters to lowercase and lowercase letters to uppercase.

IN_PLACE
A keyword directing VAXTPU to make the indicated change in the buffer, range, or string specified. This is the default.

NOT_IN_PLACE
A keyword directing VAXTPU to leave the specified string unchanged and return a string that is the result of the specified change in case. You cannot use NOT_IN_PLACE if the first parameter is specified as a range or buffer. To use NOT_IN_PLACE, you must specify a return value for CHANGE_CASE.

return values

returned_buffer

A variable of type buffer pointing to the buffer containing the modified text, if you specify a buffer for the first parameter. The variable *returned_buffer* points to the same buffer pointed to by the buffer variable specified as the first parameter.

returned_range

A range containing the modified text, if you specify a range for the first parameter. The returned range spans the same text as the range specified as a parameter, but they are two separate ranges. If you subsequently change or delete one of the ranges, this has no effect on the other range.

returned_string

A string containing the modified text, if you specify a string for the first parameter. CHANGE_CASE can return a string even if you specify IN_PLACE.

DESCRIPTION

CHANGE_CASE modifies the case of all the alphabetic characters in the specified unit of text according to the keyword that you supply.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	CHANGE_CASE requires two parameters.
TPU$_TOOMANY	ERROR	CHANGE_CASE accepts only two parameters.
TPU$_ARGMISMATCH	ERROR	One of the parameters to CHANGE_CASE is of the wrong data type.
TPU$_INVPARAM	ERROR	One of the parameters to CHANGE_CASE is of the wrong data type.
TPU$_BADKEY	WARNING	You gave the wrong keyword to CHANGE_CASE.
TPU$_NOTMODIFIABLE	WARNING	You cannot change the case of text in an unmodifiable buffer.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of CHANGE_CASE.

EXAMPLES

1
```
CHANGE_CASE (CURRENT_BUFFER, UPPER)
```

This statement makes all the characters in the current buffer uppercase. If you enter this statement on the command line of your interface, you see the effects immediately. If you use this statement within a procedure, you see the effect of the statement at the next screen update.

2
```
CHANGE_CASE (my_range, LOWER)
```

This statement makes all the characters in *my_range* lowercase. If *my_range* is part of a buffer that is mapped to a window, you see the command take effect immediately.

3
```
PROCEDURE user_lowercase_line
   LOCAL this_line;

   this_line := ERASE_LINE;
   CHANGE_CASE (this_line, LOWER);
   SPLIT_LINE;
   MOVE_VERTICAL (-1);
   COPY_TEXT (this_line);
ENDPROCEDURE;
```

This procedure changes the current line to lowercase.

4
```
PROCEDURE user_upcase_item
   ON_ERROR
   ! In case no string is found during search
      MESSAGE ("No current item.");
      RETURN;
   ENDON_ERROR;

   delimiters   := " " + ASCII(9);
   current_item := ANCHOR & SCAN (delimiters);
   item_range   := SEARCH (current_item, FORWARD, NO_EXACT);
   CHANGE_CASE (item_range, UPPER);
ENDPROCEDURE;
```

This procedure puts the current text object in uppercase.

5
```
returned_value := CHANGE_CASE (CURRENT_BUFFER, LOWER, IN_PLACE);
```

This statement makes all characters in the current buffer lowercase. The variable *returned_value* contains the newly modified current buffer.

6
```
returned_value := CHANGE_CASE (the_string, INVERT, NOT_IN_PLACE);
```

This statement inverts the case of all characters in the string pointed to by *the_string* and returns the modified string in the variable *returned_value*. It does not change *the_string* in any way.

COMPILE

Converts VAXTPU procedures and statements into an internal, compiled format. Valid items for compilation can be represented by a string, a range, or a buffer. COMPILE optionally returns a program.

FORMAT

[**program :=**] **COMPILE** ({ *buffer* / *range* / *string* })

PARAMETERS

buffer

A buffer that contains only valid VAXTPU declarations and statements.

range

A range that contains only valid VAXTPU declarations and statements.

string

A string that contains only valid VAXTPU declarations and statements.

return value

The program created by compiling the declarations and statements in the string, range, or buffer. If the program failed to compile, an integer zero is returned.

DESCRIPTION

The program that COMPILE optionally returns is the compiled form of valid VAXTPU procedures, statements, or both. You can assign the compiled version of VAXTPU code to a variable name. VAXTPU statements, as well as procedure definitions, can be stored in the program returned by COMPILE. Later in your editing session, you can execute the VAXTPU code that you compiled by using the program as a parameter for the built-in procedure EXECUTE. You can also use the program as a parameter for the built-in procedure DEFINE_KEY to define a key to execute the program. Then you can execute the program by pressing that key.

COMPILE returns a program variable only if the compilation generates executable statements. COMPILE does not return a program variable if you compile any of the following:

- Null strings or buffers
- Procedure definitions that do not have any executable statements following them
- Programs with syntax errors

VAXTPU cannot compile a string, range, or line of text in a buffer longer than 256 characters. If VAXTPU encounters a longer string, range, or line, VAXTPU truncates characters after the 256th character and attempts to compile the truncated string, buffer, or range.

If necessary, use the built-in procedure SET (INFORMATIONAL, ON) before compiling a procedure interactively to see the compiler messages.

To check the results of a compilation to determine whether execution is possible, use the following statement in a program:

```
x := COMPILE (my_range);
!if the program is nonzero, continue
IF x <> 0
THEN
  .
  .
  .
ENDIF;
```

If x = 0, no program was generated because of compilation errors or because there were no executable statements. The statement "IF x <> 0 THEN" allows your program to continue as long as a program was generated.

You can also use an ON_ERROR statement to check the result of a compilation. This statement tells you whether the compilation completed successfully; it does not tell you whether execution is possible. You can use an ON_ERROR statement when compiling code consisting of procedure definitions without following executable statements. For more information on using ON_ERROR statements, see Section 3.8.4.7.

SIGNALED ERRORS

TPU$_COMPILEFAIL	ERROR	Compilation aborted because of syntax errors.
TPU$_ARGMISMATCH	ERROR	The data type of a parameter passed to the COMPILE built-in is unsupported.
TPU$_TOOFEW	ERROR	Too few arguments.
TPU$_TOOMANY	ERROR	Too many arguments.

EXAMPLES

1
```
dwn := COMPILE ("MOVE_VERTICAL (1)")
```

This assignment statement associates the MOVE_VERTICAL (1) function with the variable *dwn*. You can use the variable *dwn* with the built-in procedure EXECUTE to move the editing point down one line.

2 `user_program := COMPILE (main_buffer)`

This assignment statement compiles the contents of the main buffer. If the buffer contains executable statements, VAXTPU returns a program that stores these executable commands. If the buffer contains procedure definitions, VAXTPU compiles the procedures and lists them in the procedure definition table so that you can call them in one of the following ways:

- Enter the name of the procedure after the appropriate prompt from the interface you are using.
- Call the procedure from within other procedures.

CONVERT

Given the coordinates of a point in one coordinate system, returns the corresponding coordinates for the point in the coordinate system you specify.

FORMAT

CONVERT ({ *DECW_ROOT_WINDOW* / *SCREEN* / *window* }, { *CHARACTERS,* / *COORDINATES,* }
from_x_integer, from_y_integer,
{ *DECW_ROOT_WINDOW* / *SCREEN* / *window* }, { *CHARACTERS,* / *COORDINATES,* }
to_x_integer, to_y_integer)

PARAMETERS

DECW_ROOT_WINDOW
Specifies the coordinate system to be that used by the root window of the screen on which VAXTPU is running.

SCREEN
Specifies the coordinate system to be that used by the DECwindows window associated with VAXTPU's top-level widget.

window
Specifies the coordinate system to be that used by the VAXTPU window.

CHARACTERS
Specifies a system that measures screen distances in rows and columns, as a character-cell terminal does. In a character-cell-based system, the cell in the top row and the leftmost column has the coordinates (1,1).

COORDINATES
Specifies a DECwindows coordinate system in which coordinate units correspond to pixels. The pixel in the upper left corner has the coordinates (0, 0).

from_x_integer
from_y_integer
Integer values representing a point in the original coordinate system and units.

to_x_integer
to_y_integer
Variables of type integer representing a point in the specified coordinate system and units. Note that the previous contents of the parameters are deleted when VAXTPU places the resulting values in them. You must specify VAXTPU variables for the parameters *to_x_integer* and *to_y_integer*. Passing a constant integer, string or keyword value causes an

error. (This requirement does not apply to the parameters *from_x_integer* and *from_y_integer*.)

DESCRIPTION

The converted coordinates are returned using the *to_x_integer* and *to_y_integer* parameters. Note that coordinate systems are distinguished both by units employed and where each places its origin.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by CONVERT.
TPU$_BADDELETE	ERROR	You are attempting to modify an integer, keyword, or string constant.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to CONVERT.
TPU$_TOOMANY	ERROR	Too many arguments passed to CONVERT.
TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_WINDNOTVIS	WARNING	CONVERT cannot operate on an invisible window.

EXAMPLE

```
PROCEDURE user_convert

LOCAL source_x,
      source_y,
      dest_x,
      dest_y;

source_x := 1;
source_y := 1;
dest_x := 0;
dest_y := 0;


CONVERT (CURRENT_WINDOW, COORDINATES, source_x, source_y,
         SCREEN, COORDINATES, dest_x, dest_y);


ENDPROCEDURE;
```

This example converts a point's location from the current window's coordinate system (with the origin in the upper left-hand corner of the window) to the VAXTPU screen's coordinate system (with the origin in the upper left-hand corner of the VAXTPU screen). For more information about the difference between a VAXTPU window and the VAXTPU screen,

see Chapter 4. If the current window is not the top window, CONVERT changes the value of the *y*-coordinate to reflect the difference in the VAXTPU screen's coordinate system. For another example of a procedure using the CONVERT built-in, see Example B–1.

COPY_TEXT

Makes a copy of the text you specify and places it in the current buffer.

FORMAT

[range2 :=] COPY_TEXT ({ *buffer* | *range1* | *string* })

PARAMETERS

buffer

The buffer containing the text you want to copy.

range1

The range containing the text you want to copy.

string

A string, a variable name representing a string constant, or an expression that evaluates to a string, representing the text you want to copy.

return value

The range where the copied text has been placed.

DESCRIPTION

If the current buffer is in insert mode, the text you specify is inserted before the current position in the current buffer. If the current buffer is in overstrike mode, the text you specify replaces text starting at the current position and continuing for the length of the string, range, or buffer.

Note: You cannot add a buffer or a range to itself. If you try to add a buffer to itself, VAXTPU issues an error message. If you try to insert a range into itself, part of the range is copied before VAXTPU signals an error. If you try to overstrike a range into itself, VAXTPU may or may not signal an error.

Using COPY_TEXT may cause VAXTPU to insert padding spaces or blank lines in the buffer. COPY_TEXT causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

SIGNALED ERRORS

TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.
TPU$_NOCOPYBUF	WARNING	Trying to copy a buffer to itself is not allowed.
TPU$_NOCACHE	ERROR	There is not enough memory to allocate a new cache.
TPU$_OVERLAPRANGE	ERROR	You tried to put the contents of a range into that same range instead of into another structure.
TPU$_TOOFEW	ERROR	COPY_TEXT requires one argument.
TPU$_TOOMANY	ERROR	COPY_TEXT accepts only one argument.
TPU$_ARGMISMATCH	ERROR	The argument to COPY_TEXT must be a string, range, or buffer.
TPU$_NOTMODIFIABLE	ERROR	You cannot copy text into an unmodifiable buffer.
TPU$_LINETOOLONG	WARNING	The line exceeds VAXTPU's maximum line length.
TPU$_TRUNCATE	WARNING	Characters have been truncated because you tried to add text that would exceed the maximum line length.

EXAMPLES

1
```
COPY_TEXT ("Perseus is near Andromeda")
```

When the buffer is set to insert mode, this statement causes the string "Perseus is near Andromeda" to be placed just before the current position in the current buffer.

2
```
COPY_TEXT (ASCII (10))
```

When the buffer is set to overstrike mode, this statement causes the ASCII character for line feed to replace the current character in the current buffer.

3
```
PROCEDURE user_simple_insert
   IF BEGINNING_OF (paste_buffer) = END_OF (paste_buffer)
   THEN
      MESSAGE ("Nothing to INSERT");
   ELSE
       COPY_TEXT (paste_buffer);
   ENDIF;
ENDPROCEDURE;
```

This procedure implements a simple INSERT HERE function. It assumes that there is a paste buffer and that this buffer contains the most recently deleted text. The procedure copies the text from that buffer into the current buffer.

CREATE_ARRAY

Creates an array.

FORMAT

[array :=]
CREATE_ARRAY *[(integer1 [, integer2])]*

PARAMETERS

integer1

The number of integer-indexed elements to be created when the array is created. VAXTPU processes elements specified by this parameter more quickly than elements created dynamically. You can add integer-indexed elements dynamically, but they are not processed as quickly as predeclared, integer-indexed elements.

integer2

The first predeclared integer index of the array. The predeclared integer indexes of the array extend from this integer through to *integer2* + *integer1* – 1. This parameter defaults to 1.

return value

The variable that is to contain the newly created array.

DESCRIPTION

This built-in creates an array.

In VAXTPU, an array is a one-dimensional collection of data values that can be considered or manipulated as a unit.

To create an array variable called *bat*, use the CREATE_ARRAY built-in as follows:

```
bat := CREATE_ARRAY;
```

VAXTPU arrays can have a static portion, a dynamic portion, or both. A static array or portion of an array contains predeclared, integer-indexed elements. These elements are allocated contiguous memory locations to support quick processing. To create an array with a static portion, specify the number of contiguous, integer-indexed elements when you create the array. You also have the option of specifying a beginning index number other than 1. For example, the following statement creates an array with 100 predeclared integer-indexed elements starting at 15:

```
bat := CREATE_ARRAY (100, 15);
```

All static elements of a newly created array are initialized to the data type unspecified.

A dynamic portion of an array contains elements indexed with expressions evaluating to any VAXTPU data type except unspecified, learn, pattern, or program. Dynamic array elements are dynamically created and deleted as needed. To create a dynamic array element, assign a value to an element of an existing array. For example, the following statement creates a

dynamic element in the array *bat* indexed by the string "bar" and assigns the integer value 10 to the element:

```
bat{"bar"} := 10;
```

To create an array with both static and dynamic elements, first create the static portion of the array. Then use assignment statements to create as many dynamic elements as you wish. For example, the following code fragment creates an array stored in the variable *small_array*. The array has 15 static elements and one dynamic element. The first static element is given the value 10. The dynamic element is indexed by the string "fred" and contains the value 100.

```
small_array := CREATE_ARRAY (15);
small_array{1} := 10;
small_array{"fred"} := 100;
```

To delete a dynamic array element, assign to it the constant TPU$K_UNSPECIFIED, which is of type unspecified.

One array can contain elements indexed with several data types. For example, you can create an array containing elements indexed with integers, buffers, windows, markers, and strings. An array element can be of any data type. All array elements of a newly created array are of type unspecified.

If the same array has been assigned to more than one variable, VAXTPU does not create multiple copies of the array. Instead, each variable points to the array that has been assigned to it. VAXTPU arrays are reference counted, meaning that each array has a counter keeping track of how many variables point to it. VAXTPU arrays are autodelete data types, meaning that when no variables point to an array, the array is deleted automatically. You can also delete an array explicitly using the DELETE built-in. For example, the following statement deletes the array *bat*:

```
DELETE (bat);
```

If you delete an array that still has variables pointing to it, the variables receive the data type unspecified after the deletion.

If you modify an array pointed to by more than one variable, modifications made using one variable show up when another variable references the modified element. To duplicate an array, you must write a procedure creating a new array and copying the old array's elements to the new array.

To refer to an array element, use the array variable name followed by an index expression enclosed in braces or parentheses. For example, if *bar* were a variable of type marker, the following statement would assign the integer value 10 to the element indexed by *bar*:

```
bat{bar} := 10;
```

You can perform the same operations on array elements that you can on other VAXTPU variables, with one exception—you cannot make partial pattern assignments to array elements.

See Chapter 2 for additional information about arrays.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	CREATE_ARRAY accepts no more than two arguments.
TPU$_NEEDTOASSIGN	ERROR	CREATE_ARRAY must appear on the right-hand side of an assignment statement.
TPU$_INVPARAM	ERROR	The arguments to CREATE_ ARRAY must be integers.
TPU$_MINVALUE	WARNING	The first argument to CREATE_ ARRAY must be 1 or greater.
TPU$_MAXVALUE	WARNING	The first argument to CREATE_ ARRAY must be no greater than 65,535.
TPU$_GETMEM	ERROR	VAXTPU could not create the array because VAXTPU did not have enough memory.

EXAMPLES

1
```
array1 := CREATE_ARRAY;
```

This assignment statement above creates an array and assigns it to the variable *array1*.

2
```
array2 := CREATE_ARRAY( 10 );
```

This assignment statement also creates an array. This array has ten predeclared integer-indexed elements that can be processed quickly by VAXTPU. It can also be indexed by any other VAXTPU data type except pattern, program, learn, and unspecified.

3
```
array3 := CREATE_ARRAY( 11, -5 );
```

This assignment statement creates an array that can be indexed by the integers –5 through 5. It can also be indexed by any other VAXTPU data type other than patterns and learn sequences.

CREATE_BUFFER

Defines a new work space for editing text. You can create an empty buffer or you can associate an input file name with the buffer. CREATE_BUFFER optionally returns a buffer.

FORMAT

[buffer2 :=] CREATE_BUFFER *(string1 [,string2 [,buffer1] [,string3]])*

PARAMETERS

string1

A string representing the name of the buffer you want to create.

string2

A string representing the file specification of an input file that is read into the buffer.

buffer1

The buffer that you want to use as a template for the buffer to be created. The information copied from the template buffer includes the following:

- End-of-buffer text
- Direction (FORWARD/REVERSE)
- Text entry mode (INSERT/OVERSTRIKE)
- Margins (right and left)
- Margin action routines
- Maximum number of lines
- Write-on-exit status (NO_WRITE)
- Modifiable status
- Tab stops
- Key map list

VAXTPU does not copy the following attributes of the template buffer to the new buffer:

- Buffer contents
- Marks or ranges
- Input file name
- Mapping to windows
- Cursor position
- Editing point
- Associated subprocesses
- Buffer name
- Permanent status, if that is an attribute of the template buffer

- System status, if that is an attribute of the template buffer

string3

The name of the journal file to be used with the buffer. Note that VAXTPU does not copy the journal file name from the template buffer. Instead, CREATE_BUFFER uses string3 as the new journal file name. If you do not specify string3, VAXTPU names the journal file using its journal file naming algorithm. For more information on the naming algorithm, see Section 1.7.1 in Chapter 1.

EVE turns on buffer change journaling by default for each new buffer. However, the CREATE_BUFFER built-in does not automatically turn on journaling; if you are layering directly on VAXTPU, your application must use SET (JOURNALING) to turn journaling on.

Caution: Journal files contain a record of *all* information being edited. Therefore, when editing files containing secure or confidential data, be sure to keep the journal files secure as well.

---

return value

The buffer created by CREATE_BUFFER.

DESCRIPTION

Although you do not have to assign the buffer that you create to a variable, you need to make a variable assignment if you want to refer to the buffer for future use. The buffer variable on the left-hand side of an assignment statement is the item that you must use when you specify a buffer as a parameter for other VAXTPU built-in procedures. For example, to move to a buffer for editing, enter the buffer variable after the built-in procedure POSITION:

```
my_buffer_variable := CREATE_BUFFER ("my_buffer_name", "my_file_name");

POSITION (my_buffer_variable);
```

The buffer name that you specify as the first parameter for the built-in procedure CREATE_BUFFER (for example, "my_buffer_name" is used by VAXTPU to identify the buffer on the status line). You can change the status line with the built-in procedure SET (STATUS_LINE).

If you want to skip an optional parameter and specify a subsequent optional parameter, you must use a comma as a placeholder for the skipped parameter.

You can create multiple buffers. Buffers can be empty or they can contain text. The current buffer is the buffer in which any VAXTPU commands that you execute take effect (unless you specify another buffer). Only one buffer can be the current buffer. See the built-in procedure CURRENT_BUFFER for more information.

A buffer is visible when it is associated with a window that is mapped to the screen. A buffer can be associated with multiple windows, in which case any edits that you make to the buffer are reflected in all of the windows in which the buffer is visible. To get a list of all the buffers in your editing context, use the built-in procedure SHOW (BUFFERS).

The following keywords used with the built-in procedure SET allow you to establish attributes for buffers. The text describes the default for the attributes:

- SET (EOB_TEXT, buffer, string) — The default end-of-buffer text is [EOB].
- SET (ERASE_UNMODIFIABLE, buffer, { ON / OFF }) — By default, unmodifiable records can be deleted from buffers by built-ins such as ERASE_LINE.
- SET (FORWARD, buffer) — The default direction is forward.
- SET (INSERT, buffer) — The default mode of text entry is insert.
- SET (JOURNALING, buffer, { ON / OFF }) — By default, buffer change journaling is turned off.
- SET (LEFT_MARGIN, buffer, integer) — The default left margin is 1 (that is, the left margin is set in column 1).
- SET (LEFT_MARGIN_ACTION, buffer, program_source) — By default, buffers do not have left margin action routines.
- SET (MARGINS, buffer, integer1, integer2) — The default left margin is 1 and the default right margin is 80.
- SET (MAX_LINES, buffer, integer) — The default maximum number of lines is 0 (in other words, this feature is turned off).
- SET (MODIFIABLE, buffer, { ON / OFF }) — By default, a buffer can be modified. Using the OFF keyword makes a buffer unmodifiable.
- SET (MODIFIED, buffer, { ON / OFF }) — Turns on or turns off the bit indicating that the specified buffer has been modified.
- SET (NO_WRITE, buffer [[,keyword]]) — By default, when you exit from VAXTPU, the buffer is written if it has been modified.
- SET (OUTPUT_FILE, buffer, string) — The default output file is the input file specification with the highest existing version number for that file plus 1.
- SET (OVERSTRIKE, buffer) — The default mode of text entry is insert.
- SET (PERMANENT, buffer) — By default, the buffer can be deleted.
- SET (RECORD_ATTRIBUTE, marker, range, buffer)
- SET (REVERSE, buffer) — The default direction is forward.
- SET (RIGHT_MARGIN, buffer, integer) — The default right margin is 80.
- SET (RIGHT_MARGIN_ACTION, buffer, program_source) — By default, buffers do not have right margin action routines.
- SET (SYSTEM, buffer) — By default, the buffer is a user buffer.

- SET (TAB_STOPS, buffer, { string / integer }) — The default tab stops are set every eight character positions.

See the built-in procedure SET for more information on these keywords.

SIGNALED ERRORS

TPU$_DUPBUFNAME	WARNING	First argument to the CREATE_BUFFER built-in must be a unique string.
TPU$_TRUNCATE	WARNING	A record was truncated to the maximum record length.
TPU$_TOOMANY	ERROR	The CREATE_BUFFER built-in takes a maximum of two arguments.
TPU$_TOOFEW	ERROR	The CREATE_BUFFER built-in requires at least one argument.
TPU$_INVPARAM	ERROR	The CREATE_BUFFER built-in accepts parameters of type string or buffer only.
TPU$_GETMEM	ERROR	VAXTPU ran out of virtual memory trying to create the buffer.
TPU$_OPENIN	ERROR	CREATE_BUFFER did not open the specified input file.

EXAMPLES

1
```
nb := CREATE_BUFFER ("new_buffer", "login.com")
```

This statement creates a buffer called NEW_BUFFER and stores a pointer to the buffer in the variable *nb*. Use the variable *nb* when you want to specify this buffer as a parameter for VAXTPU built-in procedures. The file specification "LOGIN.COM" is the input file for NEW_BUFFER.

2
```
default_buffer := CREATE_BUFFER ("defaults");
SET (REVERSE, default_buffer);
b := CREATE_BUFFER ("buffer", "", default_buffer);
```

The first statement in this example creates a buffer called DEFAULTS and stores a pointer to the buffer in the variable *default_buffer*. The second statement sets the direction of *default_buffer* to reverse. The third statement creates a buffer called BUFFER and stores a pointer to the buffer in the variable *b*. This statement takes default information from *default_buffer*. Note that buffer *b* does not receive any text, marks, or ranges from the buffer *default_buffer*.

3
```
PROCEDURE user_help_buffer
   help_buf := CREATE_BUFFER("help_buf");
   SET (EOB_TEXT, help_buf, "[End of HELP]");
   SET (NO_WRITE, help_buf);
   SET (SYSTEM, help_buf);
ENDPROCEDURE;
```

This procedure creates the help buffer.

4
```
buf1 := CREATE_BUFFER ("Scratch",,,"Scratch_jl.jl");
```

This statement creates a buffer named Scratch and directs VAXTPU to name the associated buffer change journal file Scratch_jl.jl. Note that you must use commas as placeholders for the two unspecified optional parameters. Note, too, that by default VAXTPU puts journal files in the directory defined by the logical name TPU$JOURNAL. By default, TPU$JOURNAL points to the same directory that SYS$SCRATCH points to. You can reassign TPU$JOURNAL to point to a different directory.

5
```
defaults_buffer := CREATE_BUFFER ("Defaults");
SET (EOB_TEXT, defaults_buffer, "[That's all, folks!]");
user_buffer := CREATE_BUFFER ("User1.txt", "", defaults_buffer);
```

This code fragment creates a template buffer called Defaults, changes the end-of-buffer text for the template buffer, and then creates a user buffer. The user buffer is created with the same end-of-buffer text that the defaults buffer has.

CREATE_KEY_MAP

Creates and names a key map. CREATE_KEY_MAP optionally returns a string that is the name of the key map created.

FORMAT

[string2 :=] CREATE_KEY_MAP *(string1)*

PARAMETER

string1
A quoted string, or a variable name representing a string constant, that specifies the name of the key map you create.

return value

A string that is the name of the key map created.

DESCRIPTION

A **key map** is a set of key definitions. Key maps allow you to manipulate key definitions as a group. Key maps and their key definitions are saved in section files. The default key map for VAXTPU is TPU$KEY_MAP, contained in the default key map list TPU$KEY_MAP_LIST. See the description on key map lists.

The EVE editor does not use the default key map, TPU$KEY_MAP. In EVE, the name of a key map is not the same as the variable that contains the key map. For example, the EVE variable EVE$X_USER_KEYS contains the key map named EVE$USER_KEYS, which stores the user's key definitions. EVE stores all its key maps in the default key map list, TPU$KEY_MAP_LIST. However, the default key map, TPU$KEY_MAP, is removed from the default key map list by the standard EVE section file.

When you create a key map, its keys are undefined. Each key map can hold definitions for all characters in the DEC Multinational Character Set, and all the keypad keys and the function keys, in both their shifted and unshifted forms. Each key map has its own name (a string). This name cannot be the same as that of either another key map or a key map list.

SIGNALED ERRORS

TPU$_DUPKEYMAP	WARNING	A key map with this name already exists.
TPU$_TOOFEW	ERROR	Too few arguments passed to the CREATE_KEY_MAP built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the CREATE_KEY_MAP built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the CREATE_KEY_MAP built-in.

EXAMPLE

```
PROCEDURE init_sample_key_map

sample_key_map := CREATE_KEY_MAP ("sample_key_map");

DEFINE_KEY ("EXIT", CTRL_Z_KEY, "Exit application", sample_key_map);
DEFINE_KEY ("COPY_TEXT ('XYZZY')", CTRL_B_KEY, "Magic Word", sample_key_map);

ENDPROCEDURE;
```

This procedure creates a key map and defines two keys in the key map. The name of the key map is stored in the variable *sample_key_map*.

CREATE_KEY_MAP_LIST

Creates and names a key map list, and also specifies the initial key maps in the key map list it creates. CREATE_KEY_MAP_LIST optionally returns a string that is the name of the key map list created.

FORMAT

[string3 :=]
CREATE_KEY_MAP_LIST *(string1, string2 [,...])*

PARAMETERS

string1

A quoted string, or a variable name representing a string constant, that specifies the name of the key map list that you create.

string2

Strings that specify the names of the initial key maps within the key map list you create.

return value

A string that is the name of the key map list created.

DESCRIPTION

A **key map list** is an ordered set of key maps. Key map lists allow you to change the procedures bound to your keys. To find the definition of a given key, VAXTPU searches through the key maps in the specified or default key map list until VAXTPU either finds a definition for the key or reaches the end of the last key map in the list.

VAXTPU provides the default key map list, TPU$KEY_MAP_LIST, containing the default key map, TPU$KEY_MAP. (See the description of the built-in procedure CREATE_KEY_MAP for more information on key maps.)

The built-in procedure CREATE_KEY_MAP_LIST creates a new key map list, names the key map list, and specifies the initial key maps contained in the list.

Key map lists store directions on what VAXTPU is to do when the user presses an undefined key associated with a printable character. By default, a key map list directs VAXTPU to insert undefined printable characters into the current buffer. To change the default, use the built-in procedure SET (SELF_INSERT).

A newly created key map list is not bound to any buffer. To bind a key map list to a buffer, use the built-in procedure SET (KEY_MAP_LIST). When you use the POSITION built-in to select a current buffer, the key map list that is bound to the buffer is automatically activated.

A newly created key map list has no procedure defined to be called when an undefined key is referenced. You can define such a procedure with the built-in procedure SET (UNDEFINED_KEY). The default is to display the message "key has no definition."

Key map lists are saved in section files, along with any undefined key procedures and the SELF_INSERT settings.

SIGNALED ERRORS

TPU$_DUPKEYMAP	WARNING	The string argument is already defined as a key map.
TPU$_DUPKEYMAPLIST	WARNING	The string argument is already defined as a key map list.
TPU$_NOKEYMAP	WARNING	The string argument is not a defined key map.
TPU$_TOOFEW	ERROR	Too few arguments passed to the CREATE_KEY_MAP_LIST built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the CREATE_KEY_MAP_LIST built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the CREATE_KEY_MAP_LIST built-in.

EXAMPLE

```
PROCEDURE init_help_key_map_list

help_user_keys := CREATE_KEY_MAP ("help_user_keys");
help_keys := CREATE_KEY_MAP ("help_keys");
help_key_list := CREATE_KEY_MAP_LIST ("help_key_list", help_user_keys,
                 help_keys);
ENDPROCEDURE;
```

This procedure creates two key maps and groups them into a key map list.

CREATE_PROCESS

Starts a subprocess and associates a buffer with it. You can optionally specify an initial command to send to the subprocess. CREATE_PROCESS returns a process.

FORMAT

process := CREATE_PROCESS *(buffer [,string])*

PARAMETERS

buffer

The buffer in which VAXTPU stores output from the subprocess.

string

A quoted string, a variable name representing a string constant, or an expression that evaluates to a string, that represents the first command that you want to send to the subprocess. If you do not want to include the first command when you use the built-in procedure CREATE_PROCESS, see the built-in procedure SEND for a description of how to send the first or subsequent commands to a subprocess.

return value

The process created.

DESCRIPTION

You can create multiple subprocesses. When you exit from VAXTPU, any subprocesses you have created with CREATE_PROCESS are deleted. If you want to remove a subprocess before exiting, use the built-in procedure DELETE with the process as a parameter (DELETE (p1)), or set the variable to integer zero as follows:

```
proc1 := 0
```

CREATE_PROCESS creates a subprocess of a VAXTPU session and all of the output from the subprocess goes into a VAXTPU buffer. You cannot run a program or utility that takes over control of the screen from a process created with this built-in procedure. (See Chapter 2 for a list of subprocess restrictions.) You can, however, use the built-in procedure SPAWN to create a subprocess that suspends your VAXTPU process and places you directly at DCL level. You can then run programs such as FMS or PHONE that control the whole screen.

SIGNALED ERRORS

TPU$_DUPBUFNAME	WARNING	First argument must be a unique string.
TPU$_CREATEFAIL	WARNING	Unable to activate the subprocess.
TPU$_TOOFEW	ERROR	Too few arguments passed to the CREATE_PROCESS built-in.

TPU$_TOOMANY	ERROR	Too many arguments passed to the CREATE_PROCESS built-in.
TPU$_NEEDTOASSIGN	ERROR	The CREATE_PROCESS built-in call must be on the right-hand side of an assignment statement.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the CREATE_PROCESS built-in.
TPU$_CAPTIVE	WARNING	Unable to create a subprocess in a captive account.
TPU$_NOTMODIFIABLE	WARNING	Attempt to change unmodifiable buffer. You can only write the output of the subprocess to a modifiable buffer.
TPU$_NOPROCESS	WARNING	No subprocess to interact with. The process was deleted between the time that it was created and when VAXTPU attempted to send information to it.
TPU$_SENDFAIL	WARNING	Unable to send data to the subprocess.
TPU$_DELETEFAIL	WARNING	Unable to terminate the subprocess.

EXAMPLES

1
```
my_mail_process := CREATE_PROCESS (second_buffer, "mail")
```

This assignment statement creates a subprocess and specifies SECOND_BUFFER as the buffer in which the output from the subprocess is stored. It also sends the DCL MAIL command as the first command to be executed.

2
```
! Create a buffer to hold the output from the DCL commands
! "SET NOON" and "DIRECTORY".

PROCEDURE user_dcl_process
   dcl_buffer := CREATE_BUFFER ("dcl_buffer");
   MAP (main_window, dcl_buffer);
   my_dcl_process := CREATE_PROCESS (dcl_buffer, "SET NOON");
   MESSAGE ("Creating DCL subprocess...");
   SEND ("DIRECTORY", my_dcl_process);
ENDPROCEDURE;
```

This procedure creates a buffer to hold the output from the DCL commands executed by the subprocess.

CREATE_RANGE

Returns a range that includes two delimiters and all the characters between them, and sets the video attributes for displaying the characters when they are visible on the screen. A range delimiter can be a marker, the beginning or end of a line, or the beginning or end of a buffer. The beginning and ending delimiters do not have to be of the same type but must be in the same buffer.

FORMAT

range := CREATE_RANGE ({ *marker1* / *keyword1* }, { *marker2* / *keyword1* } [, *keyword2*])

PARAMETERS

marker1

The marker indicating the point in the buffer where the range begins.

marker2

The marker indicating the point in the buffer where the range ends.

keyword1

A keyword indicating the point in the buffer where you want the range to begin or end. Table 7–1 shows the valid keywords and their meanings.

Table 7–1 CREATE_RANGE Keyword Parameters

Keyword	Meaning
LINE_BEGIN	The beginning of the current buffer's current line.
LINE_END	The end of the current buffer's current line.
BUFFER_BEGIN	Line 1, offset 0 in the current buffer. This is the first position where a character could be inserted, regardless of whether there is a character there. This is the same as the point referred to by BEGINNING_OF (CURRENT_BUFFER).
BUFFER_END	The last position in the buffer where a character could be inserted. This is the same as the point referred to by END_OF (CURRENT_BUFFER).

keyword2

The video attribute for the range: BLINK, BOLD, NONE, REVERSE, or UNDERLINE. If you omit the parameter, the default is NONE.

return value

The range created by CREATE_RANGE.

DESCRIPTION

CREATE_RANGE establishes a range that is delimited by the markers you specify. You can create multiple ranges in a buffer. When you apply video attributes to a range, you can see the range if it is in a visible buffer. A range may overlap another range.

If you clear the contents of a range with the built-in procedure ERASE, the range structure still exists. The range and its video attributes, if any, move to the next character or position beyond where the range ended before the range was erased.

To remove the range structure, use the built-in procedure DELETE or set the variable to which the range is assigned to zero (r1 := 0).

In portions of a range that either are associated with nonprintable characters or are not associated with characters at all, VAXTPU does not display any of the video attributes of the range. However, if you insert new characters into portions of a range where the video attributes have not been displayed, the new characters do display the video attributes that apply to the range.

CREATE_RANGE checks whether the markers you specify as parameters are **free markers**. A free marker is a marker not bound to a character. For more information on free markers, see the description of the MARK built-in.

If a marker defining a range is a free marker, VAXTPU ties the range to the character or end-of-line nearest to the free marker, to use as the range delimiter. Note that an end-of-line is not a character but is a point to which a marker can be bound.

SIGNALED ERRORS

TPU$_NOTSAMEBUF	WARNING	First and second marker are in different buffers.
TPU$_TOOFEW	ERROR	CREATE_RANGE requires three parameters.
TPU$_TOOMANY	ERROR	CREATE_RANGE accepts no more than three parameters.
TPU$_NEEDTOASSIGN	ERROR	CREATE_RANGE must appear on the right-hand side of an assignment statement.
TPU$_INVPARAM	ERROR	One of your arguments to CREATE_RANGE is of the wrong type.
TPU$_BADKEY	WARNING	You specified an illegal keyword.

EXAMPLES

```
1   my_range := CREATE_RANGE (start_mark, end_mark, BOLD)
```

This assignment statement creates a range starting at *start_mark* and ending at *end_mark*. When this range is visible on the screen, the characters in the range are bolded.

2
```
PROCEDURE user_erase_to_eob

   LOCAL start_of_range,
         here_to_eob;

   start_of_range := MARK (NONE);
   here_to_EOB    := CREATE_RANGE (start_of_range,
                                   END_OF (CURRENT_BUFFER),
                                   NONE);
   ERASE (here_to_eob);
ENDPROCEDURE;
```

This procedure erases the text in the current buffer, starting at the editing point, and erasing text until the end of the buffer is reached.

3
```
the_range := CREATE_RANGE (BUFFER_BEGIN, mark2, REVERSE);
```

This statement creates a range starting at the first point in the buffer where a character can be inserted and ending at the point marked by *mark2*. If the range is visible on the screen, the characters in it are highlighted with the reverse video attribute.

CREATE_WIDGET

Creates a widget instance. The CREATE_WIDGET built-in has two variants with separate syntaxes. One variant creates and returns a widget using the intrinsics or a XUI Toolkit low-level creation routine. The other variant creates an entire hierarchy of widgets (as defined in an XUI Resource Manager database) and returns the topmost widget.

FORMAT

widget := CREATE_WIDGET *(widget_class, widget_name,*
{ *parent_widget* }
{ *SCREEN* }
[, { *buffer* / *learn_sequence* / *program* / *range* / *string* }]
[, *closure*
[, *widget_args...*]]])

DESCRIPTION

Creates the widget instance you specify, using the intrinsics or an XUI Toolkit low-level creation routine. Although it has been created, the returned widget is not managed and therefore not visible. The application must call the MANAGE_WIDGET built-in to make the widget visible.

FORMAT

widget := CREATE_WIDGET *(resource_manager_name, hierarchy_id,*
{ *parent_widget* }
{ *SCREEN* }
[, { *buffer* / *learn_sequence* / *program* / *range* / *string* }]
[, *closure*
[, *widget_args...*]]])

DESCRIPTION

Creates and returns an entire hierarchy of widgets (as defined in an XUI Resource Manager database) and returns the topmost widget. All children of the returned widget are also created and managed. The topmost widget is not managed, so none of the widgets created is visible.

If you specify one or more callback arguments in your User Interface Language (UIL) file, specify either the routine TPU$WIDGET_INTEGER_CALLBACK or the routine TPU$WIDGET_STRING_CALLBACK. For more information about specifying callbacks, see Chapter 4. For more information about UIL files, see the *VMS DECwindows Guide to Application Programming*.

When you use CREATE_WIDGET to create a widget or hierarchy of widgets organized by the XUI Resource Manager, CREATE_WIDGET uses the XUI Toolkit routine FETCH WIDGET.

PARAMETERS

widget_class
The integer returned by DEFINE_WIDGET_CLASS that specifies the class of widget to be created.

widget_name
A string that is the name to be given to the widget.

parent_widget
The widget that is to be the parent of the newly created widget.

SCREEN
A keyword indicating that the newly created widget is to be the child of VAXTPU's main window widget.

buffer
The buffer containing the interface callback routine. This code is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

learn_sequence
The learn sequence that is the interface callback routine. This is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

program
The program that is the interface callback routine. This is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

range
The range containing the interface callback routine. This is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

string
The string containing the interface callback routine. This is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

closure

A string or integer. VAXTPU passes the value to the application when the widget performs a callback to VAXTPU. For more information about using closures, see Chapter 4.

If you do not specify this parameter, VAXTPU passes the closure value (if any) given to the widget in the UIL file defining the widget. If you specify the closure value with CREATE_WIDGET instead of in the UIL file, all widgets created with the same CREATE_WIDGET call have the same closure value.

widget_args

One or more pairs of resource names and resource values. You can specify a pair in an array or as a pair of separate parameters. If you use an array, you index the array with a string that is the name of the resource you want to set. Note that resource names are case-sensitive. The corresponding array element contains the value you want to assign to that resource. The array can contain any number of elements. If you use a pair of separate parameters, use the following format:

```
resource_name_string, resource_value
```

Arrays and string/value pairs may be interspersed. Each array index and its corresponding element value, or each string and its corresponding value, must be valid widget arguments for the class of widget you are creating.

resource_manager_name

A case-sensitive string that is the name assigned to the widget in the UIL file defining the widget.

hierarchy_id

The hierarchy identifier returned by the SET (DRM_HIERARCHY) built-in. This identifier is passed to the XUI Resource Manager, which uses the identifier to find the resource name in the database.

---

return value

The newly created widget instance.

DESCRIPTION

The case of a widget's name in the User Interface Definition (UID) file must match the case of the widget's name that you specify as a parameter to CREATE_WIDGET. If you specify case sensitive widget names in your UIL file, you must use the same widget name case with CREATE_WIDGET as you used in the UIL file. If you specify case insensitive widget names in your UIL file, the UIL compiler translates all widget names to uppercase, so in this instance you must use uppercase widget names with CREATE_WIDGET. The example in the following subsection specifies case insensitive widget names in the UIL file and specifies an uppercase name for the widget with the CREATE_WIDGET built-in.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_UNDWIDCLA	WARNING	You have specified a widget class integer that is not known to VAXTPU.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NEEDTOASSIGN	ERROR	CREATE_WIDGET must return a value.
TPU$_REQUIRESDECW	ERROR	You can use CREATE_WIDGET only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to CREATE_WIDGET.
TPU$_TOOMANY	ERROR	Too many arguments passed to CREATE_WIDGET.
TPU$_WIDMISMATCH	ERROR	You have specified a widget whose class is not supported.
TPU$_ARGMISMATCH	ERROR	A widget argument was not an array or a string/value pair.
TPU$_COMPILEFAIL	WARNING	Compilation of the widget interface callback routine failed due to syntax errors.
TPU$_NONAMES	WARNING	A widget argument is not supported by the specified widget.

EXAMPLES

1

```
PROCEDURE eve_display_example

LOCAL   example_widget,                 ! Variable assigned to the created widget.
        example_widget_name,            ! The name of the widget assigned
                                        ! to this variable must be uppercase
                                        ! if you specified case insensitive
                                        ! widget names in the UIL file.

        example_hierarchy;              ! XUI Resource Manager
                                        ! hierarchy for this example.

ON_ERROR
    [OTHERWISE]:                        ! Traps errors.
ENDON_ERROR;

! Set the widget hierarchy. The default file spec is "SYS$LIBRARY: .UID"

example_hierarchy := SET (DRM_HIERARCHY, "mynode$dua0:[smith]example");

! The VAXTPU CREATE_WIDGET built-in needs the name of the widget
! defined in the UIL file.

example_widget_name := "EXAMPLE_BOX";   ! The widget EXAMPLE_BOX is
                                        ! defined in the file EXAMPLE.UIL.

! Create the widget if it has not already been created.
```

```
IF GET_INFO (example_widget, "type") <> WIDGET
THEN
    example_widget := CREATE_WIDGET (example_widget_name, example_hierarchy,
                                     SCREEN, eve$kt_callback_routine);

   ! EVE defines eve$callback_dispatch to be EVE's callback routine.
   ! You do not need to define it again if you are extending EVE.

ENDIF;

! Map "example_widget" to the screen using MANAGE_WIDGET.

MANAGE_WIDGET (example_widget);

RETURN (TRUE);

ENDPROCEDURE;
```

This procedure, *eve_display_example*, creates a modal dialog box widget and maps the widget to the VAXTPU screen.

The procedure shows how to use the variant of CREATE_WIDGET that returns an entire widget hierarchy. To create a widget or widget hierarchy using this variant, you must have available the compiled form of a User Interface Language (UIL) file specifying the characteristics of the widgets you want to create. Digital recommends that you use one or more UIL files and the corresponding variant of CREATE_WIDGET whenever possible, because UIL is more efficient and because UIL files make it easier to translate your application into other languages. For more information about compiling and using UIL files, see the *VMS DECwindows Guide to Application Programming*.

```
MODULE    example
VERSION = 'V00-000'

! This is a sample UIL file that creates a message box containing
! the message "Hello World".

NAMES = case_insensitive

VALUE
        example_message         : 'Hello World';

OBJECT
        example_box : message_box {

            arguments {
                default_position = true;        ! puts box in center work area
                ok_label = example_button_label;
                label_label = example_message;
            };
        };

END MODULE;
```

This example shows a sample UIL file describing the modal dialog box called *example_box*. The UIL file specifies where the widget appears on the screen, what label appears on the box's button, and what message the widget displays.

For an example showing how to use the variant of CREATE_WIDGET that calls the XUI Toolkit low-level creation routine, see Example B–2.

CREATE_WINDOW

Defines a screen area called a window. You must specify the screen line number at which the window starts, the length of the window, and whether the status line is to be displayed. CREATE_WINDOW optionally returns the newly created window.

FORMAT

[window :=]
CREATE_WINDOW *(integer1, integer2,* { *ON* / *OFF* } *)*

PARAMETERS

integer1
The screen line number at which the window starts.

integer2
The number of rows in the window.

ON
A keyword directing VAXTPU to display a status line in the new window. The status line occupies the last row of a window. By default, the status line is displayed in reverse video and contains the following information about the buffer that is currently mapped to the window:

- The name of the buffer that is associated with the window
- The name of the file that is associated with the buffer, if one exists

See SET (STATUS_LINE) for information on changing the video attributes of the status line and/or the information displayed on the status line.

OFF
Suppresses the display of the status line.

return value

The window created by CREATE_WINDOW.

DESCRIPTION

CREATE_WINDOW optionally returns the new window. If you want to use the window that you create as a parameter for any other built-in procedure, then you should specify a variable into which the window is returned.

You can create multiple windows on the screen, but only one window can be the current window. The cursor is positioned in the current window. The current window and the current buffer are not necessarily the same.

To make a window visible, you must associate a buffer with the window and map the window to the screen. The following command maps *main_window* to the screen:

```
MAP (main_window, main_buffer)
```

See the built-in procedure MAP for further information.

The following keywords used with the built-in procedure SET allow you to establish attributes for windows. This list shows the defaults for the attributes:

- SET (PAD, window, keyword) — By default, there is no blank padding on the right.
- SET (SCROLL_BAR) — By default, VAXTPU does not create vertical and horizontal scroll bars for a window in the DECwindows environment.
- SET (SCROLL_BAR_AUTO_THUMB) — By default, VAXTPU controls the slider in any scroll bars in a window.
- SET (SCROLLING, window, keyword, integer1, integer2, integer3) — The default cursor limit for scrolling at the top of the screen is the first line of the window; the default cursor limit for scrolling at the bottom of the screen is the bottom line of the window. If the terminal type you are using does not allow you to set scrolling regions, the window is repainted.
- SET (STATUS_LINE, window, keyword, string) — The status line may be ON or OFF according to the keyword specified for the built-in procedure CREATE_WINDOW. See the preceding description of the keyword ON for information about the default attributes of a status line.
- SET (TEXT, window, keyword) — By default, the text is set to BLANK_TABS (tabs are displayed as blank spaces).
- SET (VIDEO, window, keyword) — There are no video attributes by default.
- SET (WIDTH, window, integer) — By default, the width is the same as the physical width of the terminal screen when the window is created.

See the built-in procedure SET for more information on these keywords.

Using the SHIFT built-in, you can display text that lies to the right of the window's right edge in an unshifted window. For information on using SHIFT, see the description of the built-in in this chapter.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The CREATE_WINDOW built-in requires exactly three parameters.
TPU$_TOOMANY	ERROR	The CREATE_WINDOW built-in accepts exactly three parameters.
TPU$_BADKEY	ERROR	The keyword must be either ON or OFF.

TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADWINDLEN	WARNING	Invalid window length.
TPU$_BADFIRSTLINE	WARNING	Invalid first line for window.

EXAMPLES

1
```
new_window := CREATE_WINDOW (11, 10, ON)
```

This assignment statement creates a window that starts at screen line 11 and is 10 rows long, and assigns it to the variable *new_window*. A status line is displayed as the last line of the window. To make this window visible, you must associate an existing buffer with it and map the window to the screen with the following command:

```
MAP (new_window, buffer_variable)
```

2
```
PROCEDURE user_make_window
        new_window := CREATE_WINDOW(1, 21, OFF);
        SET (TEXT, new_window, GRAPHIC_TABS);
        new_buffer := CREATE_BUFFER ("user_buffer_name");
        SET (NO_WRITE, new_buffer);
        MAP (new_window, new_buffer);
ENDPROCEDURE;
```

This procedure creates a window called *new_window* that starts at screen line 1 and is 21 lines long. No status line is displayed. Tabs are displayed as special graphic characters. The buffer *new_buffer*, which is set to NO_WRITE, is associated with the window and the window is mapped to the screen.

CURRENT_BUFFER

Returns the buffer in which you are currently positioned.

FORMAT

buffer := CURRENT_BUFFER

PARAMETERS

None.

return value

The buffer in which you are currently positioned.

DESCRIPTION

The current buffer is the work space in which any VAXTPU statements you execute take effect. The editing point is in the current buffer. Note that the editing point is not necessarily the same as the cursor position.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	CURRENT_BUFFER takes no parameters.
TPU$_NEEDTOASSIGN	ERROR	The CURRENT_BUFFER built-in must be on the right-hand side of an assignment statement.
TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.

EXAMPLES

1
```
my_cur_buf := CURRENT_BUFFER
```

This assignment statement stores a pointer to the current buffer in the variable *my_cur_buf*.

2
```
SHOW (CURRENT_BUFFER)
```

This statement returns the buffer in which you are currently positioned and uses that buffer as the parameter for the built-in procedure SHOW.

3
```
PROCEDURE user_toggle_direction
        IF CURRENT_DIRECTION = FORWARD
        THEN
            SET (REVERSE, CURRENT_BUFFER);
        ELSE
            SET (FORWARD, CURRENT_BUFFER);
        ENDIF;
ENDPROCEDURE;
```

This procedure reverses the direction of the current buffer.

CURRENT_CHARACTER

Returns the character at the editing point in the current buffer.

FORMAT

string := CURRENT_CHARACTER

PARAMETERS

None.

return value

A string consisting of the character at the editing point in the current buffer.

DESCRIPTION

The editing point is the character position in the current buffer at which most editing operations are carried out. Each buffer maintains its own editing point, but only the editing point in the current buffer is the active editing point. An editing point, which always refers to a character position in a buffer, is not necessarily the same as the cursor position, which always refers to a location in a window. For more information on the distinction between the editing point and the cursor position, see Chapter 6.

If the editing point is at the end of a line, CURRENT_CHARACTER returns a null string. If the editing point is at the end of a buffer, CURRENT_CHARACTER returns a null string and also signals a warning.

Using CURRENT_CHARACTER may cause VAXTPU to insert padding spaces or blank lines in the buffer. CURRENT_CHARACTER causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	CURRENT_CHARACTER takes no parameters.
TPU$_NEEDTOASSIGN	ERROR	The CURRENT_CHARACTER built-in must be on the right-hand side of an assignment statement.
TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.
TPU$_NOEOBSTR	WARNING	You are positioned at the EOB (end-of-buffer) mark.

EXAMPLES

1
```
my_cur_char := CURRENT_CHARACTER
```

This assignment statement stores the string that represents the editing point in the variable *my_cur_char*.

2
```
MESSAGE (CURRENT_CHARACTER)
```

This statement returns the string that represents the editing point and uses this string as the parameter for the built-in procedure MESSAGE.

3
```
PROCEDURE user_display_current_character
! This procedure returns the ASCII character in the editing point.
   ascii_char := CURRENT_CHARACTER;
   IF ascii_char <> ""
      THEN
         MESSAGE ("The current character is '" + ascii_char + "'");
      ELSE
         MESSAGE ("There is no current character.");
   ENDIF;
ENDPROCEDURE;
```

This procedure writes the character that is at the current character position into the message area.

CURRENT_COLUMN

Returns an integer that is the current column number of the cursor position on the screen.

FORMAT

integer := CURRENT_COLUMN

PARAMETERS

None.

return value

An integer that is the column number of the current cursor position on the screen.

DESCRIPTION

The current column is the column at which the cursor is positioned on the screen. The column numbers range from 1 on the extreme left of the screen to the maximum value allowed for the terminal type you are using on the extreme right of the screen.

The value returned by the built-in procedure CURRENT_COLUMN and the value returned by GET_INFO (SCREEN, "current_column") are equivalent.

When used in a procedure, CURRENT_COLUMN does not necessarily return the position where the cursor has been placed by other statements in the procedure. VAXTPU generally does not update the screen until all statements in a procedure are executed. If you want the cursor position to reflect the actual editing location, put an UPDATE statement in your procedure immediately before any statements containing CURRENT_COLUMN, as follows:

```
UPDATE (CURRENT_WINDOW);
```

If you do not want to update a window to get the current value for CURRENT_COLUMN, you can use the built-in GET_INFO (buffer_variable, "offset_column"). This built-in returns the column number that the current offset in the buffer would have if it were mapped to a window, and if you were to force a screen update. Note, however, that this built-in returns an accurate value only if both of the following conditions are true:

- You are using bound cursor movement (MOVE_VERTICAL, MOVE_HORIZONTAL) or other built-in procedures that cause cursor movement because of character movement within a buffer.
- The window is not shifted.

The built-in GET_INFO (window_variable, "current_column") does not necessarily return the column number that the cursor would occupy if you caused an explicit screen update.

If a window is shifted, CURRENT_COLUMN still returns the current column number of the cursor on the screen. However, the value returned by x := GET_INFO (buffer, "offset_column") includes the number of columns by which the window is shifted. For example, if a window is shifted to the left by 8 columns, CURRENT_COLUMN returns the value 1, while x := GET_INFO (buffer, "offset_column") returns the value 9.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	CURRENT_COLUMN takes no parameters.
TPU$_NEEDTOASSIGN	ERROR	The CURRENT_COLUMN built-in must be on the right-hand side of an assignment statement.
TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.

EXAMPLES

1
```
my_cur_col := CURRENT_COLUMN
```

This assignment statement stores the column position of the cursor in the variable *my_cur_col*.

2
```
MESSAGE (STR (CURRENT_COLUMN))
```

This statement combines three VAXTPU built-in procedures. CURRENT_COLUMN returns the integer that is the current column position, STR converts the integer to a string, and MESSAGE writes this string to the message buffer.

3
```
PROCEDURE user_split_line
   LOCAL old_position, new_position;

   SPLIT_LINE;
   IF (CURRENT_ROW = 1) AND (CURRENT_COLUMN = 1)
   THEN
       old_position := MARK (NONE);
       SCROLL (CURRENT_WINDOW, -1);
       new_position := MARK (NONE);
       !Make sure we scrolled before doing CURSOR_VERTICAL
          IF new_position <> old_position
          THEN
              CURSOR_VERTICAL (1);
          ENDIF;
   ENDIF;
ENDPROCEDURE;
```

This procedure splits a line at the editing point. If the editing point is row 1, column 1, the procedure causes the screen to scroll.

CURRENT_DIRECTION

Returns a keyword (FORWARD or REVERSE) that indicates the current direction of the current buffer. See also the descriptions of the built-in procedures SET (FORWARD) and SET (REVERSE).

FORMAT

keyword := CURRENT_DIRECTION

PARAMETERS

None.

return value

A keyword (FORWARD or REVERSE) indicating the current direction of the current buffer.

DESCRIPTION

If the keyword FORWARD is returned, the current direction is toward the end of the buffer. If the keyword REVERSE is returned, the current direction is toward the beginning of the buffer.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	CURRENT_DIRECTION takes no parameters.
TPU$_NEEDTOASSIGN	ERROR	The CURRENT_DIRECTION built-in must be on the right-hand side of an assignment statement.
TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.

EXAMPLES

1
```
my_cur_dir := CURRENT_DIRECTION
```

This assignment statement stores in the variable *my_cur_dir* the keyword that indicates whether the current direction setting for the buffer is FORWARD or REVERSE.

2
```
PROCEDURE user_show_direction
    IF CURRENT_DIRECTION = FORWARD
    THEN
        my_message1 := MESSAGE ("Forward");
    ELSE
        my_message2 := MESSAGE ("Reverse");
    ENDIF;
ENDPROCEDURE;
```

This procedure writes to the message buffer a message indicating the current direction of character movement in the buffer.

CURRENT_LINE

Returns a string that represents the current line. The current line is the line that contains the editing point.

FORMAT

string := CURRENT_LINE

PARAMETERS

None.

return value

A string representing the current line.

DESCRIPTION

If you are positioned on a line that has a length of 0, CURRENT_LINE returns a null string. If you are positioned at the end of the buffer, CURRENT_LINE returns a null string and also signals a warning.

Using CURRENT_LINE may cause VAXTPU to insert padding spaces or blank lines in the buffer. CURRENT_LINE causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	CURRENT_LINE takes no parameters.
TPU$_NEEDTOASSIGN	ERROR	The CURRENT_LINE built-in must be on the right-hand side of an assignment statement.
TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.
TPU$_NOEOBSTR	WARNING	You are positioned at or beyond the EOB (end-of-buffer) mark.

EXAMPLES

1

```
my_cur_lin := CURRENT_LINE
```

This assignment statement stores in the variable *my_cur_lin* the string that represents the current line. The current line is the line in the current buffer that contains the editing point.

```
2  PROCEDURE user_runoff_line
   IF LENGTH (CURRENT_LINE) < 2
   THEN
      user_runoff_line := 0;
   ELSE
      IF CURRENT_CHARACTER <> "."
      THEN
         user_runoff_line := 0;
      ELSE
         MOVE_HORIZONTAL (1);
         IF INDEX
            ("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!;",
            CURRENT_CHARACTER) = 0
         THEN
            user_runoff_line := 0;
         ELSE
            user_runoff_line := 1;
         ENDIF;
         MOVE_HORIZONTAL (-1);
      ENDIF;
   ENDIF;
   ENDPROCEDURE;
```

This procedure returns true if the current line has the format of a DSR command (starts with a period followed by an alphabetic character, a semicolon, or an exclamation point). If not, the procedure returns false. The procedure assumes that the cursor was at the beginning of the line, and moves it back to the beginning of the line when done.

CURRENT_OFFSET

Returns an integer for the offset of the editing point within the current line.

FORMAT

integer := CURRENT_OFFSET

PARAMETERS

None.

return value

An integer that is the offset of the editing point within the current line.

DESCRIPTION

The current offset is the number of positions a character is located from the first character position in the current line (offset 0). In VAXTPU, the leftmost character position is offset 0, and this offset is increased by 1 for each character position (including the TAB character) to the right. VAXTPU numbers columns starting with the leftmost position on the screen where a character could be placed, regardless of where the margin is. This leftmost position is numbered 1.

Note: The current offset value is not the same as the position of the cursor on the screen. See the CURRENT_COLUMN built-in if you want to determine where the cursor is. For example, if you have a line with a left margin of 10 and if the cursor is on the first character in that line, then CURRENT_OFFSET returns 0, while CURRENT_COLUMN returns 10.

Using CURRENT_OFFSET may cause VAXTPU to insert padding spaces or blank lines in the buffer. CURRENT_OFFSET causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

If you are using an interface with free cursor motion, when you move beyond the end of a line CURRENT_OFFSET makes the current cursor position the new end-of-line.

If the current offset equals the length of the current line, you are positioned at the end of the line.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	CURRENT_OFFSET takes no parameters.
TPU$_NEEDTOASSIGN	ERROR	The CURRENT_OFFSET built-in must be on the right-hand side of an assignment statement.
TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.

EXAMPLES

1
```
my_cur_off := CURRENT_OFFSET
```

This assignment statement stores the integer that is the offset position of the current character in the variable *my_cur_off*.

2
```
PROCEDURE user_delete
   IF CURRENT_OFFSET = 0
   THEN
       APPEND_LINE;
   ELSE
       ERASE_CHARACTER (-1);
   ENDIF;
ENDPROCEDURE;
```

This procedure uses the built-in procedure CURRENT_OFFSET to determine whether the editing position is at the beginning of a line. (For more information on the difference between the editing position and the current cursor position, see Chapter 6.) If the position is at the beginning, the procedure appends the current line to the previous line; otherwise, it deletes the previous character. Compare this procedure with the procedure used as an example for the built-in procedure APPEND_LINE.

CURRENT_ROW

Returns an integer that is the screen line on which the cursor is located.

FORMAT

integer := CURRENT_ROW

PARAMETERS

None.

return value

An integer representing the screen line on which the cursor is located.

DESCRIPTION

The current row is the screen line on which the cursor is located. The screen lines are numbered from 1 at the top of the screen to the maximum number of lines available on the terminal. You can get the value of the current row by using the built-in procedure GET_INFO (SCREEN, "current_row").

When used in a procedure, CURRENT_ROW does not necessarily return the position where the cursor has been placed by other statements in the procedure. The reason that the value returned by CURRENT_ROW may not be the current value is that VAXTPU generally does not update the screen until all statements in a procedure are executed. If you want the cursor position to reflect the actual editing location, put an UPDATE statement in your procedure immediately before any statements containing CURRENT_ROW, as follows:

```
UPDATE (CURRENT_WINDOW);
```

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	The CURRENT_ROW built-in must be on the right-hand side of an assignment statement.
TPU$_TOOMANY	ERROR	CURRENT_ROW takes no parameters.

EXAMPLES

1

```
my_cur_row := CURRENT_ROW
```

This assignment statement stores in the variable *my_cur_row* the integer that is the screen line number on which the cursor is located.

```
2  PROCEDURE user_go_up
      IF CURRENT_ROW = GET_INFO (CURRENT_WINDOW, "visible_top")
      THEN
          SCROLL (CURRENT_WINDOW, -1);
      ELSE
            CURSOR_VERTICAL (-1);
      ENDIF;
   ENDPROCEDURE;


3  PROCEDURE user_go_down
      IF CURRENT_ROW = GET_INFO (CURRENT_WINDOW, "visible_bottom")
      THEN
          SCROLL (CURRENT_WINDOW, 1);
      ELSE
            CURSOR_VERTICAL (1);
      ENDIF;
   ENDPROCEDURE;
```

These procedures cause the cursor to move up or down the screen. Because CURSOR_VERTICAL crosses window boundaries, you must use the built-in procedure SCROLL to keep the cursor motion within a single window if you are using free cursor motion. (See CURSOR_HORIZONTAL and CURSOR_VERTICAL.) If the movement of the cursor would take it outside the window, the preceding procedures scroll text into the window to keep the cursor visible. You can bind these procedures to a key so that the cursor motion can be accomplished with a single keystroke.

CURRENT_WINDOW

Returns the window in which the cursor is visible.

FORMAT

window := CURRENT_WINDOW

PARAMETERS

None.

return value

The window in which the cursor is visible.

DESCRIPTION

The current window is the window on which you have most recently performed one of the following operations:

- Selection using the POSITION built-in
- Mapping to the screen using the MAP built-in
- Adjustment using the ADJUST_WINDOW built-in

The current window contains the cursor at the screen coordinates *current_row* and *current_column*. The current buffer is not necessarily associated with the current window.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	CURRENT_WINDOW takes no parameters.
TPU$_NEEDTOASSIGN	ERROR	The CURRENT_WINDOW built-in must be on the right-hand side of an assignment statement.
TPU$_WINDNOTMAPPED	WARNING	No windows are mapped to the screen.

EXAMPLES

```
1  my_cur_win := CURRENT_WINDOW
```

This assignment statement stores the window that holds the cursor in the variable *my_cur_win*.

```
2  PROCEDURE user_next_screen
      LOCAL how_much_scroll;
      how_much_scroll := GET_INFO (CURRENT_WINDOW, "visible_length");
      SCROLL (CURRENT_WINDOW, how_much_scroll);
   ENDPROCEDURE;
```

This procedure determines the length of the current window and then uses that value as a parameter for the built-in procedure SCROLL.

CURSOR_HORIZONTAL

Moves the cursor position across the screen and optionally returns the cursor movement status.

FORMAT

[integer2 :=] CURSOR_HORIZONTAL *(integer1)*

PARAMETER

integer1

The signed plus or minus integer value that specifies the number of screen columns to move the cursor position. A positive value directs VAXTPU to move the cursor to the right; a negative value directs VAXTPU to move the cursor to the left. The value 0 causes VAXTPU merely to synchronize the active editing point with the cursor position.

return value

An integer representing the number of columns the cursor moved. If VAXTPU cannot move the cursor as many columns as specified by *integer1*, VAXTPU moves the cursor as many columns as possible. VAXTPU allows the return value to be negative. This notation is reserved for future versions of VAXTPU. A negative return value does *not* denote that the cursor moved to the left. Rather, the integer shows the number of spaces that the cursor moved right or left. If the cursor did not move, *integer2* has the value 0. If the CURSOR_HORIZONTAL built-in produces an error, the value of *integer2* is indeterminate.

DESCRIPTION

The CURSOR_HORIZONTAL built-in procedure can be used to provide free cursor movement in a horizontal direction. Free cursor movement means that the cursor is not tied to text, but can move across all available columns in a screen line.

If you move before the beginning of a line, after the end of a line, in the middle of a tab, or beyond the end-of-file mark, other built-ins may cause padding lines or spaces to be added to the buffer.

If you use the CURSOR_HORIZONTAL built-in within a procedure, screen updating occurs as follows:

- When you execute a built-in that modifies the buffer or the editing point before you issue the call to CURSOR_HORIZONTAL, the screen is updated before CURSOR_HORIZONTAL is executed. This action ensures that the horizontal movement of the cursor starts at the correct character position.
- Otherwise, the screen manager does not update the screen until the procedure has finished executing and control is returned to the screen manager.

CURSOR_HORIZONTAL does not move the cursor beyond the left or right edge of the window in which it is located. You cannot move the cursor outside the bounds of a window.

CURSOR_HORIZONTAL has no effect if you use any input device other than a video terminal supported by VAXTPU.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	CURSOR_HORIZONTAL requires one parameter.
TPU$_TOOMANY	ERROR	CURSOR_HORIZONTAL accepts only one parameter.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLES

1
```
int_x := CURSOR_HORIZONTAL (1)
```

This statement moves the cursor position one screen column to the right.

2
```
PROCEDURE user_free_cursor_right
   move_right := CURSOR_HORIZONTAL (1);
ENDPROCEDURE;
```

3
```
PROCEDURE user_free_cursor_left
   move_left := CURSOR_HORIZONTAL (-1);
ENDPROCEDURE;
```

These procedures provide for free cursor motion to the right and to the left. These procedures can be bound to keys (for example, the arrow keys) so that the movement can be accomplished with a single keystroke.

CURSOR_VERTICAL

Moves the cursor position up or down the screen and optionally returns the cursor movement status.

FORMAT

[integer2 :=] CURSOR_VERTICAL *(integer1)*

PARAMETER

integer1

The signed integer value that specifies how many screen lines to move the cursor position. A positive value for *integer1* moves the cursor position down. A negative integer moves the cursor position up.

return value

An integer representing the number of rows that the cursor moved up or down. If VAXTPU could not move the cursor as many rows as specified by *integer1*, VAXTPU moves the cursor as many rows as possible.

If CROSS_WINDOW_BOUNDS is set to ON, CURSOR_VERTICAL may position the cursor to another window. In this case, CURSOR_VERTICAL returns the negative of the number of rows the cursor moved. A negative return value does *not* denote that the cursor moved upward.

If the cursor did not move, *integer2* has the value 0. If the CURSOR_VERTICAL built-in produced an error, the value of *integer2* is indeterminate.

DESCRIPTION

CURSOR_VERTICAL can be used to provide free cursor movement in a vertical direction. Free cursor movement means that the cursor is not tied to text, but that it can move up and down to all lines on the screen that can be edited, whether or not there is text at that column in the new line.

The cursor does not move beyond the top or the bottom edges of the screen. However, CURSOR_VERTICAL can cross window boundaries, depending upon the current setting of the CROSS_WINDOW_BOUNDS flag. See SET (CROSS_WINDOW_BOUNDS) for information on how to set this flag. (Use the POSITION built-in to move the cursor to a different window on the screen.)

When CROSS_WINDOW_BOUNDS is set to ON, CURSOR_VERTICAL can move the cursor position to a new window. The new window in which the cursor is positioned becomes the current window. The column position of the cursor remains unchanged unless vertical movement would position the cursor outside the bounds of a window narrower than the previous window. In this instance, the cursor moves to the left until it is positioned within the right boundary of the narrower window.

When CROSS_WINDOW_BOUNDS is set to OFF, CURSOR_VERTICAL does not move the cursor outside the current window. If the SET (SCROLLING) built-in has been used to set scrolling margins, CURSOR_VERTICAL also attempts to keep the cursor within the scroll margins.

CURSOR_VERTICAL positions the cursor only in screen areas in which editing can occur. For example, CURSOR_VERTICAL does not position the cursor on the status line of a window, in the prompt area, or in an area of the screen that is not part of a window. The blank portion of a segmented window is not considered part of a window for this purpose.

If you use CURSOR_VERTICAL within a procedure, screen updating occurs as follows:

- When you execute a built-in that modifies the buffer or the current character position before you issue the call to CURSOR_VERTICAL, the screen is updated before CURSOR_VERTICAL is executed. This action ensures that the vertical movement of the cursor starts at the correct character position.
- Otherwise, the screen manager does not update the screen until the procedure has finished executing and control is returned to the screen manager.

CURSOR_VERTICAL has no effect if you use an input device other than a video terminal supported by VAXTPU.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	CURSOR_VERTICAL requires at least one parameter.
TPU$_TOOMANY	ERROR	CURSOR_VERTICAL accepts at most one parameter.
TPU$_INVPARAM	ERROR	You did not specify an integer as the parameter.

EXAMPLES

1
```
int_y := CURSOR_VERTICAL (5)
```

This statement moves the cursor position five lines toward the bottom of the screen.

2
```
! Free cursor motion procedures
PROCEDURE user_free_cursor_up
   IF GET_INFO (CURRENT_WINDOW, "CURRENT_ROW") =
               GET_INFO (CURRENT_WINDOW, "VISIBLE_TOP")
   THEN
       SCROLL (CURRENT_WINDOW, -1);
   ELSE
       left_y := CURSOR_VERTICAL (-1);
   ENDIF;
ENDPROCEDURE;
```

```
PROCEDURE user_free_cursor_down
   IF GET_INFO (CURRENT_WINDOW, "CURRENT_ROW") =
               GET_INFO (CURRENT_WINDOW, "VISIBLE_BOTTOM")
   THEN
       SCROLL (CURRENT_WINDOW, 1);
   ELSE
       right_x := CURSOR_VERTICAL (1);
   ENDIF;
ENDPROCEDURE;
```

These procedures provide for free cursor motion up and down the screen. These procedures can be bound to keys (for example, the arrow keys) so that the movement can be accomplished with a single keystroke.

These examples work regardless of the setting of CROSS_WINDOW_BOUNDS, because the built-in procedure SCROLL keeps the cursor motion within a single window.

DEBUG_LINE

Returns the line number of the current breakpoint.

FORMAT

integer := DEBUG_LINE

PARAMETERS

None.

return value

An integer representing the line number of the current breakpoint.

DESCRIPTION

The DEBUG_LINE built-in procedure returns the line number of the current breakpoint. Use DEBUG_LINE when writing your own VAXTPU debugger.

Digital recommends that you use the debugger provided in SYS$SHARE:TPU$DEBUG.TPU.

SIGNALED ERROR

TPU$_NEEDTOASSIGN	ERROR	The DEBUG_LINE built-in must appear on the right-hand side of an assignment statement.

EXAMPLE

```
the_line := GET_INFO (DEBUG, "line_number");
IF the_line = 0
    THEN the_line := DEBUG_LINE;
ENDIF;
```

This code fragment first uses GET_INFO to request the line number of the breakpoint in the current procedure. If the line number is 0, meaning that the breakpoint is not in a procedure, the code uses DEBUG_LINE to determine the breakpoint's line number relative to the buffer.

DEFINE_KEY

Associates executable VAXTPU code with a key or a combination of keys.

FORMAT

DEFINE_KEY ({ *buffer* | *learn* | *program* | *range* | *string1* }, *key-name*
[[,*string2* [[,*string3*]]])

PARAMETERS

buffer
A buffer that contains the VAXTPU statements to be associated with a key.

learn
A learn sequence that specifies the executable code associated with a key.

program
A program that contains the executable code to be associated with a key.

range
A range that contains the VAXTPU statements to be associated with a key.

string1
A string that specifies the VAXTPU statements to be associated with a key.

key-name
A VAXTPU key name for a key or a combination of keys. See Table 2–1 for a list of the VAXTPU key names for the VT300, VT200, and VT100 series of keyboards. You can also display all the VAXTPU keywords with the built-in procedure SHOW (KEYWORDS).

See the Description section of this built-in procedure for information on keys that you cannot define.

To define a key for which there is no VAXTPU key name, use the built-in procedure KEY_NAME to create your own key name for the key. For example, KEY_NAME ("A", SHIFT_KEY) creates a key name for the combination of PF1, the default shift key for VAXTPU, and the keyboard character *A*. For more information, see the description of the built-in procedure KEY_NAME.

string2
An optional string associated with a key that you define. The string is treated as a comment that can be retrieved with the built-in procedure LOOKUP_KEY. You might want to use the comment if you are creating a help procedure for keys that you have defined.

string3

A key map or a key map list in which the key is to be defined. If a key map list is specified, the key is defined in the first key map in the key map list. If neither a key map nor a key map list is specified, the key is defined in the first key map in the key map list bound to the current buffer. See the descriptions of the built-in procedures CREATE_KEY_MAP, CREATE_KEY_MAP_LIST, and SET (KEY_MAP_LIST) for more information on key maps and key map lists.

DESCRIPTION

The built-in procedure DEFINE_KEY compiles the first parameter if it is a string, buffer, or range.

If you use DEFINE_KEY to change the definition of a key that was previously defined, VAXTPU does not save the previous definition.

You can define all the keys on the VT300, VT200, and VT100 keyboards and keypads with the following exceptions:

- The COMPOSE CHARACTER key on VT300 and VT200 keyboards
- The SHIFT keys

There are some keys that you can define but that Digital strongly recommends you avoid defining. VAXTPU does not signal an error when you use them as keyword parameters. However, in some cases the definitions you assign to these key combinations are not executed unless you set your terminal in special ways at the DCL level:

- CTRL/C, CTRL/O, CTRL/X, and F6 — To execute programs that you bind to these keys, you must first enter the DCL command SET TERMINAL/PASTHRU.
- CTRL/T, CTRL/Y — To execute programs that you bind to these keys, you must first enter the DCL command SET TERMINAL/PASTHRU and/or the DCL command SET NOCONTROL.
- CTRL/S, CTRL/Q — To execute programs that you bind to these keys, you must first enter the DCL command SET TERMINAL/NOTTSYNC.
- The PF1 key — This is the default shift key for the editor. You cannot define PF1 unless you use the built-in procedure SET (SHIFT_KEY, keyword) to define a different key as the shift key for the editor.
- The ESCAPE key
- The keys F1 through F5

Digital recommends that you do not use the special terminal settings mentioned above. The settings may cause unpredictable results if you do not understand all the implications of changing the default settings.

Whenever you extend EVE by writing a procedure that can be bound to a key, the procedure must return true and false as needed to indicate whether execution of the procedure completed successfully. EVE's REPEAT command relies on this return value to determine whether to halt repetition of a command, a procedure bound to a key, or a learn sequence.

SIGNALED ERRORS

TPU$_NOTDEFINABLE	WARNING	Second argument is not a valid reference to a key.
TPU$_RECURLEARN	WARNING	This key definition was used as a part of a learn sequence. You cannot use it in this context.
TPU$_NOKEYMAP	WARNING	Fourth argument is not a defined key map.
TPU$_NOKEYMAPLIST	WARNING	Fourth argument is not a defined key map list.
TPU$_KEYMAPNTFND	WARNING	The key map listed in the fourth argument is not found.
TPU$_EMPTYKMLIST	WARNING	The key map list specified in the fourth argument contains no key maps.
TPU$_TOOFEW	ERROR	Too few arguments passed to the DEFINE_KEY built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the DEFINE_KEY built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the DEFINE_KEY built-in.
TPU$_COMPILEFAIL	WARNING	Compilation aborted.
TPU$_UNKKEYWORD	ERROR	An unknown keyword has been used as an argument.
TPU$_BADKEY	ERROR	An unknown keyword has been used as an argument.
TPU$_KEYSUPERSEDED	INFORMATIONAL	Key definition superseded.

EXAMPLES

```
1  DEFINE_KEY ("POSITION (main_window)", CTRL_B_KEY)
```

This statement associates the VAXTPU statement POSITION (main_window) with the key combination CTRL/B. Note that you must use quotation marks around the VAXTPU statement.

2
```
DEFINE_KEY (main_buffer, KEY_NAME (PF4, SHIFT_KEY), "mainbuf")
```

This statement causes VAXTPU to compile the main buffer (containing VAXTPU statements). If there are no errors in the compilation, VAXTPU binds the executable code to the combination of the editor's shift key (PF1 by default) and PF4 on the keypad. The final string in the statement "mainbuf" is a comment that is associated with the key combination.

3
```
DEFINE_KEY ('COPY_TEXT ("Extendable")', KEY_NAME ("z", SHIFT_KEY))
```

This statement causes VAXTPU to make a copy of the word "Extendable" at the current character location in the current buffer when you press the key combination PF1 (VAXTPU's default shift key) and *z*. Notice that the inner set of quotation marks must be of a different kind from the outer set in the first parameter. Also notice that you must place quotation marks around the keyboard character that you use in combination with the editor's shift key.

4
```
PROCEDURE user_define_key
   def := READ_LINE ("Definition: ");
   key := READ_LINE ("Press key to define.",1);
   IF LENGTH (key) > 0
   THEN
       key := KEY_NAME (key)
   ELSE
       key := LAST_KEY;
   ENDIF;
   DEFINE_KEY (def,key);
ENDPROCEDURE;
```

This procedure prompts the user for the VAXTPU statements to be bound to the key that the user specifies.

5
```
PROCEDURE user_change_mode
! Toggle mode between insert and overstrike
   IF GET_INFO (CURRENT_BUFFER, "mode") = OVERSTRIKE
   THEN
       SET (INSERT, CURRENT_BUFFER);
   ELSE
       SET (OVERSTRIKE, CURRENT_BUFFER);
   ENDIF;
ENDPROCEDURE;

! The following statement binds this procedure to the
! key combination CTRL/A. This emulates the VMS key binding
! that toggles between insert and overstrike for text entry
! in command line editing.

DEFINE_KEY ("user_change_mode", CTRL_A_KEY);
```

This procedure changes the mode of text entry from insert to overstrike, or from overstrike to insert.

6
```
DEFINE_KEY ('MESSAGE ("Hello VAXTPU user")', CTRL_A_KEY, "Greeting", "TPU$KEY_MAP");
```

This example defines a key in a key map. The DEFINE_KEY statement defines CTRL/A in the key map TPU$KEY_MAP such that VAXTPU displays the message "Hello VAXTPU user" when CTRL/A is pressed.

```
7    DEFINE_KEY ("POSITION (MESSAGE_WINDOW)", F20,"", "movement_map")
```

This example uses a key map ("movement_map") but does not include a comment in the optional third parameter. Note the null string after the keyword F20 in the second parameter.

DEFINE_WIDGET_CLASS

Defines a widget class and optional creation routine for later use in creating widgets of that class using the DECwindows intrinsics or the XUI Toolkit low-level creation routines.

FORMAT

integer := *DEFINE_WIDGET_CLASS (class_name*
[, creation_routine_name
[, creation_routine_image_name]])

PARAMETERS

class_name

A string that is the name of a universal symbol pointing to the desired widget class record. A universal symbol is a symbol in a sharable image that can be referred to in an image other than the one in which the symbol is defined.

creation_routine_name

A string that is the name of the low-level widget creation routine for this widget class. Specify the case of the string correctly. To determine the correct case of the string, consult the documentation for the widget whose class you are defining. The current version of VAXTPU, which is bundled with the VMS operating system, ignores the case of the string. However, future versions of VAXTPU may treat the string as case sensitive.

If you do not specify this parameter, VAXTPU uses the X Toolkit CREATE WIDGET routine to create the widget instead of using a low-level widget creation routine. The routine must have the same calling sequence as the XUI Toolkit low-level widget creation routines.

In the current version of VAXTPU, you must specify the VMS binding of the creation routine name.

creation_routine_image_name

A string that is the name of the shareable image in which the class record can be found. If you specify a low-level creation routine, DEFINE_WIDGET_CLASS also looks for the routine in the program image. If you do not specify an image, VAXTPU assumes the widget is defined in SYS$LIBRARY:DECW$DWTLIBSHR.EXE.

return value

An integer used by the CREATE_WIDGET built-in to identify the class of widget to be created.

DESCRIPTION

Returns a class integer, which you use to specify the class of a widget when you create it.

Defining a class that is already defined returns the existing class integer. Defining a new class also defines the widget creation routine as the second parameter, if specified, or the X toolkit CREATE_WIDGET routine. VAXTPU searches for a new class record in the third parameter, if specified, or in SYS$LIBRARY:DECW$DWTLIBSHR.EXE.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by DEFINE_WIDGET_CLASS.
TPU$_NEEDTOASSIGN	ERROR	DEFINE_WIDGET_CLASS must return a value.
TPU$_TOOFEW	ERROR	Too few arguments passed to DEFINE_WIDGET_CLASS.
TPU$_TOOMANY	ERROR	Too many arguments passed to DEFINE_WIDGET_CLASS .
TPU$_REQUIRESDECW	ERROR	You can use DEFINE_WIDGET_CLASS only if you are using DECwindows VAXTPU.
TPU$_SYSERROR	ERROR	Could not find class record or creation routine in shareable image.

EXAMPLE

For a sample procedure using the DEFINE_WIDGET_CLASS built-in, see Example B–2.

DELETE

Removes VAXTPU structures from your editing context. When you delete a structure (for example, a range) all variables that refer to that structure are reset to unspecified. If the deleted structure had any associated resources, these resources are returned to the editor. When a buffer is deleted, the associated journal file (if any) is closed and deleted.

FORMAT

DELETE ({ *array* | *buffer* | *integer* | *keyword* | *learn* | *marker* | *pattern* | *process* | *program* | *range* | *string* | *unspecified* | *widget* | *window* })

PARAMETERS

array

The array you want to delete. The memory used by the array is freed for later use. If some other data structure, such as a pattern, is referenced only in the array, then that data structure is deleted when the array is deleted.

buffer

The buffer you want to delete. Any ranges or markers that point to this buffer, any subprocess that is associated with this buffer, the memory for the buffer control structure, the pages for storing text, and the memory for ranges and markers associated with the buffer are deleted also. If the buffer is associated with a window that is mapped to the screen, the window is unmapped. Any associated buffer change journal file is also closed and deleted.

integer

The integer to delete. Integers use no internal structures or resources so deleting a variable of type integer simply changes that variable to type unspecified.

keyword

The keyword to delete. Keywords use no internal structures or resources so deleting a variable of type keyword simply assigns to that variable the type unspecified.

learn
The learn sequence you wish to delete. The memory used by the learn sequence is freed for later use.

marker
The marker you want to delete. The memory for the marker control structure is deleted also.

pattern
The pattern you wish to delete. The memory used by the pattern is freed for later use. If the pattern includes a reference to another pattern and there are no other references to that pattern, then that pattern is deleted as well.

process
The process you want to delete. The memory for the process control structure and the subprocess is deleted also.

program
The program you want to delete. The memory for the program control structure and the memory for the program code are deleted also.

range
The range that you want to delete. The memory for the range control structure is deleted also. The text in a range does not belong to the range. Rather, it belongs to the buffer in which it is located. A range is merely a way of manipulating sections of text within a buffer. When you delete a range, the text delimited by the range is not deleted. See the built-in procedure ERASE for a description of how to remove the text in a range.

string
The string you wish to delete. The memory used by the string is freed for later use.

unspecified
Deleting a variable of type unspecified is allowed but does nothing.

widget
The widget to be deleted. When you use the DELETE (widget) built-in, all variables and array elements that refer to the widget are set to unspecified. If an array element is indexed by the deleted widget, the array element is deleted as well.

window
The window you want to delete. Along with the window, the memory for the window control structure and the record history associated with the window are deleted. If you delete a window that is mapped to the screen, VAXTPU unmaps the window before deleting it. The screen appears just as it does when you use the built-in procedure UNMAP.

DESCRIPTION

Depending upon how many variables are referencing an entity, or how many other entities are associated with the entity you are deleting, processing the built-in procedure DELETE can be time consuming. DELETE cannot be terminated by a CTRL/C.

Any variables that reference the deleted entity are set to unspecified and all other entities that are associated with the deleted entity are also deleted. Use the built-in procedure DELETE with caution.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	DELETE requires one argument.
TPU$_TOOMANY	ERROR	DELETE accepts only one argument.
TPU$_BADDELETE	ERROR	You attempted to delete a constant.
TPU$_DELETEFAIL	WARNING	DELETE could not delete the process.
TPU$_INVBUFDELETE	WARNING	You cannot delete a permanent buffer.

EXAMPLES

1
```
DELETE (main_buffer)
```

This statement deletes the main buffer and any associated resources that VAXTPU allocated for the main buffer. As a result of this command, the SHOW (BUFFERS) command does not list MAIN_BUFFER as one of the buffers in your editing context.

2
```
PROCEDURE user_delete_extra

   WRITE_FILE (extra_buf);
   DELETE (extra_window);
   DELETE (extra_buf);

! Return the 11 lines from extra_window to the main window
   ADJUST_WINDOW (main_window, -11, 0);

ENDPROCEDURE;
```

This procedure writes the contents of EXTRA_BUF to a file (because you do not specify a file name, the associated file for the buffer is used) and then removes the extra window and buffer from your editing context. You must have previously created these structures and added them to your editing context in order for this procedure to execute successfully.

```
PROCEDURE sample_create_and_delete

LOCAL example_widget,
      example_widget_name,
      example_hierarchy;

example_hierarchy := SET (DRM_HIERARCHY, "mynode$dua0:[smith]example.uid");
example_widget_name := "EXAMPLE_BOX";
example_widget := CREATE_WIDGET (example_widget_name,
                                 example_hierarchy, SCREEN,
                                 "user_callback_dispatch_routine");
!     .
!     .
!     .

DELETE (example_widget);

ENDPROCEDURE;
```

This code fragment creates a modal dialog box widget and later deletes it. For purposes of this example, the procedure *user_callback_dispatch_routine* is assumed to be a user-written procedure that handles widget callbacks. For a sample DECwindows User Interface Language (UIL) file to be used with VAXTPU code creating a modal dialog box widget, see the example in the description of the CREATE_WIDGET built-in.

EDIT

Modifies a string according to the keywords you specify. EDIT is similar although not identical to the DCL lexical function F$EDIT. Differences between the built-in procedure and the lexical function are noted in the Description section.

FORMAT

{ **buffer1** / **range1** / **string1** } := **EDIT** ({ *buffer2* / *range2* / *string2* }, *keyword1*[,...] [,*keyword2*] [,*keyword3*])

PARAMETERS

buffer2

The buffer in which you want VAXTPU to edit text. Note that you cannot use the keyword NOT_IN_PLACE if you specify a buffer for the first parameter.

range2

The range in which you want VAXTPU to edit text. Note that you cannot use the keyword NOT_IN_PLACE if you specify a range for the first parameter.

string2

The string you want to modify. If you specify a return value, the returned string consists of the string you specify for the first parameter, modified in the way you specify in the second and subsequent parameters. If you specify IN_PLACE for the third parameter, EDIT makes the specified change to the string specified in the first parameter. Note that if *string2* is a constant, IN_PLACE has no effect.

keyword1

A keyword specifying the editing operation you want to perform on the string. Valid keywords and their meaning are as follows:

Keyword	Meaning
COLLAPSE	Removes all spaces and tabs.
COMPRESS	Replaces multiple spaces and tabs with a single space.
TRIM	Removes leading and trailing spaces and tabs.
TRIM_LEADING	Removes leading spaces and tabs.
TRIM_TRAILING	Removes trailing spaces and tabs.
LOWER	Converts all uppercase characters to lowercase.
UPPER	Converts all lowercase characters to uppercase.
INVERT	Changes the current case of the specified characters; uppercase characters become lowercase and lowercase characters become uppercase.

keyword2

A keyword specifying whether VAXTPU quote characters are used as quote characters or as regular text. The valid keywords are ON or OFF. The default is ON.

keyword3

A keyword indicating where VAXTPU is to make the indicated change. The valid keywords and their meanings are as follows:

Keyword	Meaning
IN_PLACE	Makes the indicated change in place. This is the default.
NOT_IN_PLACE	Leaves the specified string unchanged and returns a string that is the result of the specified editing. You cannot use NOT_IN_PLACE if the first parameter is specified as a range or buffer. To use NOT_IN_PLACE, you must specify a return value for EDIT.

Note that this keyword is ignored if *string2* is a string constant. EDIT never edits string constants in place. It does return the edited string.

return values

buffer1

A variable of type buffer pointing to the buffer containing the modified text, if you specify a buffer for the first parameter. The variable "returned_buffer" points to the same buffer pointed to by the buffer variable specified as the first parameter.

range1

A range containing the modified text, if you specify a range for the first parameter. The returned range spans the same text as the range specified as a parameter, but they are two separate ranges. If you subsequently change or delete one of the ranges, this has no effect on the other range.

string1

A string containing the modified text, when you specify a string for the first parameter. EDIT can return a string even if you specify IN_PLACE.

DESCRIPTION

VAXTPU modifies the first parameter of the EDIT built-in in place. EDIT does not modify a literal string.

By default, EDIT does not modify quoted text that occurs within a string. For example, the following code does not change the case of WELL:

```
string_to_change := 'HE SANG "WELL"';
edit (string_to_change, LOWER);
```

The variable *string_to_change* has the value *he sang "WELL"*.

If you specify more than one of the TRIM keywords (TRIM, TRIM_LEADING, TRIM_TRAILING), all of the TRIM operations you specify are performed.

If you specify more than one of the case conversion keywords (UPPER, LOWER, INVERT), the last keyword that you specify determines how the characters in the string are modified.

If you specify both of the quote recognition keywords (ON, OFF), the last keyword you specify determines whether or not EDIT modifies quoted text.

If you specify no keywords, EDIT does nothing to the passed string.

You can disable the recognition of quotation marks and apostrophes as VAXTPU quote characters by using the keyword OFF as a parameter for EDIT. When you use the keyword OFF, VAXTPU preserves any quotation marks and apostrophes in the edited text and performs the editing tasks you specify on the text within the quotation marks and apostrophes. OFF may appear anywhere in the keyword list. It need not be the final parameter.

If the string you specify has opening quotation marks but not closing quotation marks, the status TPU$_MISSINGQUOTE is returned. All text starting at the unclosed opening quotation mark and continuing to the end of the string is considered to be part of the quoted string and is not modified.

EDIT is similar to the DCL lexical function F$EDIT. However, you should note the following differences:

- EDIT modifies the characters in place while F$EDIT returns a result.
- EDIT takes keywords as parameters while F$EDIT requires that the edit commands be specified by a string.

SIGNALED ERRORS

TPU$_MISSINGQUOTE	ERROR	Character string is missing terminating quotation marks.
TPU$_TOOFEW	ERROR	EDIT requires at least one parameter.
TPU$_TOOMANY	ERROR	You supplied keywords that are duplicative or contradictory.
TPU$_ARGMISMATCH	ERROR	One of the parameters to EDIT is of the wrong data type.
TPU$_INVPARAM	ERROR	One of the parameters to EDIT is of the wrong data type.
TPU$_BADKEY	WARNING	You gave the wrong keyword to EDIT.

EXAMPLES

1
```
pn := "PRODUCT NAME";
EDIT (pn, LOWER);
MESSAGE (pn);
```

These statements edit the string "PRODUCT NAME" by changing it to lowercase, and display the edited string in the message window.

2
```
PROCEDURE user_edit_string (input_string)
   is := input_string;
   EDIT (is, LOWER);
   MESSAGE (is);
ENDPROCEDURE;
```

This procedure shows a generalized way of changing any input string to lowercase.

After compiling the preceding procedure, you can direct VAXTPU to print the lowercase word "zephyr" in the message area by entering the following command:

```
user_edit_string ("ZEPHYR")
```

3
```
returned_value := EDIT (the_string, COLLAPSE, OFF, NOT_IN_PLACE);
```

This statement removes all spaces and tabs from the string pointed to by *the_string* and does not treat quotation marks or apostrophes as quote characters. Returns the modified string in the variable *returned_value*, but does not change the string in the variable *the_string*.

END_OF

Returns a marker that points to the last character position in a buffer or a range.

FORMAT

marker := END_OF ({ *buffer* / *range* })

PARAMETERS

buffer

The buffer whose last character position you want to mark.

range

The range whose last character position you want to mark.

return value

A marker pointing to the last character position in a buffer or range.

DESCRIPTION

If you use the marker returned by the END_OF built-in as a parameter for the built-in procedure POSITION, the editing point moves to this marker.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	END_OF must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	END_OF requires one argument.
TPU$_TOOMANY	ERROR	END_OF accepts only one argument.
TPU$_ARGMISMATCH	ERROR	You passed something other than a range or a buffer to END_OF.

EXAMPLES

1
```
the_end := END_OF (CURRENT_BUFFER)
```

This assignment statement stores the last position in the current buffer in the variable *the_end*.

2
```
POSITION (END_OF (delete_range))
```

This statement uses two built-in procedures to move your current character position to the end of *delete_range*. If *delete_range* is in a visible buffer in which the cursor is located, the cursor position also moves to the end of *delete_range*.

```
3   PROCEDURE user_paste
       LOCAL paste_text;
       IF (BEGINNING_OF (paste_buffer) <> END_OF (paste_buffer))
       THEN
           COPY_TEXT (paste_buffer);
       ENDIF;
    ENDPROCEDURE;
```

This procedure implements a simple INSERT HERE function. The variable *paste_buffer* points to a buffer that holds previously cut text.

ERASE

Removes the contents of the range or buffer that you specify.

FORMAT

ERASE ({ *buffer* / *range* })

PARAMETERS

buffer

The buffer whose contents you want to remove.

range

The range whose contents you want to remove.

DESCRIPTION

When you erase a buffer, the contents of the buffer are removed. However, the buffer structure still remains a part of your editing context and the editing point remains in the buffer even if you remove the contents of the buffer. The space that was occupied by the contents of the buffer is returned to the system and is available for reuse. Only the end-of-buffer line remains.

When you erase a range, the contents of the range are removed from the buffer. The range structure is still a part of your editing context. You can use the range structure later in your editing session to delimit an area of text within a buffer.

Note that text does not belong to a range; it belongs to a buffer. Ranges are merely a way of manipulating portions of text within a buffer. For more information on ranges, see Chapter 2.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	ERASE requires one argument.
TPU$_TOOMANY	ERROR	ERASE accepts only one argument.
TPU$_INVPARAM	ERROR	The argument to ERASE is of the wrong type.
TPU$_NOTMODIFIABLE	WARNING	You cannot erase text in an unmodifiable buffer.

EXAMPLES

1

```
ERASE (main_buffer)
```

This statement erases all the text in the buffer referenced by *main_buffer*. Since the buffer still exists, you can select the buffer using the POSITION built-in or map the buffer to a window. The procedure simply removes all text from the buffer. All markers in the buffer now mark the end of the buffer.

2

```
PROCEDURE user_remove_crlfs

   LOCAL crlf,
         here,
         cr_range;

   crlf := ASCII (13) + ASCII (10);
   here := MARK (NONE);
   POSITION (BEGINNING_OF (CURRENT_BUFFER));

   LOOP
      cr_range := SEARCH_QUIETLY (crlf, FORWARD, EXACT);
      EXITIF cr_range = 0;
      ERASE (cr_range);
      POSITION (cr_range);
   ENDLOOP;

   POSITION (here);
ENDPROCEDURE;
```

This procedure gets rid of embedded carriage-return/line-feed pairs.

ERASE_CHARACTER

Deletes the number of characters you specify and optionally returns a string that represents the characters you deleted.

FORMAT

[string :=] ERASE_CHARACTER *(integer)*

PARAMETER

integer
An expression that evaluates to an integer, which may be signed. The value indicates which characters, and how many of them, are to be erased.

return value

A string representing the characters deleted by ERASE_CHARACTER.

DESCRIPTION

ERASE_CHARACTER deletes up to the specified number of characters from the current line. If the argument to ERASE_CHARACTER is a positive integer, ERASE_CHARACTER deletes that many characters, starting at the current position and continuing toward the end of the line. If the argument is negative, ERASE_CHARACTER deletes characters to the left of the current character. It uses the absolute value of the parameter to determine the number of characters to delete. ERASE_CHARACTER stops deleting characters if it reaches the beginning or the end of the line before deleting the specified number of characters.

Using ERASE_CHARACTER may cause VAXTPU to insert padding spaces or blank lines in the buffer. ERASE_CHARACTER causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

ERASE_CHARACTER optionally returns a string containing the characters that it deleted.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	ERASE_CHARACTER requires one argument.
TPU$_TOOMANY	ERROR	ERASE_CHARACTER accepts only one argument.
TPU$_INVPARAM	ERROR	The argument to ERASE_CHARACTER must be an integer.

TPU$_NOCURRENTBUF	WARNING	There is no current buffer to erase characters from.
TPU$_NOTMODIFIABLE	WARNING	You cannot modify an unmodifiable buffer.

EXAMPLES

1
```
take_out_chars := ERASE_CHARACTER (10)
```

This assignment statement removes the current character and the nine characters following it and copies them in the string variable *take_out_chars*. If there are only five characters following the current character, then this statement deletes only the current character and the five following it. It does not delete characters on the next line as well.

2
```
prev_chars := ERASE_CHARACTER (-5)
```

This assignment statement removes the five characters preceding the current character and copies them in the string variable *prev_chars*.

3
```
! This procedure deletes the character to the
! left of the current character.  If at the
! beginning of a line, it appends the current
! line to the previous line.

PROCEDURE user_delete_key

   LOCAL deleted_char;

   deleted_char := ERASE_CHARACTER (-1);

   IF deleted_char = ""     ! nothing deleted
   THEN
       APPEND_LINE;
   ENDIF;
ENDPROCEDURE;
```

This procedure deletes the character to the left of the editing point. If the editing point is at the beginning of a line, the procedure appends the current line to the previous line.

ERASE_LINE

Removes the current line from the current buffer.

ERASE_LINE optionally returns a string containing the text of the deleted line.

FORMAT

[string :=] ERASE_LINE

PARAMETERS

None.

return value

A string containing the text of the deleted line.

DESCRIPTION

ERASE_LINE deletes the current line, optionally storing the deleted text in a string before doing so. The current position moves to the first character of the line following the deleted line.

Using ERASE_LINE may cause VAXTPU to insert padding spaces or blank lines in the buffer. ERASE_LINE causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

If the screen manager inserts padding spaces, ERASE_LINE deletes these spaces when it deletes the line. The spaces appear in the returned string. If the screen manager inserts padding lines into the buffer, ERASE_LINE deletes only the last of these lines.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	ERASE_LINE accepts no arguments.
TPU$_NOTMODIFIABLE	WARNING	You cannot erase a line in an unmodifiable buffer.
TPU$_NOCURRENTBUF	ERROR	You must select a buffer before erasing a line.

EXAMPLES

1
```
ERASE_LINE
```

This statement removes the current line from the current buffer.

2
```
take_out_line := ERASE_LINE
```

This statement removes the current line from the current buffer and stores the string of characters representing that line in the variable *take_out_line*.

ERROR

Returns a keyword for the latest error.

FORMAT

keyword := ERROR

PARAMETERS

None.

return value

A keyword representing the most recent error.

DESCRIPTION

The possible error and warning codes for each built-in procedure are included in the description of each built-in procedure. Appendix C contains an alphabetized list of all the possible completion codes and severity levels in VAXTPU. The *VMS System Messages and Recovery Procedures Reference Manual* includes all the possible completion codes for VAXTPU as well as the appropriate explanations and suggested user actions.

The value returned by ERROR is only meaningful inside an error handler, after an error has occurred. The value outside an error handler is indeterminate.

Although ERROR behaves much like a built-in, it is actually a VAXTPU language element.

ERROR is evaluated for correct syntax at compile time. In contrast, VAXTPU procedures are usually evaluated for a correct parameter count and parameter types at execution.

SIGNALED ERROR

ERROR is a language element and has no completion codes.

EXAMPLE

```
PROCEDURE strip_blanks

! Remove trailing blanks from all the lines in a buffer

    LOCAL blank_chars,
          blank_pattern,
          blank_range;

    ON_ERROR
        IF ERROR = TPU$_STRNOTFOUND
        THEN
            RETURN;
        ELSE
            MESSAGE (ERROR_TEXT);
            ABORT;
        ENDIF;
    ENDON_ERROR;
```

```
    blank_chars := ASCII (32) + ASCII (9);
    blank_pattern := (SPAN (blank_chars) @ blank_range) + LINE_END;
    LOOP
       SEARCH (blank_pattern, FORWARD);
       POSITION (BEGINNING_OF (blank_range));
       ERASE (blank_range);
    ENDLOOP;
ENDPROCEDURE;
```

This procedure uses the ERROR language element to determine the error that invoked the error handler. If the error was that SEARCH could not find the specified string, then the procedure returns normally. (For more information on error handlers, see Chapter 3 and the descriptions of ABORT and RETURN in this chapter.) If the error was something else, then the text of the error message is written to the MESSAGES buffer and any executing procedures are terminated.

ERROR_LINE

Returns the line number for the latest error.

FORMAT

integer := ERROR_LINE

PARAMETERS

None.

return value

An integer representing the line number of the most recent error.

DESCRIPTION

ERROR_LINE returns the line number at which the error or warning occurs. If a procedure was compiled from a buffer or range, ERROR_LINE returns the line number within the buffer. This may be different from the line number within the procedure. If the procedure was compiled from a string, ERROR_LINE returns 1.

The value returned by ERROR_LINE is only meaningful inside an error handler, after an error has occurred. The value outside an error handler is indeterminate.

Although ERROR_LINE behaves much like a built-in, it is actually a VAXTPU language element.

ERROR_LINE is evaluated for correct syntax at compile time. In contrast, VAXTPU procedures are usually evaluated for a correct parameter count and parameter types at execution.

SIGNALED ERROR

ERROR is a language element and has no completion codes.

EXAMPLE

```
PROCEDURE strip_blanks

! Remove trailing blanks from all the lines in a buffer

   LOCAL blank_chars,
         blank_pattern,
         blank_range;

    ON_ERROR
        MESSAGE (ERROR_TEXT);
        MESSAGE ("Error on line " + STR (ERROR_LINE));
        RETURN;
    ENDON_ERROR;

    blank_chars := ASCII (32) + ASCII (9);
    blank_pattern := (SPAN (blank_chars) @ blank_range) + LINE_END;
```

```
    LOOP
       SEARCH (blank_pattern, FORWARD);
       POSITION (blank_range);
       ERASE (blank_range);
    ENDLOOP;
ENDPROCEDURE;
```

This procedure uses the ERROR_LINE built-in procedure to report the line in which the error occurred.

ERROR_TEXT

Returns the text of the latest error message.

FORMAT

string := ERROR_TEXT

PARAMETERS

None.

return value

A string containing the text of the most recent error message.

DESCRIPTION

ERROR_TEXT returns the text for the most recent error or warning.

The possible error and warning codes for each built-in procedure are included in the description of each built-in procedure. Appendix C contains an alphabetized list of all the possible completion codes and severity levels in VAXTPU. The *VMS System Messages and Recovery Procedures Reference Manual* includes all the possible completion codes for VAXTPU as well as the appropriate explanations and suggested user actions.

The value returned by ERROR_TEXT is meaningful only inside an error handler, after an error has occurred. The value outside an error handler is indeterminate.

Although ERROR_TEXT behaves much like a built-in, it is actually a VAXTPU language element.

ERROR_TEXT is evaluated for correct syntax at compile time. In contrast, VAXTPU procedures are usually evaluated for a correct parameter count and parameter types at execution.

SIGNALED ERROR

ERROR_TEXT is a language element and has no completion codes.

EXAMPLE

```
PROCEDURE strip_blanks

! Remove trailing blanks from all the lines in a buffer

   LOCAL blank_chars,
         blank_pattern,
         blank_range;

   ON_ERROR
       MESSAGE (ERROR_TEXT);
       MESSAGE ("Error on line " + STR (ERROR_LINE));
       RETURN;
   ENDON_ERROR;

   blank_chars := ASCII (32) + ASCII (9);
   blank_pattern := (SPAN (blank_chars) @ blank_range) + LINE_END;
```

```
   LOOP
      SEARCH (blank_pattern, FORWARD);
      POSITION (BEGINNING_OF (blank_range));
      ERASE (blank_range);
   ENDLOOP;
ENDPROCEDURE;
```

This procedure uses the built-in procedure ERROR_TEXT to report what happened and where.

EXECUTE

Does one of the following:

- Executes programs that you have previously compiled
- Compiles and then executes any executable statements in a buffer, a range, or a string
- Replays a learn sequence
- Executes a program bound to a key

FORMAT

EXECUTE ({ *buffer* | *key-name* [[, *key-map-list-name*]] [[, *key-map-name*]] | *learn* | *program* | *range* | *string* })

PARAMETERS

buffer

The buffer that you want to execute.

key-name

The VAXTPU key name for a key or a combination of keys. VAXTPU locates and executes the definition bound to the key.

key-map-list-name

The name of the key map list in which the key is defined. This optional parameter is only valid when the first parameter is a key name. If you specify a key map list as the second parameter, VAXTPU uses the first definition of the key specified by *key_name* found in any of the key maps specified by the key map list. If you do not specify any value for the second parameter, VAXTPU uses the first definition of the key specified by *key_name* found in the key map list bound to the current buffer.

key-map-name

The name of the key map in which the key is defined. This optional parameter is valid only when the first parameter is a key name. Use this parameter only if the key specified by the first parameter is defined in the key map specified as the second parameter. If you do not specify any value for the second parameter, VAXTPU uses the first definition of the key specified by *key_name* found in the key map list bound to the current buffer.

learn

The learn sequence that you want to replay.

program

The program that you want to execute.

range
The range that you want to execute.

string
The string that you want to execute.

DESCRIPTION

EXECUTE performs different actions depending upon the data type of the parameter.

If the parameter is a string or the contents of a buffer or range, it must contain only valid VAXTPU statements. Otherwise, you get an error message and no action is taken. See the description of the built-in procedure COMPILE for restrictions and other information on compiling strings or the contents of a buffer or range. When you pass a string to EXECUTE, the string cannot be longer than 256 characters.

Procedures are usually executed by entering the name of a compiled procedure at the appropriate prompt from your editing interface, or by calling the procedure from within another procedure. However, it is possible to execute procedures with the built-in procedure EXECUTE if the procedure returns a data type that is a valid parameter.

In the following example, the procedure *test* returns a program data type. If you execute a buffer or range that contains the following code, VAXTPU compiles and executes the procedure *test*, a program data type is returned, the program is then used as the parameter for the built-in procedure EXECUTE, and the string "abc" is written to the message area.

```
PROCEDURE test

! After compiling the string 'MESSAGE ("abc")',
! VAXTPU returns a program that is the compiled
! form of the string.

   RETURN COMPILE ('MESSAGE ("abc")');
ENDPROCEDURE;

! The built-in procedure EXECUTE executes the
! program returned by the procedure "test."

EXECUTE (test);
```

SIGNALED ERRORS

TPU$_NODEFINITION	WARNING	There is no definition for this key.
TPU$_REPLAYWARNING	WARNING	Inconsistency during the execution of a learn sequence . . . sequence is proceeding.
TPU$_REPLAYFAIL	WARNING	Inconsistency during the execution of a learn sequence . . . execution stopped.
TPU$_RECURLEARN	ERROR	You cannot execute learn sequences recursively.

TPU$_CONTROLC	ERROR	The execution of the command terminated because you pressed CTRL/C.
TPU$_EXECUTEFAIL	WARNING	Execution of the indicated item has halted because it contains an error.
TPU$_COMPILEFAIL	WARNING	Compilation aborted because of syntax errors.
TPU$_ARGMISMATCH	ERROR	A parameter's data type is unsupported.
TPU$_TOOFEW	ERROR	Too few arguments.
TPU$_TOOMANY	ERROR	Too many arguments.
TPU$_NOTDEFINABLE	WARNING	Key cannot be defined.
TPU$_NOCURRENTBUF	WARNING	Key map or key map list not specified, and there is no current buffer.
TPU$_NOKEYMAP	WARNING	Key map or key map list not defined.
TPU$_NOTMODIFIABLE	WARNING	You cannot copy text into an unmodifiable buffer.
TPU$_NODEFINITION	WARNING	Key not defined.

EXAMPLES

1
```
EXECUTE (user_program)
```

This statement executes the executable statements in the program named *user_program*.

2
```
EXECUTE (main_buffer)
```

This statement first compiles the contents of *main_buffer* and then executes any executable statements. If you have any text in the main buffer other than VAXTPU statements, you get an error message. If there are procedure definitions in *main_buffer*, they are compiled, but they are not executed until you run the procedure (either by entering the procedure name after the appropriate prompt from your interface or by calling the procedure from within another procedure).

3
```
EXECUTE (RET_KEY, "TPU$KEY_MAP_LIST");
```

This statement first finds the program bound to the return key in the default VAXTPU key map list, and then executes the code or learn sequence found.

4
```
PROCEDURE user_do
   command_string := READ_LINE ("Enter VAXTPU command to execute: ");
   EXECUTE (command_string);
ENDPROCEDURE;
```

This procedure prompts the user for a VAXTPU command to execute and then executes the command.

5
```
PROCEDURE user_tpu (TPU_COMMAND)
   SET (INFORMATIONAL, ON);
   EXECUTE (TPU_COMMAND);
   SET (INFORMATIONAL, OFF);
ENDPROCEDURE;
```

This procedure executes a command with informational messages turned on, and then turns the informational messages off after the command is executed. You must replace the parameter *TPU_COMMAND* with the desired VAXTPU statement.

EXIT

Terminates the editing session and writes out any modified buffers that have associated files. VAXTPU queries you for a file name for any buffer that you have modified that does not already have an associated file.

Buffers that have the NO_WRITE attribute are not written out. See SET (NO_WRITE, buffer).

FORMAT

EXIT

PARAMETERS

None.

DESCRIPTION

If you do not modify a buffer, VAXTPU does not write out the next version of the file associated with the buffer when you use the built-in procedure EXIT to exit from VAXTPU.

If you modify a buffer that does not have an associated file name, (because you did not specify a file name for the second parameter of CREATE_BUFFER), VAXTPU asks you to specify a file name if you want to write the buffer. If you press the RETURN key rather than entering a file name, the modified buffer is discarded. VAXTPU queries you about all modified buffers that do not have associated file names. The order of the query is the order in which the buffers were created.

Journal files (if any) are deleted upon exiting.

If an error occurs while you are trying to exit, the exit halts and control returns to the editor.

SIGNALED ERRORS

TPU$_EXITFAIL	WARNING	The EXIT did not complete successfully because of problems writing modified buffers.
TPU$_TOOMANY	ERROR	EXIT takes no arguments.

EXAMPLE

```
EXIT
```

This ends the editing session and writes out any modified buffers that have associated file names. If you have modified a buffer that does not have an associated file name, VAXTPU queries you with the following prompt:

```
Enter a file name to write buffer "buffer_name"; else press RETURN:
```

Enter a file name such as TEXT_FILE.LIS if you want the contents of the buffer written to a file. Press the RETURN key if you do not want to write the contents of the buffer to a file.

EXPAND_NAME

Returns a string that contains the names of any VAXTPU global variables, keywords, or procedures (built-in or user-written) that begin with the string that you specify. VAXTPU searches its internal symbol tables to find a match, using your input string as the directive for the match.

FORMAT

string2 := EXPAND_NAME *(string1* { *, ALL* | *, KEYWORDS* | *, PROCEDURES* | *, VARIABLES* } *)*

PARAMETERS

string1

An expression that evaluates to a string. If the string contains one or more asterisks (*) or percent signs (%), then the string is a wildcard specification of the VAXTPU names to match. An asterisk matches zero or more characters and a percent sign matches exactly one character. If the string does not contain any asterisks or percent signs, then the string is the initial substring of a VAXTPU name.

ALL

A keyword specifying that you want VAXTPU to match all names.

KEYWORDS

A keyword specifying that you want VAXTPU to match only keyword names.

PROCEDURES

A keyword specifying that you want VAXTPU to match only procedure names.

VARIABLES

A keyword specifying that you want VAXTPU to match only global variable names. EXPAND_NAME does not expand the names of local variables.

DESCRIPTION

If there are no matches for the substring you specify, a null string is returned and a warning (TPU$_NONAMES) is signaled. If only one VAXTPU name matches the substring you specify, the name is returned with no trailing space. If more than one VAXTPU name matches your substring, all of the matching names are returned. The matching names are returned as a concatenated string with words separated by a single space. Multiple names signal a warning (TPU$_MULTIPLENAMES).

Use EXPAND_NAME in procedures that perform command completion or that interpret abbreviated names.

EXPAND_NAME does not expand the names of local variables.

SIGNALED ERRORS

TPU$_NONAMES	WARNING	No names were found matching the one requested.
TPU$_MULTIPLENAMES	WARNING	More than one name matching the one requested was found.
TPU$_NEEDTOASSIGN	ERROR	EXPAND_NAME must appear on the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	EXPAND_NAME requires two arguments.
TPU$_TOOMANY	ERROR	EXPAND_NAME accepts no more than two arguments.
TPU$_INVPARAM	ERROR	One of the arguments you passed to EXPAND_NAME has the wrong data type.
TPU$_BADKEY	WARNING	You specified an invalid keyword as the second argument.

EXAMPLES

1
```
full_name := EXPAND_NAME ("MOVE", ALL)
```

This assignment statement returns the following VAXTPU names in the string *full_name*:

```
MOVE_HORIZONTAL MOVE_VERTICAL MOVE_TEXT
```

2
```
full_name := EXPAND_NAME ("*EXACT", KEYWORDS)
```

This assignment statement returns the following VAXTPU keyword names in the string *full_name*:

```
EXACT NO_EXACT
```

3
```
full_name := EXPAND_NAME ("%%", KEYWORDS)
```

This assignment statement returns the following VAXTPU keyword names in the string *full_name*:

```
ON UP DO E5 F6 E4 F7 E6 E1 E3 E2 F8 F9
```

These are all the keywords whose names are two characters long.

4
```
PROCEDURE user_quick_parse (abbreviated_name)

   ON_ERROR
       IF ERROR = TPU$_NONAMES
       THEN
           MESSAGE ("No such procedure.");
       ELSE
          IF ERROR = TPU$_MULTIPLENAMES
          THEN
              MESSAGE ("Ambiguous abbreviation.");
          ENDIF;
       ENDIF;
       RETURN;
   ENDON_ERROR;
```

```
   expanded_name := EXPAND_NAME (abbreviated_name, PROCEDURES);
   MESSAGE ("The procedure is " + expanded_name + ".");
ENDPROCEDURE;
```

This procedure uses the string that you enter as the parameter, and puts the expanded form of a valid VAXTPU procedure name that matches the string in the message area. If the initial string matches multiple procedure names, or if it is not a valid VAXTPU procedure name, an explanatory message is written to the message area.

FAO

Invokes the Formatted ASCII Output ($FAO) system service to convert a control string to a formatted ASCII output string. By specifying arguments for FAO directives in the control string, you can control the processing performed by the $FAO system service. The built-in procedure FAO returns a string that contains the formatted ASCII output.

For complete information on the $FAO system service, see the *VMS System Services Reference Manual.*

FORMAT

string2 := FAO *(string1* [, { *integer1* / *string3* } [, ... { *integer_n* / *string_n* }]])

PARAMETERS

string1

A string, a variable name representing a string constant, or an expression that evaluates to a string, that consists of the fixed text of the output string and FAO directives.

Some FAO directives that you can use as part of the string are the following:

!AS	Inserts a string as is
!OL	Converts a longword to octal notation
!XL	Converts a longword to hexadecimal notation
!ZL	Converts a longword to decimal notation
!UL	Converts a longword to decimal notation without adjusting for negative number
!SL	Converts a longword to decimal notation with negative numbers converted properly
!/	Inserts a new line (carriage return/line feed)
!_	Inserts a tab
!}	Inserts a form feed
!!	Inserts an exclamation mark
!%S	Inserts an *s* if the most recently converted number is not 1
!%T	Inserts the current time if you enter 0 as the parameter (you cannot pass a specific time because VAXTPU does not use quadwords)
!%D	Inserts the current date and time if you enter 0 as the parameter (you cannot pass a specific date because VAXTPU does not use quadwords)

integer1 ... integer_n

An expression that evaluates to an integer. $FAO uses these integers as arguments to the FAO directives in *string2* to form *string1*.

string3 ... string_n

An expression that evaluates to a string. $FAO uses these strings as arguments to the FAO directives in *string2* to form *string1*.

return value

A string containing the output you specify in ASCII format.

DESCRIPTION

FAO returns a formatted string, constructed according to the rules of the $FAO system service. The control string directs the formatting process, and the optional arguments are values to be substituted into the control string.

To ensure that you get meaningful results, you should use the !AS directive for strings and the !OL, !XL, !ZL, !UL, or !SL directive for integers.

SIGNALED ERRORS

TPU$_INVFAOPARAM	WARNING	Argument was not a string or an integer.
TPU$_NEEDTOASSIGN	ERROR	FAO must appear on the right-hand side of an assignment statement.
TPU$_INVPARAM	ERROR	The first argument to FAO must be a string.
TPU$_TOOFEW	ERROR	FAO requires at least one parameter.

EXAMPLES

1
```
date_and_time := FAO ("!%D",0)
```

This assignment statement stores the current date and time in the variable *date_and_time*.

2
```
PROCEDURE user_fao_conversion (count)

   report := FAO ("number of forms = !SL", count);
   MESSAGE (report);
ENDPROCEDURE;
```

This procedure uses the FAO directive !SL in a control string to convert the number equated to the variable *count* to a string. The converted string is stored in the variable *report* and then written to the message area.

3
```
PROCEDURE user_error_message (strng, line, col)

   error_count := error_count + 1;
   MESSAGE (FAO ("!AS at line !UL column !UL", strng, line, col));
ENDPROCEDURE;
```

This procedure formats the message that is being written to the message area. The message tells the user the line and column at which an error occurred.

FILE_PARSE

Performs the equivalent of the DCL F$PARSE lexical function. That is, it calls the RMS service $PARSE to parse a file specification and to return either an expanded file specification or the file specification field that you request.

FILE_PARSE returns a string that contains the expanded file specification or the field you specify. If you do not provide a complete file specification, FILE_PARSE supplies defaults in the return string, as described in the Description section.

If an error occurs during the parse, FILE_PARSE returns a null string.

FORMAT

string3 := FILE_PARSE *(filespec [[, string1*
[[, string2 [[, NODE]]
[[, DEVICE]]
[[, DIRECTORY]]
[[, NAME]] [[, TYPE]]
[[, VERSION]]]]])

PARAMETERS

filespec
The file specification to be parsed.

string1
A default file specification. Any field of the file specification that you provide with this parameter is substituted in the output string if that field is missing in the *filespec*.

string2
A related file specification. Some of the fields in the related file specification are substituted in the output string if a field is missing from both the *filespec* and the *string1* parameters.

NODE
Keyword specifying that FILE_PARSE should return a file specification including the node. For more information on using the optional keyword parameters to FILE_PARSE, see the Description section.

DEVICE
Keyword specifying that FILE_PARSE should return a file specification including the device. For more information on using the optional keyword parameters to FILE_PARSE, see the Description section.

DIRECTORY
Keyword specifying that FILE_PARSE should return a file specification including the directory. For more information on using the optional keyword parameters to FILE_PARSE, see the Description section.

NAME

Keyword specifying that FILE_PARSE should return a file specification including the name. For more information on using the optional keyword parameters to FILE_PARSE, see the Description section.

TYPE

Keyword specifying that FILE_PARSE should return a file specification including the type. For more information on using the optional keyword parameters to FILE_PARSE, see the Description section.

VERSION

Keyword specifying that FILE_PARSE should return a file specification including the version. For more information on using the optional keyword parameters to FILE_PARSE, see the Description section.

return value

A string containing an expanded file specification or the file specification field you specify.

DESCRIPTION

The built-in procedure FILE_PARSE allows you to parse file specifications using the RMS service $PARSE. For more information on the $PARSE service, see the *VMS Record Management Services Manual*.

If you do not supply any of the optional parameters, FILE_PARSE returns the device, directory, file name, and type of the file specified in *filespec*.

Specify the first three parameters as strings. The remaining parameters are keywords. Logical names and device names must terminate with a colon. If you omit optional parameters to the left of a given parameter, you must include null strings as place holders for the missing parameters.

You can specify as many of the keywords for field names as you wish. If one or more of these keywords are present, FILE_PARSE returns a string containing only those fields requested. The fields are returned in normal file specification order. The normal delimiters are included, but there are no other characters separating the fields. For example, suppose you direct VAXTPU to execute the following statements:

```
result := FILE_PARSE ("junk.txt","","",NODE, DEVICE, TYPE);
MESSAGE (result);
```

Suppose, too, that the node is WORK and the device is DISK1. When the statements execute, VAXTPU displays the following string:

```
work::disk1:.txt
```

If you omit the file name, type, or version number, FILE_PARSE supplies defaults, first from *string1* and then from *string2*. If you do not provide these parameters, FILE_PARSE returns a null specification for these fields.

The FILE_PARSE built-in procedure does not check that the file exists. It merely parses the file specification provided, and returns the portions of the resultant file specification requested.

You can use wildcard directives in supplying file specifications.

SIGNALED ERRORS

TPU$_PARSEFAIL	WARNING	RMS detected an error while parsing the file specification.
TPU$_NEEDTOASSIGN	ERROR	FILE_PARSE must appear on the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	FILE_PARSE requires at least one argument.
TPU$_INVPARAM	ERROR	One of the parameters to FILE_PARSE has the wrong data type.
TPU$_BADKEY	ERROR	You specified an invalid keyword to FILE_PARSE.

EXAMPLES

1
```
spec := FILE_PARSE ("program.pass", "[abbott]")
```

This assignment statement calls RMS to parse and return a full file specification for the file PROGRAM.PAS. The second parameter provides the name of the directory in which the file can be found.

2
```
PROCEDURE user_start_journal

! Default journal name
! Auxiliary journal name derived from file name

   LOCAL default_journal_name,
         aux_journal_name;

   IF (GET_INFO (COMMAND_LINE, "journal") = 1)
      AND
         (GET_INFO (COMMAND_LINE, "read_only") <> 1)
   THEN
       aux_journal_name := GET_INFO (CURRENT_BUFFER, "file_name");
         IF aux_journal_name = ""
         THEN
             aux_journal_name := GET_INFO (CURRENT_BUFFER, "output_file");
         ENDIF;
         IF aux_journal_name = 0
         THEN
             aux_journal_name := "";
         ENDIF;
         IF aux_journal_name = ""
         THEN
             default_journal_name := "user.TJL";
         ELSE
             default_journal_name := ".TJL";
         ENDIF;
         journal_file := GET_INFO (COMMAND_LINE, "journal_file");
         journal_file := FILE_PARSE (journal_file, default_journal_name,
                                     aux_journal_name);
         JOURNAL_OPEN (journal_file);
   ENDIF;
ENDPROCEDURE;
```

This procedure starts journaling. It is called from the TPU$INIT_PROCEDURE after a file is read into the current buffer. FILE_PARSE is used to return the full file specification for the journal file.

FILE_SEARCH

Calls the RMS service $SEARCH to search a directory and return the partial or full file specification for the file that you specify.

FILE_SEARCH returns a string containing the resulting file specification or a null string if no file is found.

FORMAT

string3 := FILE_SEARCH *(filespec*
[[, string1
[[, string2
[[, NODE]]
[[, DEVICE]]
[[, DIRECTORY]]
[[, NAME]] [[, TYPE]]
[[, VERSION]]]])

PARAMETERS

filespec

The file specification you want to find. If you omit the device or directory names, FILE_SEARCH supplies defaults from the optional parameters or from your current default device and directory if you do not supply optional parameters.

string1

A default file specification. If you fail to specify a field in *filespec* and that field is present in the default file specification, VAXTPU uses the field from *string1* when searching for the file.

string2

A related file specification. If you fail to specify a field in *filespec* and *string1* and that field is present in the related file specification, VAXTPU uses the field from *string2* when searching for the file.

NODE

Keyword specifying that FILE_SEARCH should return a file specification including the node. For more information on using the optional keyword parameters to FILE_SEARCH, see the Description section.

DEVICE

Keyword specifying that FILE_SEARCH should return a file specification including the device. For more information on using the optional keyword parameters to FILE_SEARCH, see the Description section.

DIRECTORY

Keyword specifying that FILE_SEARCH should return a file specification including the directory. For more information on using the optional keyword parameters to FILE_SEARCH, see the Description section.

NAME

Keyword specifying that FILE_SEARCH should return a file specification including the name. For more information on using the optional keyword parameters to FILE_SEARCH, see the Description section.

TYPE

Keyword specifying that FILE_SEARCH should return a file specification including the type. For more information on using the optional keyword parameters to FILE_SEARCH, see the Description section.

VERSION

Keyword specifying that FILE_SEARCH should return a file specification including the version. For more information on using the optional keyword parameters to FILE_SEARCH, see the Description section.

---

return value

A string containing the partial or full file specification you request from $SEARCH.

DESCRIPTION

The built-in procedure FILE_SEARCH allows you to search for files in a directory using the $SEARCH routine. You must use this built-in procedure multiple times with the same parameter to get all of the occurrences of a file name in a directory. See the *VMS Record Management Services Manual* for more information on $SEARCH.

Specify the first three parameters as strings. The remaining parameters are keywords. Logical names and device names must terminate with a colon. If you omit optional parameters to the left of a given parameter, you must include null strings as place holders for the missing parameters.

You can specify as many of the keyword parameters (such as NODE or DEVICE) as you wish. If one or more of these keywords are present, FILE_SEARCH returns only those fields requested in the keyword list, not the full file specification. The fields appear in the same order as they do in a full file specification. There is no separator between fields.

If you omit all the optional parameters, FILE_SEARCH returns the device, directory, file name, type, and version.

Unlike the FILE_PARSE built-in, FILE_SEARCH verifies that the file you specify exists.

If FILE_SEARCH does not find a matching file, or if the built-in finds one or more matches but is invoked again and does not find another match, FILE SEARCH returns a null string but not an error status. Thus, the null string can act as an "end of matching files" indicator. When FILE_SEARCH returns the status TPU$_SEARCHFAIL, look in the message buffer to see why the search was unsuccessful.

SIGNALED ERRORS

TPU$_SEARCHFAIL	WARNING	RMS detected an error while searching for the file.
TPU$_TOOFEW	ERROR	FILE_SEARCH requires at least one parameter.
TPU$_NEEDTOASSIGN	ERROR	FILE_SEARCH must be on the right-hand side of an assignment statement.
TPU$_INVPARAM	ERROR	One of the arguments you passed to FILE_SEARCH has the wrong type.
TPU$_BADKEY	WARNING	One of the keyword arguments you specified is not one of those FILE_SEARCH accepts.

EXAMPLES

1
```
fil := FILE_SEARCH ("SYS$SYSTEM:*.EXE")
```

Each time this assignment statement is executed, it returns a string containing the resulting file specification of an EXE file in SYS$SYSTEM. Because no version number is specified, only the latest version is returned. When you get a null string, it means there are no more EXE files in the directory.

2
```
PROCEDURE user_collect_rnos
   LOCAL filename;
   filename := FILE_SEARCH ("");
   LOOP
      filename := FILE_SEARCH ("*.RNO", "", "", NAME, TYPE);
      EXITIF filespec = "";
      CREATE_BUFFER (filename, filename);
   ENDLOOP;
ENDPROCEDURE;
```

This procedure is similar to the previous procedure. It makes use of the fact that you are looking for files in the current directory and that FILE_SEARCH can return parts of the file specification to eliminate the call to FILE_PARSE.

FILL

Reformats the text in the specified buffer or range so that the lines of text are approximately the same length.

FORMAT

FILL ({ *buffer* / *range* } [, *string* [, *integer1* [, *integer2* [, *integer3*]]]])

PARAMETERS

buffer
The buffer whose text you want to fill.

range
The range whose text you want to fill.

string
The list of additional word separators. The space character is always a word separator.

integer1
The value for the left margin. The left margin value must be at least 1 and must be less than the right margin value. Defaults to the buffer's left margin.

integer2
The value for the right margin. This value defaults to the same value as the buffer's right margin. *Integer2* must be greater than the left margin and cannot exceed the maximum record size for the buffer.

integer3
The value for the first line indent. This value modifies the left margin of the first filled line. It may be positive or negative. The result of adding the first line indent to the left margin must be greater than 1 and less than the right margin. Defaults to 0.

DESCRIPTION

FILL recognizes two classes of characters, text characters and word separators. Any character may be a word separator and any character other than the space character may be a text character. The space character is always a word separator, even if it is not present in the second parameter passed to FILL.

A word is a contiguous sequence of text characters, all of which are included on the same line, immediately preceded by a word separator or a line break, and immediately followed by a word separator or line break. If the first or last character in the specified range is a text character, that character marks the beginning or end of a word, regardless of any characters outside the range. Filling a range that starts or ends in the middle of a word may result in the insertion of a line break between that

part of the word inside the filled range and that part of the word outside the range.

When filling a range or buffer, FILL does the following to each line:

- Removes any spaces at the beginning of the line
- Sets the left margin of the line
- Moves text up to the previous line if it fits
- Deletes the line if it contains no text
- Splits the line if it is too long

FILL sets the line's left margin to the fill left margin unless that line is the first line of the buffer or range being filled. In this case, FILL sets the line's left margin to the fill left margin plus the first line indent. However, if you are filling a range and the range does not start at the beginning of a line, FILL does not change the left margin of that line.

FILL moves a word up to the previous line if the previous line is in the range to be filled and if the word fits on the previous line without extending beyond the fill right margin. Before moving the word up, FILL appends a space to the end of the previous line if that line ends in a space or a text character. It does not append a space if the previous line ends in a word separator other than the space character.

When moving a word up, FILL also moves up any word separators that follow the word, even if these word separators extend beyond the fill right margin. Fill does not move up any word separator that would cause the length of the previous line to exceed the buffer's maximum record size. If the previous line now ends in a space, FILL deletes that space. FILL does not delete more than one such space.

FILL moves any word separators at the beginning of a line up to the previous line. It does this even if the word separators will extend beyond the fill right margin.

FILL splits a line into two lines whenever the line contains two or more words and one of the words extends beyond the fill right margin. FILL splits the line at the first character of the first word that contains characters to the right of the fill right margin, unless that word starts at the beginning of the line. In this case, FILL does not split the line.

When operating on a range that does not begin at the first character of a line but does begin left of the fill left margin, FILL splits the line at the first character of the range.

FILL places the cursor at the end of the filled text after completing the tasks described above.

SIGNALED ERRORS

TPU$_INVRANGE	WARNING	You specified an invalid range enclosure.
TPU$_TOOFEW	ERROR	FILL requires at least one argument.
TPU$_TOOMANY	ERROR	FILL accepts no more than five arguments.
TPU$_ARGMISMATCH	ERROR	One of the parameters to FILL is of the wrong type.
TPU$_BADMARGINS	WARNING	You specified one of the fill margins incorrectly.
TPU$_INVPARAM	ERROR	One of the parameters to FILL is of the wrong type.
TPU$_NOTMODIFIABLE	WARNING	You cannot fill text in an unmodifiable buffer.
TPU$_NOCACHE	ERROR	FILL could not create a new line because there was no memory allocated for it.
TPU$_CONTROLC	ERROR	FILL terminated because you pressed CTRL/C.

EXAMPLES

1
```
FILL (current_buffer)
```

This statement fills the current buffer. It uses the buffer's left and right margins for the fill left and right margins. The space character is the only word separator. Upon completion, the current buffer contains no blank lines. All lines begin with a word. Unless the buffer contains a word too long to fit between the left and right margins, all text is between the buffer's left and right margins. Spaces may appear beyond the buffer's right margin.

2
```
FILL (paragraph_range, "-", 5, 65, 5)
```

If *paragraph_range* references a range that contains a paragraph, this statement fills a paragraph. FILL uses a left margin of 5 and a right margin of 65. It indents the first line of the paragraph an additional five characters. The space character and the hyphen are the two word separators. If the paragraph contains a hyphenated word, FILL breaks the word after the hyphen if necessary.

3
```
FILL (paragraph_range, "-", 10, 65, -3)
```

This example is like the previous one except that FILL unindents the first line of the paragraph by three characters. This is useful for filling numbered paragraphs.

GET_CLIPBOARD

Reads STRING format data from the clipboard and returns a string containing this data.

FORMAT

string := GET_CLIPBOARD

return value

A string consisting of the data read from the clipboard. Line breaks are indicated by a line-feed character (ASCII (10)).

DESCRIPTION

DECwindows provides a clipboard that allows you to move data between applications. Applications can write to the clipboard to replace previous data, and can read from the clipboard to get a copy of existing data. The data in the clipboard may be in multiple formats, but all the information in the clipboard must be written at the same time.

VAXTPU provides no clipboard support for applications not written for DECwindows.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	GET_CLIPBOARD must return a value.
TPU$_TOOMANY	ERROR	Too many arguments passed to GET_CLIPBOARD.
TPU$_CLIPBOARDFAIL	WARNING	The clipboard has not returned any data.
TPU$_CLIPBOARDLOCKED	WARNING	VAXTPU cannot read from the clipboard because some other application has locked it.
TPU$_CLIPBOARDNODATA	WARNING	There is no string format data in the clipboard.
TPU$_TRUNCATE	WARNING	Characters have been truncated because you tried to add text that would exceed the maximum line length.
TPU$_STRTOOLARGE	ERROR	The amount of data in the clipboard exceeds 65,535 characters.
TPU$_REQUIRESDECW	ERROR	You can use GET_CLIPBOARD only if you are using DECwindows VAXTPU.

EXAMPLE

```
new_string := GET_CLIPBOARD;
```

This statement reads what is currently in the clipboard and assigns it to *new_string*.

GET_DEFAULT

Returns the value of an X resource from the X resources database.

FORMAT

{ **string3** / **integer** } **:= GET_DEFAULT** *(string1, string2)*

PARAMETERS

string1

The name of the resource whose value you want GET_DEFAULT to fetch. Note that resource names are case sensitive.

string2

The class of the resource. Note that resource class names are case sensitive.

return value

The string equivalent of the resource value or 0 if the specified resource is not defined. Note that, if necessary, the application must convert the string to the data type appropriate to the resource.

DESCRIPTION

GET_DEFAULT is useful for initializing a layered application that uses an X defaults file. You can use GET_DEFAULT only in the DECwindows environment.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to GET_DEFAULT.
TPU$_TOOMANY	ERROR	Too many arguments passed to GET_DEFAULT.
TPU$_NEEDTOASSIGN	ERROR	GET_DEFAULT must return a value.
TPU$_REQUIRESDECW	ERROR	You can use GET_DEFAULT only if you are using DECwindows VAXTPU.

EXAMPLE

```
PROCEDURE application_module_init

LOCAL
    keypad_name;

    :
    :
    :

keypad_name := GET_DEFAULT ("user.keypad", "User.Keypad");

EDIT (keypad_name, UPPER); ! Convert the returned string to uppercase.

IF keypad_name <> '0'
THEN

   CASE keypad_name

      "EDT"      :  eve_set_keypad_edt ();
      "NOEDT"    :  eve_set_keypad_noedt ();
      "WPS"      :  eve_set_keypad_wps ();
      "NOWPS"    :  eve_set_keypad_nowps ();
      "NUMERIC"  :  eve_set_keypad_numeric ();
      "VT100"    :  eve_set_keypad_vt100 ();
      [INRANGE, OUTRANGE]  :  eve_set_keypad_numeric; ! If user has
                                                      ! used invalid value,
                                                      ! set the keypad to
                                                      ! NUMERIC setting.

   ENDCASE;

ENDIF;

    :
    :
    :

ENDPROCEDURE;
```

This code fragment shows the portion of a module_init procedure directing VAXTPU to fetch the value of a resource from the X resources database. For more information on module_init procedures, see Appendix G.

If you want to create an extension of EVE that enables use of an X defaults file to choose a keypad setting, you can use a GET_DEFAULT statement in a module_init procedure.

To provide a value for the GET_DEFAULT statement to fetch, an X defaults file would contain an entry similar to the following:

```
User.Keypad : EDT
```

GET_GLOBAL_SELECT

Supplies information about a global selection.

FORMAT

{ **unspecified** / **string** / **integer** / **array** } := *GET_GLOBAL_SELECT* ({ *PRIMARY* / *SECONDARY* / *selection_name* } , *selection_property_name*)

PARAMETERS

PRIMARY

A keyword indicating that the layered application is requesting information about a property of the primary global selection.

SECONDARY

A keyword indicating that the layered application is requesting information about a property of the secondary global selection.

selection_name

A string identifying the global selection whose property is the subject of the layered application's information request. Specify the selection name as a string if the layered application needs information about a selection other than the primary or secondary global selection.

selection_property_name

A string specifying the property whose value the layered application is requesting.

return value

unspecified	A data type indicating that the information requested by the layered application was not available.
string	The value of the specified global selection property. The return value is of type string if the value of the specified global selection property is of type string.
integer	The value of the specified global selection property. The return value is of type integer if the value of the specified global selection property is of type integer.
array	An array passing information about a global selection whose contents describe information that is not of a data type supported by VAXTPU. VAXTPU does not use or alter the information in the array; the application layered on VAXTPU is responsible for determining how the information is used, if at all. Since the array is used to receive information from other DECwindows applications, all applications that exchange information whose data type is not supported by VAXTPU must adopt a convention on how the information is to be used.

The element *array {0}* contains a string naming the data type of the information being passed. For example, if the information being passed is a span, the element contains the string "*SPAN*". The element *array {1}* contains either the integer 8, indicating that the information is passed as a series of bytes, or the integer 32, indicating that the information is passed as a series of longwords. If *array {1}* contains the value 8, the element *array {2}* contains a string and there are no array elements after *array {2}*. The string does not name anything, but rather is a series of bytes of information. As mentioned, the meaning and use of the information is agreed upon by convention among the DECwindows applications. To interpret this string, the application can use the SUBSTR built-in to obtain substrings one at a time, and the ASCII built-in to convert the data to integer format if necessary. For more information about using these VAXTPU elements, see the description of the SUBSTR and ASCII built-in procedures.

If *array {1}* contains the value 32, the element *array {2}* and any subsequent elements contain integers. The number of integers in the array is determined by the application which responded to the request for information about the global selection. The interpretation of the data is a convention that must be agreed upon by the cooperating application. To determine how many longwords are being passed, an application can determine the length of the array and subtract 2 to allow for elements *array {0}* and *array {1}*.

DESCRIPTION

If an owner for the global selection exists, and if the owner provides the information requested in a format that VAXTPU can recognize, GET_GLOBAL_SELECT returns the information.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to GLOBAL_SELECT.
TPU$_NEEDTOASSIGN	ERROR	GLOBAL_SELECT must return a value.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_REQUIRESDECW	ERROR	You can use GLOBAL_SELECT only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to GLOBAL_SELECT.
TPU$_TOOMANY	ERROR	Too many arguments passed to GLOBAL_SELECT.
TPU$_GBLSELOWNER	WARNING	VAXTPU owns the global selection.

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVGBLSELDATA	WARNING	The global selection owner provided data that VAXTPU cannot process.
TPU$_NOGBLSELDATA	WARNING	The global selection owner has indicated that it cannot provide the information requested.
TPU$_NOGBLSELOWNER	WARNING	You have requested information about an unowned global selection.
TPU$_TIMEOUT	WARNING	The global selection owner did not respond before the timeout period expired.

EXAMPLE

```
string_to_paste := GET_GLOBAL_SELECT (PRIMARY, "STRING");
```

This statement fetches the text in the primary global selection and assigns it to the variable *string_to_paste*.

For another example of how to use the GET_GLOBAL_SELECT built-in, see Example B–4.

GET_INFO

Returns information about the current status of the editor.

For information on how to get a screen display of the status of your editor, see the description of the built-in procedure SHOW.

DESCRIPTION

This description provides general information on the GET_INFO built-ins. In this part, you can also find descriptions of individual GET_INFO built-ins. The individual GET_INFO built-ins are grouped according to the value of their first parameter. For a list of the groups of GET_INFO built-ins, see Table 7–2.

All GET_INFO built-in procedures have the following two characteristics in common:

- They return a value that is the piece of information you have requested.
- They consist of the GET_INFO statement followed by at least two parameters, as follows:
  - The first parameter specifies the general topic about which you want information. If you want the GET_INFO built-in to return information on a given variable, use that variable as the first parameter. For example, if you want to know what row contains the cursor in a window stored in the variable *command_window*, you would specify the variable *command_window* as the first parameter. Thus, you would use use the following statement:

    ```
    the_row := GET_INFO (command_window, "current_row");
    ```

    Otherwise, the first parameter is a keyword specifying the general subject about which GET_INFO is to return information. The valid keywords for the first parameter are as follows:

    ARRAY
    BUFFER
    COMMAND_LINE
    DEBUG
    DEFINED_KEY
    KEY_MAP
    KEY_MAP_LIST
    mouse_event_keyword
    PROCEDURES
    PROCESS
    SCREEN
    SYSTEM
    WINDOW
    WIDGET

    For a list of valid mouse event keywords, see Table 7–3.

Do not confuse a GET_INFO built-in whose first parameter is a keyword (such as ARRAY) with a GET_INFO built-in whose first parameter is a variable of a given data type, such as *array_variable*. For example, the built-in GET_INFO (array_variable) shows what string constants can be used when the first parameter is an array variable, while the built-in GET_INFO (ARRAY) shows what can be used when the first parameter is the keyword ARRAY.

— The second parameter (a VAXTPU string) specifies the exact piece of information you want.

— The third and subsequent parameters, if necessary, provide additional information that VAXTPU uses to identify and return the requested value or structure.

Each GET_INFO built-in in this section shows the possible return values for a given combination of the first and second parameters. For example, the built-in GET_INFO (any_variable) shows that when you use any variable as the first parameter and the string "*type*" as the second parameter, GET_INFO returns a keyword for the data type of the variable.

Depending upon the kind of information requested, GET_INFO returns any one of the following:

- An array
- A buffer
- An integer
- A keyword
- A marker
- A process
- A range
- A string
- A window

VAXTPU maintains internal lists of the following items:

- Arrays
- Array elements
- Breakpoints
- Buffers
- Defined keys
- Key maps
- Key map lists
- Processes
- Windows

You can step through an internally-maintained list by using *"first"*, *"next"*, *"previous"*, or *"last"* as the second parameter to GET_INFO. Note that the order in which VAXTPU maintains these lists is private and may change in a future version. Do not write code that depends on a list being maintained in a particular order. When you write code to search a list, remember that VAXTPU keeps only one pointer for each list. If you create nested loops that attempt to search the same list, the results are unpredictable. For example, suppose that a program intended to search two key map lists for common key maps sets up a loop within a loop. The outer loop might contain the following statement:

```
GET_INFO (KEY_MAP, "previous", name_of_second_key_map)
```

The inner loop might contain the following statement:

```
GET_INFO (KEY_MAP, "next", name_of_first_key_map)
```

In VAXTPU, the behavior of such a nested loop is unpredictable.

Unless documented otherwise, the order of the internal list is not defined.

The syntax of GET_INFO depends on the kind of information you are trying to get. For more information on specific GET_INFO built-ins, see the descriptions in this section. GET_INFO built-ins whose first parameter is a keyword are grouped separately from GET_INFO built-ins whose first parameter is a variable.

Table 7–2 GET_INFO Built-in Procedures by First Parameter

Variable	Keyword	Any Keyword or Key Name
GET_INFO (any_variable)	GET_INFO (ARRAY)	GET_INFO (any_keyname)
GET_INFO (array_variable)	GET_INFO (BUFFER)	GET_INFO (any_keyword)
GET_INFO (buffer_variable)	GET_INFO (COMMAND_LINE)	
GET_INFO (integer_variable)	GET_INFO (DEBUG)	
GET_INFO (marker_variable)	GET_INFO (DEFINED_KEY)	
GET_INFO (process_variable)	GET_INFO (KEY_MAP)	
GET_INFO (range_variable)	GET_INFO (KEY_MAP_LIST)	
GET_INFO (string_variable)	GET_INFO (mouse_event_keyword)	
GET_INFO (widget_variable)	GET_INFO (PROCEDURES)	
GET_INFO (window_variable)	GET_INFO (PROCESS)	
	GET_INFO (SCREEN)	
	GET_INFO (SYSTEM)	
	GET_INFO (WIDGET)	
	GET_INFO (WINDOW)	

SIGNALED ERRORS

TPU$_BADREQUEST	WARNING	Request represented by second argument is not understood for data type of first argument.
TPU$_BADKEY	WARNING	Bad keyword value or unrecognized data type is passed as the first argument.
TPU$_NOCURRENTBUF	WARNING	Current buffer is not defined.
TPU$_NOKEYMAP	WARNING	Key map is not defined.
TPU$_NOKEYMAPLIST	WARNING	Key map list is not defined.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong data type.
TPU$_NEEDTOASSIGN	ERROR	The GET_INFO built-in can only be used on the right-hand side of an assignment statement.
TPU$_NOBREAKPOINT	WARNING	This string constant is valid only after a breakpoint.
TPU$_NONAMES	WARNING	There are no names matching the one requested.
TPU$_TOOFEW	ERROR	Too few arguments passed to the GET_INFO built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the GET_INFO built-in.
TPU$_UNKKEYWORD	ERROR	An unknown keyword has been used as an argument.

EXAMPLES

1
```
my_buffer := GET_INFO (BUFFERS, "current");
```

This assignment statement stores the pointer to the current buffer in the variable *my_buffer*.

2
```
my_string := GET_INFO (my_buffer, "file_name");
```

This assignment statement stores the name of the input file for *my_buffer* in the variable *my_string*.

3
```
my_buffer := GET_INFO (BUFFERS, "current");
```

This assignment statement stores a reference to the current buffer in the variable *my_buffer*.

```
4   my_string := GET_INFO (CURRENT_BUFFER, "file_name");
```

This statement calls the CURRENT_BUFFER built-in, which returns the current buffer. The GET_INFO built-in determines the name of the input file associated with the current buffer. The input filename is assigned to the variable *my_string*.

```
5   is_buf_mod := GET_INFO (CURRENT_BUFFER, "modified");
```

This assignment statement stores the integer 1 or 0 in the variable *is_buf_mod*. A value of 1 means the current buffer has been modified. A value of 0 means the current buffer has not been modified.

```
6   my_window := GET_INFO (WINDOWS, "current");
    length_integer := GET_INFO (my_window, "length", visible_window);
    width_integer := GET_INFO (my_window, "width");
```

These assignment statements store the size of the current window in the variables *length_integer* and *width_integer*.

```
7   PROCEDURE user_getinfo

       top_of_window := GET_INFO (CURRENT_WINDOW, "top", visible_window);

           ! Remove the top five lines from the main window
       ADJUST_WINDOW (CURRENT_WINDOW, +5, 0);

           ! Replace removed lines with an example window
       example_window := CREATE_WINDOW (top_of_window, 5, ON);
       example_buffer := CREATE_BUFFER ("EXAMPLE",
                                        "sys$login:template.txt");
       MAP (example_window, example_buffer);
    ENDPROCEDURE;
```

This procedure uses GET_INFO to find the top of the current window. It then removes the top five lines and replaces them with an example window.

```
8   PROCEDURE user_display_key_map_list

       current_key_map_list := GET_INFO (CURRENT_BUFFER,
                               "key_map_list");
       MESSAGE (current_key_map_list);
    ENDPROCEDURE;
```

This procedure retrieves and displays the name of the key map list in the current buffer.

```
9   PROCEDURE show_key_map_lists

       LOCAL key_map_list_name;

       key_map_list_name := GET_INFO (KEY_MAP_LIST, "first");
       LOOP
          EXITIF key_map_list_name = 0;
          SPLIT_LINE;
          COPY_TEXT (key_map_list_name);
          key_map_list_name := GET_INFO (KEY_MAP_LIST, "next");
      ENDLOOP;
    ENDPROCEDURE;
```

This procedure displays all the key map lists.

10
```
PROCEDURE show_self_insert
   LOCAL key_map_list_name;
   key_map_list_name := GET_INFO (CURRENT_BUFFER, "key_map_list");
   IF GET_INFO (key_map_list_name, "self_insert")
   THEN
      MESSAGE ("Undefined printable characters will be inserted");
   ELSE
      MESSAGE ("Undefined printable characters will cause an error");
   ENDIF;
ENDPROCEDURE;
```

This procedure shows whether the key map list associated with the current buffer inserts undefined printable characters.

11
```
PROCEDURE show_key_maps_in_list (key_map_list_name)
   LOCAL key_map_name;
    key_map_name := GET_INFO (KEY_MAP, "first", key_map_list_name);
    LOOP
       EXITIF key_map_name = 0;
       SPLIT_LINE;
       COPY_TEXT (key_map_name);
       key_map_name := GET_INFO (KEY_MAP, "next", key_map_list_name);
   ENDLOOP;
ENDPROCEDURE;
```

This procedure displays the key maps in the key map list *key_map_list_name*.

GET_INFO (any_keyname)

Returns a keyword describing the type of key named by *any_keyname.*

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

{ **keyword** / **integer** } := **GET_INFO** *(any_keyname,* { *"key_modifiers"* / *"key_type"* } *)*

PARAMETERS

"key_modifiers"

Returns a bit-encoded integer indicating what key modifier or modifiers were used to create the VAXTPU key name specified by the parameter *any_keyname.* For more information about the meaning and possible values of key modifiers, see the description of the KEY_NAME built-in.

VAXTPU defines four constants to be used when referring to or testing the numerical value of key modifiers. The correspondence between key modifiers, defined constants, and bit-encoded integers is as follows:

Key Modifier	Constant	Bit-Encoded Integer
SHIFT_MODIFIED	TPU$K_SHIFT_MODIFIED	1
CTRL_MODIFIED	TPU$K_CTRL_MODIFIED	2
HELP_MODIFIED	TPU$K_HELP_MODIFIED	4
ALT_MODIFIED	TPU$K_ALT_MODIFIED	8

Note that the keyword SHIFT_KEY may have been used to create a VAXTPU key name. SHIFT_KEY is not a modifier, it is a prefix. The SHIFT key, also called the GOLD key by the EVE editor, is pressed and released before any other key is pressed. In DECwindows, modifying keys such as the CTRL key are pressed and held down while the modified key is pressed.

Note, too, that if more than one key modifier was used with the KEY_NAME built-in, the value of the returned integer is the sum of the integer representations of the key modifiers. For example, if you create a key name using the modifiers HELP_MODIFIED and ALT_MODIFIED, the built-in GET_INFO (key_name, "key_modifiers") returns the integer 12.

"key_type"

Returns a keyword describing the type of key named by *any_keyname.* The keywords that can be returned are PRINTING, KEYPAD, FUNCTION, SHIFT_KEY, KEYPAD, SHIFT_FUNCTION, and SHIFT_CONTROL. Returns 0 if *parameter1* is not a valid key name. Note that there are cases in which GET_INFO (any_keyname, "name") returns the keyword PRINTING but the key described by the keyname is not associated with a printable character. For example, if you use the KEY_NAME built-in to define a key name as the combination of the character A and the ALT modifier, and if you then use GET_INFO (any_keyname, "name") to find out how VAXTPU classifies the key, the GET_INFO built-in returns the

keyword PRINTING. However, if you use the ASCII built-in to obtain the string representation of the key, the ASCII built-in returns a null string because ALT/A is not printable.

EXAMPLE

```
new_key := KEY_NAME (KP4, SHIFT_MODIFIED, CTRL_MODIFIED);
modifier_value := GET_INFO (new_key, "key_modifiers");
MESSAGE (STR (modifier_value));
IF GET_INFO (new_key, "key_modifiers")
THEN
    the_name := GET_INFO (new_key, "name")
    MESSAGE (STR (the_name));
ENDIF;
```

The first statement in the preceding code creates a VAXTPU key name for the key sequence produced by pressing the CTRL key, the SHIFT key, and the 4 key on the keypad all at once. The new key name is assigned to the variable *new_key*. The second statement fetches the integer equivalent of this combination of key modifiers. The third statement displays the integer 3 in the message buffer. The IF clause of the fourth statement shows how to test whether a key name was created using a modifier. (Note, however, that this statement does not detect whether a key name was created using the keyword SHIFT_KEY.) The THEN clause shows how to fetch the key modifier keyword or keywords used to create a key name. The final statement displays the string KEY_NAME (KP4, SHIFT_MODIFIED, CTRL_MODIFIED) in the message buffer.

GET_INFO (any_keyword)

Returns the string representation of the keyword specified in the first parameter to GET_INFO.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO. See also the description of GET_INFO (integer_ variable).

FORMAT

string := GET_INFO *(keyword, "name")*

PARAMETERS

keyword

Returns a VAXTPU keyword whose string equivalent you want GET_INFO to return.

You can use GET_INFO (keyword, "name") to obtain the string equivalent of a key name. This is useful for displaying screen messages about keys. For example, to obtain the string equivalent of the key name PF1, you could use the following statement:

```
the_string := GET_INFO (PF1, "name");
```

If a key name is created using several key modifiers, the built-in returns the string representations of all the keywords used to create the key name. For more information on creating key names, see the description of the KEY_NAME built-in.

The following code fragment shows one possible use of GET_INFO (keyword_variable, "name"):

```
new_key := KEY_NAME (KP4, SHIFT_MODIFIED, CTRL_MODIFIED);
!    .
!    .
!    .
IF GET_INFO (new_key, "key_modifiers") <> 0
THEN
    the_name := GET_INFO (new_key, "name")
ENDIF;
MESSAGE (STR (the_name));
```

The first statement creates a VAXTPU key name for the key sequence produced by pressing the CTRL key, the SHIFT key, and the 4 key on the keypad all at once. The new key name is assigned to the variable *key_name*. The IF clause of the statement shows how to test whether a key name was created using one or more key modifier keywords. (Note, however, that this statement does not detect whether a key name was created using the keyword SHIFT_KEY. The built-in GET_ INFO (key_name, "key_modifiers") returns 0 even if the key name was created using SHIFT_KEY.) The THEN clause shows how to fetch the key modifier keyword or keywords used to create a key name. The final statement displays the string KEY_NAME (KP4, SHIFT_MODIFIED, ALT_MODIFIED) in the message buffer.

"name"

Returns the string equivalent of the specified keyword.

GET_INFO (any_variable)

Returns a keyword specifying the data type of the variable.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

keyword := GET_INFO *(any_variable, "type")*

PARAMETERS

"type"

Returns a keyword that is the data type of the variable specified in *any_variable*.

EXAMPLE

```
IF  GET_INFO (select_popup, "type") <> WIDGET
    THEN
        MESSAGE ("Select_popup widget not created.")
ENDIF;
```

The preceding code tests whether the variable *select_popup* has been assigned a widget instance. If not, the code causes a message to be displayed on the screen.

GET_INFO (ARRAY)

Returns an array in VAXTPU's internal list of arrays.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

array := GET_INFO *(ARRAY,* { *"current"* | *"first"* | *"last"* | *"next"* | *"previous"* } *)*

PARAMETERS

"current"

Returns the current array in VAXTPU's internal list of arrays. You must use either GET_INFO (ARRAY, "first") or GET_INFO (ARRAY, "last") before you can use GET_INFO (ARRAY, "current"). If you use these built-ins in the wrong order or if no arrays have been created, GET_INFO (ARRAY, "current") returns 0.

"first"

Returns the first array in the VAXTPU internal list of arrays. Returns 0 if no arrays are defined.

"last"

Returns the last array in the VAXTPU internal list of arrays. Returns 0 if no arrays are defined.

"next"

Returns the next array in VAXTPU's internal list of arrays. You must use GET_INFO (ARRAY, "first") before you can use GET_INFO (ARRAY, "next"). Returns 0 if no arrays are defined.

"previous"

Returns the previous array in VAXTPU's internal list of arrays. You must use either GET_INFO (ARRAY, "current") or GET_INFO (ARRAY, "last") before you can use GET_INFO (ARRAY, "previous"). If you use these built-ins in the wrong order or if no arrays have been created, GET_INFO (ARRAY, "previous") returns 0.

GET_INFO (array_variable)

Returns information about a specified array.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

array
buffer
integer
keyword
marker
process
range
string
widget
window
unspecified

:= GET_INFO *(array_variable,*

"current"
"first"
"high_index"
"last"
"low_index"
"next"
"previous"

)

PARAMETERS

"current"

Returns the index value of the current element of the specified array, whether the index is of type integer or some other type. Returns any type except program, pattern, or learn. Returns the type unspecified if there is no current element.

You must use either GET_INFO (array_variable, "first") or GET_INFO (array_variable, "last") before you can use GET_INFO (array_variable, "current").

"first"

Returns the index value of the first element of the specified array, whether the index is of type integer or some other type. Returns any type except program, pattern, or learn. Returns the type unspecified if there is no first element.

"high_index"

Returns an integer that is the highest valid integer index for the static predeclared portion of the array. If the GET_INFO call returns a high index lower than the low index, the array has no static portion.

"last"

Returns the index value of the last element of the specified array, whether the index is of type integer or some other type. Returns any type except program, pattern, or learn. Returns the type unspecified if there is no last element.

"low_index"

Returns an integer that is the lowest valid integer index for the static predeclared portion of the array. If the GET_INFO call returns a high index lower than the low index, the array has no static portion.

"next"

Returns the index value of the next element of the specified array, whether the index is of type integer or some other type. Returns any type except program, pattern, or learn. Returns the type unspecified if there is no next element.

You must use GET_INFO (array_variable, "first") before you can use GET_INFO (array_variable, "next").

"previous"

Returns the index value of the previous element of the specified array, whether the index is of type integer or some other type. Returns any type except program, pattern, or learn. Returns the type unspecified if there is no previous element.

You must use either GET_INFO (array_variable, "current") or GET_INFO (array_variable, "last") before you can use GET_INFO (array_variable, "previous").

GET_INFO (BUFFER)

Returns a buffer in VAXTPU's internal list of buffers.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

buffer := GET_INFO *(BUFFER[[S]]* { *"current"* | *"erase_unmodifiable"* | *"find_buffer", buffer_name* | *"first"* | *"last"* | *"next"* | *"previous"* })

PARAMETERS

"current"

Returns the current buffer in VAXTPU's internal list of buffers. Returns 0 if there is no current buffer.

GET_INFO (BUFFER[[S]], "current") always returns the current buffer, regardless of whether or you have first used GET_INFO (BUFFER[[S]], "first") or GET_INFO (BUFFER[[S]], "last"). Thus, GET_INFO (BUFFER[[S]], "current") is equivalent to the built-in CURRENT_BUFFER.

"erase_unmodifiable"

Returns 1 if unmodifiable records can be erased from the specified buffer and returns 0 if the records cannot be erased.

"find_buffer"

Returns the buffer whose name you specify (as a string) as the third parameter. Returns 0 if no buffer with the name you specify is found.

"first"

Returns the first buffer in VAXTPU's internal list of buffers. Returns 0 if there is none.

"last"

Returns the last buffer in VAXTPU's internal list of buffers. Returns 0 if there is none.

"next"

The next buffer in VAXTPU's internal list of buffers. Returns 0 if there are no more.

"previous"

Returns the preceding buffer in VAXTPU's internal list of buffers. Returns 0 if there is none.

GET_INFO (buffer_variable)

Returns information about a specified buffer.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ integer
  keyword
  learn_sequence
  marker
  program
  range
  string } := GET_INFO (buffer_variable,
                        { "before_bol"
                          "beyond_eob"
                          "beyond_eol"
                          "bound"
                          "character"
                          "direction"
                          "eob_text"
                          "erase_unmodifiable"
                          "file_name"
                          "first_marker"
                          "first_range"
                          "journaling"
                          "journal_file"
                          "journal_name"
                          "key_map_list"
                          "left_margin"
                          "left_margin_action"
                          "line"
                          "map_count"
                          "max_lines"
                          "middle_of_tab"
                          "mode"
                          "modifiable"
                          "modified"
                          "name"
                          "next_marker"
                          "next_range"
                          "no_write"
                          "offset"
                          "offset_column"
                          "output_file" }
```

```
                  "permanent"
                  "read_routine", GLOBAL_SELECT
                  "record_count"
                  "record_number"
                  "record_size"
                  "right_margin"                  )
                  "right_margin_action"
                  "safe_for_journaling"
                  "system"
                  "tab_stops"
                  "unmodifiable_records"
```

PARAMETERS

"before_bol"

Returns an integer (1 or 0) that indicates whether the editing point is located before the beginning of a line.

"beyond_eob"

Returns an integer (1 or 0) that indicates whether the editing point is located beyond the end of a buffer.

"beyond_eol"

Returns an integer (1 or 0) that indicates whether the editing point is located beyond the end of a line.

"bound"

Returns an integer (1 or 0) that indicates whether or not the marker that is the specified buffer's editing point is bound to text. For more information about bound markers, see Chapter 2.

"character"

Returns a string that is the character at the editing point for the buffer.

"direction"

Returns the keyword FORWARD or REVERSE. This parameter is established or changed with the built-in procedures SET (FORWARD) and SET (REVERSE).

"eob_text"

Returns a string representing the end-of-buffer text. This parameter is established or changed with the built-in procedure SET (EOB_TEXT).

"erase_unmodifiable"

Returns 1 if unmodifiable records can be erased from the specified buffer and returns 0 if the records cannot be erased.

"file_name"

Returns a string that is the name of a file given as the second parameter to CREATE_BUFFER; null if none was specified.

"first_marker"

Returns the first marker in VAXTPU's internal list of markers for the buffer. Returns 0 if there is none. You must use GET_INFO (buffer_variable, "first_marker") before the first use of GET_INFO (buffer_

variable, "next_marker"). If you do not follow this rule, GET_INFO (buffer_variable, "next_marker") returns 0.

Note that there is no corresponding *"last_marker"* or *"prev_marker"* parameter.

Do not write code that relies on VAXTPU storing markers in one particular order. Creating markers or ranges may alter the internal order. In addition, the internal ordering may change in future releases.

"first_range"

Returns the first range in VAXTPU's internal list of ranges for the buffer. Returns 0 if there are none. You must use GET_INFO (buffer_variable, "first_range") before you use GET_INFO (buffer_variable, "next_range") or the *"next_range"* built-in returns 0.

Note that there is no corresponding *"last_range"* or *"prev_range"* parameter.

Do not write code that relies on VAXTPU storing ranges in one particular order. Creating markers or ranges may alter the internal order. In addition, the internal ordering may change in future releases.

"journaling"

Returns 1 if the specified buffer is being journaled or returns 0 if it is not.

"journal_file"

Returns a string that is the name of the journal file for the specified buffer. If the buffer is not being journaled, the call returns 0.

"journal_name"

Converts a buffer's name to a journal file name using the VAXTPU default journal file name algorithm. VAXTPU converts the buffer name to a journal file name regardless of journaling status. The GET_INFO call does not require journaling to be turned on for the specified buffer. For more information on this algorithm, see Section 1.7.1.

"key_map_list"

Returns a string that is the key map list bound to the buffer. This parameter is established or changed with the built-in procedure SET.

"left_margin"

Returns an integer that is the current left margin setting. This parameter is established or changed with the built-in procedure SET (LEFT_MARGIN).

"left_margin_action"

Returns a program or learn sequence specifying what VAXTPU should do if the user tries to insert text to the left of the left margin. Returns UNSPECIFIED if no left margin action routine has been set. This parameter is established or changed with the built-in procedure SET (LEFT_MARGIN_ACTION).

"line"

Returns a string that is the line of text at the editing point for the buffer.

"map_count"
Returns an integer that is the number of windows associated with the buffer.

"max_lines"
Returns an integer that is the maximum number of records (lines) in the buffer. This parameter is established or changed with the built-in procedure SET.

"middle_of_tab"
Returns an integer (1 or 0) that indicates whether the editing point is located in the white space within a tab.

"mode"
Returns the keyword INSERT or OVERSTRIKE. This parameter is established or changed with the built-in procedures SET (INSERT) and SET (OVERSTRIKE).

"modifiable"
Returns an integer (1 or 0) that indicates whether the buffer is modifiable.

"modified"
Returns an integer (1 or 0) that indicates whether the buffer has been modified.

"name"
Returns a string that is the name given to the buffer when it was created.

"next_marker"
Returns the next marker in VAXTPU's internal list of markers for the buffer. Returns 0 if there are no more. You must use GET_INFO (buffer_variable, "first_marker") before you use GET_INFO (buffer_variable, "next_marker") or the "*next_marker*" built-in returns 0.

Note that there is no corresponding "*last_marker*" or "*prev_marker*" parameter.

Do not write code that relies on VAXTPU storing markers in one particular order. Creating markers or ranges may alter the internal order. In addition, the internal ordering may change in future releases.

"next_range"
Returns the next range in VAXTPU's internal list of ranges for the buffer. Returns 0 if there are no more. You must use GET_INFO (buffer_variable, "first_range") before you use GET_INFO (buffer_variable, "next_range") or the "*next_range*" built-in returns 0.

Note that there is no corresponding "*last_range*" or "*prev_range*" parameter.

Do not write code that relies on VAXTPU storing ranges in one particular order. Creating markers or ranges may alter the internal order. In addition, the internal ordering may change in future releases.

"no_write"

Returns an integer (1 or 0) that indicates whether the buffer should be written to a file at exit time. Note that VAXTPU writes the buffer to a file only if the buffer has been modified during the editing session. This parameter is established or changed with the built-in procedure SET (NO_WRITE).

"offset"

Returns an integer that is the number of characters between the left margin and the editing point. The left margin is counted as character 0. A tab is counted as one character, regardless of width. Window shifts have no effect on the value returned when you use "*offset*". The value returned has no relation to the visible screen column in which a character is displayed.

"offset_column"

Returns an integer that is the screen column in which VAXTPU displays the character at the editing point. When calculating this value, VAXTPU does not take window shifts into account; VAXTPU assumes that any window mapped to the current buffer is not shifted. The value returned when you use "*offset_column*" reflects the location of the left margin and the width of tabs preceding the editing point. In contrast, the value returned when you use *"offset"* is not affected by the location of the left margin or the width of tabs.

"output_file"

Returns a string that is the name of the file used with the built-in procedure SET (OUTPUT_FILE). Returns 0 if there is no output file associated with the specified buffer. This parameter is established or changed with the built-in procedure SET (OUTPUT_FILE).

"permanent"

Returns an integer (1 or 0) that indicates whether the buffer is permanent or can be deleted. This parameter is established or changed with the built-in procedure SET (PERMANENT).

"read_routine"

This parameter is used with DECwindows only.

Returns the program or learn sequence that VAXTPU executes when it owns a global selection and another application has requested information about that selection. If the application has not specified a global selection read routine, 0 is returned.

GLOBAL_SELECT is a keyword indicating that the built-in is to return the global selection read routine. When you use "*read_routine*" as the second parameter to this built-in, you must use the keyword GLOBAL_SELECT as the third parameter, as follows:

```
GET_INFO (buffer_variable, "read_routine", GLOBAL_SELECT)
```

"record_count"

Returns an integer that is the number of records (lines) in the buffer. Note that GET_INFO (buffer, "record_count") does not count the end-of-buffer text as a record, but GET_INFO (marker, "record_number") does if the specified marker is on the end-of-buffer text. Thus, the maximum value returned by GET_INFO (buffer, "record_count") is one less than the

maximum value returned by GET_INFO (marker, "record_number") if the specified marker is on the end-of-buffer text.

"record_number"
Returns the record number of the editing point.

"record_size"
Returns an integer that is the maximum length for records (lines) in the buffer.

"right_margin"
Returns an integer that is the current right margin setting. This parameter is established or changed with the built-in procedure SET (RIGHT_MARGIN).

"right_margin_action"
Returns a program or learn sequence specifying what VAXTPU should do if the user tries to insert text to the right of the right margin. Returns TPU$K_UNSPECIFIED if the buffer does not have a right margin action.

This parameter is established or changed with the built-in procedure SET (RIGHT_MARGIN_ACTION).

"safe_for_journaling"
Returns 1 if the specified buffer is safe for journaling or returns 0 if it is not. "Safe_for_journaling" means that journaling can be turned on by using the SET (JOURNALING) built-in procedure. A buffer is safe for journaling if it is empty, has never been modified, or has not been modified since the last time it was written to a file.

"system"
Returns an integer (1 or 0) that indicates whether the buffer is a system buffer. This parameter is established or changed with the built-in procedure SET (SYSTEM).

"tab_stops"
Returns either an integer or a string. Use the built-in SET (TAB_STOPS) to determine the data type of the return value. If you specify a return value of type string, the built-in GET_INFO (buffer_variable, "tab_stops") returns a string representation of all the column numbers where tab stops are set. The column numbers are separated by spaces. If you specify a return value of type integer, the return value is the number of columns between tab stops.

"unmodifiable_records"
Returns 1 if the specified buffer contains one or more unmodifiable records. The call returns 0 if no unmodifiable records are present in the specified buffer.

GET_INFO (COMMAND_LINE)

Returns information about the command line used to invoke VAXTPU.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ integer }
{ string  } := GET_INFO (COMMAND_LINE,
                         { "character"
                         | "command"
                         | "command_file"
                         | "create"
                         | "display"
                         | "file_name"
                         | "initialization"
                         | "init_file"
                         | "initialization_file"
                         | "journal"
                         | "journal_file"
                         | "line"
                         | "modify"
                         | "nomodify"
                         | "output"
                         | "output_file"
                         | "read_only"
                         | "recover"
                         | "start_character"
                         | "start_record"
                         | "section"
                         | "section_file"
                         | "work_file"
                         | "write"                 }
```

PARAMETERS

"character"

Returns an integer that is the column number of the character position specified by the /START_POSITION command qualifier. This parameter is useful in a procedure to determine where VAXTPU should place the cursor at startup time. The default is 1 if the /START_POSITION qualifier is not specified. This parameter is the same as the "*start_character*" parameter.

"command"

Returns an integer (1 or 0) that indicates whether /COMMAND was specified when you invoked VAXTPU.

"command_file"

Returns a string that is the command file specification.

"create"
Returns an integer (1 or 0) that indicates whether /CREATE is active (either by default or because /CREATE was specified when VAXTPU was invoked).

"display"
Returns an integer (1 or 0) that indicates whether /DISPLAY or /INTERFACE is active (either by default, or because /DISPLAY or /INTERFACE was specified when VAXTPU was invoked).

"file_name"
Returns a string that is a file specification used as a parameter when the user invokes VAXTPU.

"initialization"
Returns an integer (1 or 0) that indicates whether /INITIALIZATION is active (either by default or because /INITIALIZATION was specified when VAXTPU was invoked).

"init_file"
Returns a string that is a file specification for /INITIALIZATION. This is a synonym for GET_INFO (COMMAND_LINE, "initialization_file").

"initialization_file"
Returns a string that is the initialization file specification for /INITIALIZATION.

"journal"
Returns an integer (1 or 0) that indicates whether /JOURNAL is active (either by default or because /JOURNAL was specified when VAXTPU was invoked).

"journal_file"
Returns a string that is the journal file specification for /JOURNAL.

"line"
Returns an integer that is the record number of the line specified by the /START_POSITION command qualifier. This parameter is useful in a procedure to determine where VAXTPU should place the cursor at startup time. The default is 1 if the /START_POSITION qualifier is not specified. This parameter is the same as the "*start_record*" parameter.

"modify"
Returns an integer (1 or 0) that indicates whether the qualifier /MODIFY was specified when VAXTPU was invoked by the user or by another program.

"nomodify"
Returns an integer (1 or 0) that indicates whether the qualifier /NOMODIFY was specified when VAXTPU was invoked by the user or by another program.

"output"
Returns an integer (1 or 0) that indicates whether /OUTPUT is active (either by default or because /OUTPUT was specified when VAXTPU was invoked).

"output_file"
Returns a string that is the output file specification for /OUTPUT.

"read_only"
Returns an integer (1 or 0) that indicates whether /READ_ONLY was specified when VAXTPU was invoked. For more information on this call, see Chapter 5.

"recover"
Returns an integer (1 or 0) that indicates whether /RECOVER was specified when VAXTPU was invoked.

"start_character"
Returns an integer that is the column number of the character position specified by the /START_POSITION command qualifier. This parameter is useful in a procedure to determine where VAXTPU should place the cursor at startup time. The default is 1 if the /START_POSITION qualifier is not specified.

This parameter is a synonym for "*character*".

"start_record"
Returns an integer that is the record number of the line specified by the /START_POSITION command qualifier. This parameter is useful in a procedure to determine where VAXTPU should place the cursor at startup time. The default is 1 if the /START_POSITION qualifier is not specified. This parameter is a synonym for "*line*".

"section"
Returns an integer (1 or 0) that indicates whether /SECTION is active (either by default or because /SECTION was specified when VAXTPU was invoked).

"section_file"
Returns a string that is the section file specification for /SECTION.

"work_file"
Returns a string that is the work file specification for /WORK.

"write"
Returns an integer (1 or 0) that indicates whether /WRITE was specified when VAXTPU was invoked. For more information on this statement, see Chapter 5.

GET_INFO (DEBUG)

Returns information about the status of a debugging session using the VAXTPU Debugger.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ string    }
{ contents  }
{ integer   } := GET_INFO (DEBUG,
{ parameter }
{ string    }
{ variable  }
                { "breakpoint"                  }
                { "examine", variable_name      }
                { "line_number"                 }
                { "local"                       } )
                { "next"                        }
                { "parameter"                   }
                { "previous"                    }
                { "procedure"                   }
```

PARAMETERS

"breakpoint"

Returns a string that is the name of the first breakpoint. This establishes a breakpoint context for the *"next"* and *"previous"* parameters. TPU$_NONAMES is returned if there are no breakpoints.

"examine"

Returns the contents of the specified variable. TPU$_NONAMES is returned if the specified variable cannot be found.

You must specify a string containing the name of the variable as the third parameter to GET_INFO (DEBUG, "examine").

"line_number"

Returns an integer that is the line number of the breakpoint within the procedure. If the procedure is unnamed, 0 is returned.

"local"

Returns the first local variable in the procedure. This establishes a context for the *"next"* and *"previous"* parameters. TPU$_NONAMES is returned if there are no local variables.

"next"

Returns the next parameter, local variable, or breakpoint. Before using GET_INFO (DEBUG, "next"), you must first use one of the following built-ins:

- GET_INFO (DEBUG, "local")
- GET_INFO (DEBUG, "breakpoint")
- GET_INFO (DEBUG, "parameter")

TPU$_NONAMES is returned if there are no more.

"parameter"

Returns the first parameter of the procedure. GET_INFO (DEBUG, "parameter") causes the VAXTPU Debugger to construct a list of all the formal parameters of the procedure you are debugging. Once this list is constructed, you can use GET_INFO (DEBUG, "next") and GET_INFO (DEBUG, "previous"). VAXTPU signals TPU$_NONAMES if the procedure you are debugging does not have any parameters.

"previous"

Returns the previous parameter, local variable, or breakpoint. TPU$_ NONAMES is returned if there are no more.

"procedure"

Returns a string that is the name of the procedure containing the breakpoint. The null string is returned if the procedure has no name.

GET_INFO (DEFINED_KEY)

Returns a keyword that is the key name of a specified key. GET_INFO (DEFINED_KEY) takes a string as a third parameter. The string specifies the name of either the key map or key map list to be searched.

Note that *"current"* is not valid when the first parameter is DEFINED_KEY or KEY_MAP, although it is valid when the first parameter is KEY_MAP_LIST.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

keyword := GET_INFO *(DEFINED_KEY,* { *"first"* | *"last"* | *"next"* | *"previous"* } *, name_string)*

PARAMETERS

"first"

Returns a keyword that is the key name of the first key in the specified key map or key map list.

"last"

Returns a keyword that is the key name of the last key in the specified key map or key map list.

"next"

Returns a keyword that is the key name of the next key in the specified key map or key map list. Returns 0 if last. Use string constant *"first"* before using *"next."*

"previous"

Returns a keyword that is the key name of the previous key in the specified key map or key map list. Returns 0 if first. Use *"last"* before using *"previous."*

GET_INFO (integer_variable)

Returns the string representation of any integer that is an equivalent of a keyword.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO. See also the description of GET_INFO (any_keyword).

FORMAT

string := GET_INFO *(integer, "name")*

PARAMETERS

integer

Returns an integer that is the equivalent of a VAXTPU keyword. When you use GET_INFO (integer, "name"), the built-in returns the string representation of the keyword that is equivalent to the specified integer.

For example, the following statement assigns the string *object* to the variable *equiv_string*:

```
equiv_string := GET_INFO (10, "name");
```

(The value 14 is the integer equivalent of the keyword PROCESS.)

Note that you should not use the integer equivalents of keywords in VAXTPU code. Digital does not guarantee that the existing equivalences between integers and keywords will always remain the same.

"name"

Returns the string equivalent of the specified integer or keyword.

GET_INFO (KEY_MAP)

Returns information about a key map in a specified key map list. GET_INFO (KEY_MAP) takes a string as a third parameter. The string specifies the name of the key map list to be searched.

Note that *"current"* is not valid when the first parameter is DEFINED_KEY or KEY_MAP, although it is valid when the first parameter is KEY_MAP_LIST.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

{ **string** / **integer** } := **GET_INFO** *(KEY_MAP,* { *"first"* / *"last"* / *"next"* / *"previous"* } *, name_string)*

PARAMETERS

"first"

Returns a string that is the name of the first key map in the key map list. Returns 0 if there is none.

"last"

Returns a string that is the name of the last key map in the key map list. Returns 0 if there is none.

"next"

Returns a string that is the name of the next key map in the key map list. Returns 0 if there is none. Use string constant *"first"* before using *"next."*

"previous"

Returns a string that is the name of the previous key map in the key map list. Returns 0 if there is none. Use *"last"* before using *"previous."*

GET_INFO (KEY_MAP_LIST)

Returns information about a key map list.

Note that *"current"* is not valid when the first parameter is DEFINED_KEY or KEY_MAP, although it is valid when the first parameter is KEY_MAP_LIST.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

{ **string** / **integer** } := **GET_INFO** *(KEY_MAP_LIST,* { *"current"* / *"first"* / *"last"* / *"next"* / *"previous"* } *)*

PARAMETERS

"current"

Returns a string that is the name of the current key map list. Returns 0 if there is none.

"first"

Returns a string that is the name of the first key map list. Returns 0 if there is none.

"last"

Returns a string that is the name of the last key map list. Returns 0 if there is none.

"next"

Returns a string that is the name of the next key map list. Returns 0 if there is none. Use string constants *"current"* or *"first"* before using *"next."*

"previous"

Returns a string that is the name of the previous key map list. Returns 0 if there is none. Use *"current"* or *"last"* before using *"previous."*

GET_INFO (marker_variable)

Returns information about a specified marker.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ buffer  }
{ integer }:= GET_INFO (marker_variable,
{ keyword }
        { "before_bol"                  }
        { "beyond_eob"                  }
        { "beyond_eol"                  }
        { "bound"                       }
        { "buffer"                      }
        { "display_value"               }
        { "left_margin"                 }
        { "middle_of_tab"               }
        { "offset"                      }
        { "offset_column"               }
        { "record_number"               }
        { "right_margin"                }
        { "unmodifiable_records"        }
        { "video"                       }
        { "within_range", range         }
```

PARAMETERS

"before_bol"

Returns 1 if the specified marker is located before the beginning of a line; returns 0 if it is not.

"beyond_eob"

Returns 1 if the specified marker is located beyond the end of a buffer; returns 0 if it is not.

"beyond_eol"

Returns 1 if the specified marker is located beyond the end of a line; returns 0 if it is not.

"bound"

Returns 1 if the specified marker is attached to a character; returns 0 if the marker is free. For more information on bound and free markers, see Section 2.6.

"buffer"

Returns the buffer in which the marker is located.

"display_value"

Returns the display value of the record in which the specified marker is located. For more information about display values, see the descriptions of the SET (DISPLAY_VALUE) and SET (RECORD_ATTRIBUTES) built-in procedures.

"left_margin"

Returns an integer that is the current left margin setting of the line containing the marker.

"middle_of_tab"

Returns an integer (1 or 0) that indicates whether the marker is located in the white space created by a tab.

"offset"

Returns an integer that is the number of characters between the left margin and the marker. The left margin is counted as character 0. A tab is counted as one character, regardless of width. Window shifts have no effect on the value returned when you use *"offset."* The value returned has no relation to the visible screen column in which the character bound to the marker is displayed.

"offset_column"

Returns an integer that is the screen column in which VAXTPU displays the character to which the marker is bound. When calculating this value, VAXTPU does not take window shifts into account; VAXTPU assumes that any window mapped to the current buffer is not shifted. The value returned when you use *"offset_column"* does reflect the location of the left margin and the width of tabs preceding the editing point. In contrast, the value returned when you use *"offset"* is not affected by the location of the left margin or the width of tabs.

"record_number"

Returns an integer that is the number associated with the record (line) containing the specified marker.

A record number indicates the location of a record in a buffer. Record numbers are dynamic; as you add or delete records, VAXTPU changes the number associated with a particular record, as appropriate. VAXTPU counts each record in a buffer, regardless of whether the line is visible in a window or whether the record contains text. Note that GET_INFO (marker, "record_number") counts the end-of-buffer text as a record if the specified marker is on the end-of-buffer text, but GET_INFO (buffer, "record_count") never counts the end-of-buffer text as a record. Thus, it is possible for the value returned by GET_INFO (buffer, "record_count") to be one less than the maximum value returned by GET_INFO (marker, "record_number").

"right_margin"

Returns an integer that is the current right margin setting of the line containing the marker.

"unmodifiable_records"

Returns 1 if the record containing the specified marker is unmodifiable. The call returns 0 if the record is modifiable.

"video"

Returns a keyword that is the video attribute of the marker. Returns 0 if the marker is a free marker.

"within_range"

Returns an integer (1 or 0) that indicates whether the marker is in the range specified by the third parameter.

GET_INFO (mouse_event_keyword)

Returns information about a mouse event. A *mouse_event_keyword* is a keyword representing a single click, multiple click, upstroke, downstroke, or drag of a mouse button. For a list of the valid mouse event keywords that you can use for the first parameter, see Table 7–3.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

{ **integer** / **window** } **:= GET_INFO** *(mouse_event_keyword,* { *"mouse_button"* / *"window"* } *)*

PARAMETERS

"mouse_button"

Returns an integer that is the number of the mouse button specified with a mouse event keyword.

Table 7–3 lists the valid keywords for the first parameter when you use *"mouse_button"* as the second parameter.

Table 7–3 VAXTPU Keywords Representing Mouse Events

M1UP	M2UP	M3UP	M4UP	M5UP
M1DOWN	M2DOWN	M3DOWN	M4DOWN	M5DOWN
M1DRAG	M2DRAG	M3DRAG	M4DRAG	M5DRAG
M1CLICK	M2CLICK	M3CLICK	M4CLICK	M5CLICK
M1CLICK2	M2CLICK2	M3CLICK2	M4CLICK2	M5CLICK2
M1CLICK3	M2CLICK3	M3CLICK3	M4CLICK3	M5CLICK3
M1CLICK4	M2CLICK4	M3CLICK4	M4CLICK4	M5CLICK4
M1CLICK5	M2CLICK5	M3CLICK5	M4CLICK5	M5CLICK5

"window"

Returns the window in which the down stroke occurred that started the current drag operation. Returns 0 if no drag operation is in progress for the specified mouse button when the built-in is executed.

The valid keywords for the first parameter when you use *"window"* as the second parameter are M1DOWN, M2DOWN, M3DOWN, M4DOWN, and M5DOWN.

EXAMPLES

1
```
x := GET_INFO (M3CLICK2, "mouse_button");
```

This statement causes VAXTPU to assign the value 3 to the variable *x*.

2
```
the_key := READ_KEY;
IF GET_INFO (the_key, "mouse_button") = 3
THEN
    MESSAGE ("MB3 has no effect in this context.");
```

These statements test whether you have pressed MB3 and, if so, display a message in the message window.

3
```
PROCEDURE sample_m1_drag

LOCAL the_window,
      new_window,
      column,
      row,
      temp;

the_window := GET_INFO (M1DOWN, "window");
IF the_window = 0
THEN
    RETURN (FALSE)
ENDIF;

LOCATE_MOUSE (new_window, column, row);

IF the_window <> new_window
THEN
    MESSAGE ("Invalid drag of pointer across window boundaries.");
ENDIF;
ENDPROCEDURE;
```

This procedure, when bound to M1DRAG, responds to a drag event by checking whether you have dragged the mouse across window boundaries. If you have, the procedure displays a message. If not, the procedure creates a select range.

GET_INFO (PROCEDURES)

Returns information about a specified procedure. GET_INFO (PROCEDURES) takes a string as a third parameter. The string specifies the name of the procedure about which you are requesting information.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

integer := GET_INFO *(PROCEDURES,* { *"defined"* | *"minimum_parameters"* | *"maximum_parameters"* }, *string)*

PARAMETERS

"defined"

Returns an integer (1 or 0) that indicates whether the specified procedure is user defined.

"minimum_parameters"

Returns an integer that is the minimum number of parameters required for the specified user-defined procedure.

"maximum_parameters"

Returns an integer that is the maximum number of parameters required for the specified user-defined procedure.

string

A string that is the name of the procedure about which you want information.

GET_INFO (PROCESS)

Returns a specified process in VAXTPU's internal list of processes.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

process := GET_INFO *(PROCESS,* { *"current"* | *"first"* | *"last"* | *"next"* | *"previous"* } *)*

PARAMETERS

"current"

Returns the current process in VAXTPU's internal list of processes. You can only use GET_INFO (PROCESS, "current") after you have used GET_INFO (PROCESS, "first") or GET_INFO (PROCESS, "last"). The built-in returns 0 if you do not use these GET_INFO built-ins in the correct order.

"first"

Returns the first process in VAXTPU's internal list of processes. Returns 0 if there is none.

"last"

Returns the last process in VAXTPU's internal list of processes. Returns 0 if there is none.

"next"

Returns the next process in VAXTPU's internal list of processes. Returns 0 if there are no more processes. Use *"first"* before using *"next"*.

"previous"

Returns the preceding process in VAXTPU's internal list of processes. Returns 0 if there is no previous process. Use *"last"* before using *"previous"*.

GET_INFO (process_variable)

Returns information about a specified process.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

{ **buffer** / **integer** } **:= GET_INFO** *(process_variable,* { *"buffer"* / *"pid"* } *)*

PARAMETERS

"buffer"

Returns the buffer associated with the process.

"pid"

Returns an integer that is the process identification number.

GET_INFO (range_variable)

Returns information about a specified range.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ buffer  }
{ keyword } := GET_INFO (range_variable,
                         { "buffer"              }
                         { "unmodifiable_records" })
                         { "video"               }
```

PARAMETERS

"buffer"

Returns the buffer in which the range is located.

"unmodifiable_records"

Returns 1 if the specified range contains one or more unmodifiable records. The call returns 0 if no unmodifiable records are present in the specified range.

"video"

Returns a keyword that is the video attribute of the range.

GET_INFO (SCREEN)

Returns information about the screen.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ array
  integer
  keyword
  learn_sequence
  PRIMARY
  program
  SECONDARY
  selection_name
  string }
:= GET_INFO (SCREEN,
{ "active_area"
  "ansi_crt"
  "auto_repeat"
  "avo"
  "client_message"
  "client_message_routine"
  "cross_window_bounds"
  "current_column"
  "current_row"
  "dec_crt"
  "dec_crt2"
  "decwindows"
  "detached_action"
  "detached_reason"
  "edit_mode"
  "eightbit"
  "event", GLOBAL_SELECT
  "global_select", { PRIMARY
                     SECONDARY
                     selection_name }
  "grab_routine", { GLOBAL_SELECT
                    INPUT_FOCUS }
  "icon_name"
  "input_focus"
  "length"
  "line_editing" }
```

```
{ "mouse"
  "new_length"
  "new_width"
  "old_length"
  "old_width"
  "original_length"
  "original_width"
  "prompt_length"
  "prompt_row"
  "read_routine", GLOBAL_SELECT
  "screen_limits"
  "screen_update"
  "scroll"
  "time", GLOBAL_SELECT
  "ungrab_routine", { GLOBAL_SELECT }
                    { INPUT_FOCUS   }
  "visible_length"
  "vk100"
  "vt100"
  "vt200"
  "vt300"
  "width" }
```

PARAMETERS

"active_area"

Returns an array containing information on the location and dimensions of the application's active area. Returns the integer 0 if there is no active area. The active area is the region in a window in which VAXTPU ignores movements of the pointer cursor for purposes of distinguishing clicks from drags. When you press down a mouse button, VAXTPU interprets the event as a click if the upstroke occurs in the active area with the downstroke. If the upstroke occurs outside the active area, VAXTPU interprets the event as a drag operation.

A VAXTPU layered application can have only one active area at a time, even if the application has more than one window visible on the screen. An active area is only valid if you are pressing a mouse button. The default active area occupies one character cell. By default, the active area is located on the character cell pointed to by the pointer cursor.

For information on mouse button clicks, which are related to the concept of an active area, see the *XUI Style Guide*.

GET_INFO (SCREEN, "active_area") returns five pieces of information about the active area in integer-indexed elements of the returned array. You need not use the CREATE_ARRAY built-in before using GET_INFO (SCREEN, "active_area"); VAXTPU assigns a properly structured array to the return variable you specify. The structure of the array is as follows:

Array Element	Contents
array {1}	The window containing the active area
array {2}	The column forming the leftmost edge of the active area

Array Element	Contents
array {3}	The row forming the top edge of the active area
array {4}	The width of the active area, expressed in columns
array {5}	The height of the active area, expressed in rows

"ansi_crt"

Returns an integer (1 or 0) that indicates whether the terminal is an ANSI_CRT.

"auto_repeat"

Returns an integer (1 or 0) that indicates whether the terminal's autorepeat feature is on.

"avo"

Returns an integer (1 or 0) that indicates whether the ADVANCED_VIDEO attribute has been set for the terminal.

"client_message"

Returns a keyword indicating whether VAXTPU has received a KILL_SELECTION client message or a STUFF_SELECTION client message. If the call is used when there is no current client message, the integer 0 is returned.

GET_INFO (SCREEN, "client_message") is used in a VAXTPU-layered or EVE-layered application's client message routine. This routine provides the application's response to a client message received from another application.

GET_INFO (SCREEN, "client_message") returns the keyword KILL_SELECTION when the user is copying from an application layered on VAXTPU or on EVE that owns the input focus to another application. To do so, the user selects text in the VAXTPU/EVE-layered application. This designates the text to be placed in the primary global selection when another application asks to read the selection. Next, the user clicks the MB3 button in the other application. This causes the text in the primary global selection to be copied at the location indicated by the pointer when the user clicked on MB3. If the user uses CTRL/MB3 to copy the selection into the other application, this means that after the selection is copied into the other application, it is deleted from the VAXTPU/EVE-layered application. In this case, after the other application inserts the text from the primary global selection, that application sends a KILL_SELECTION client message to the VAXTPU/EVE-layered application. When the VAXTPU/EVE-layered application detects that a client message has been received, it executes its client message routine. This routine contains a statement using GET_INFO (SCREEN, "client_message"). In the case described here, the return value is the keyword KILL_SELECTION. The VAXTPU/EVE-layered application then deletes the selected text.

GET_INFO (SCREEN, "client_message") returns the keyword STUFF_SELECTION when the user is copying from some application into the VAXTPU/EVE-layered application that owns the input focus. The user performs a drag operation using the MB3 button to select the text in the other application. The application grabs ownership of the secondary global selection and assigns to it the selected text. The application then sends a STUFF_SELECTION client message to the VAXTPU/EVE-layered application. When the VAXTPU/EVE-layered application detects that a

client message has been received, it executes its client message routine. This routine contains a statement using GET_INFO (SCREEN, "client_ message"). In the case described here, the return value is the keyword STUFF_SELECTION. The VAXTPU/EVE-layered application then inserts the text from the secondary global selection at the VAXTPU/EVE-layered application's editing point.

"client_message_routine"

Returns the program or learn sequence designated as an application's client message action routine. Returns 0 if none is designated.

"cross_window_bounds"

Returns an integer (1 or 0) that indicates whether the CURSOR_ VERTICAL built-in causes the cursor to cross a window boundary if the cursor is at the top or bottom of the window.

"current_column"

Returns an integer that is the number of the current column.

"current_row"

Returns an integer that is the number of the current row.

"dec_crt"

Returns an integer (1 or 0) that indicates whether the terminal is a DEC_ CRT. For more information on this terminal characteristic, see the SET TERMINAL command in the *VMS DCL Dictionary*.

"dec_crt2"

Returns an integer (1 or 0) that indicates whether the terminal is a DEC_ CRT2. For more information on this terminal characteristic, see the SET TERMINAL command in the *VMS DCL Dictionary*.

"decwindows"

Returns 1 if your system is running the DECwindows version of VAXTPU, otherwise returns 0. For more information about the DECwindows version of VAXTPU, see Chapter 1.

"detached_action"

Returns the current detached action routine. If no such routine is designated, returns the type UNSPECIFIED.

"detached_reason"

Returns a bit-encoded integer indicating which of the five possible detached states the cursor is in.

Digital recommends that you use the VAXTPU predefined constants rather than the actual integers to refer to the reasons for detachment. Table 7–4 shows the correspondence of constants, integers, and reasons.

Table 7–4 Detached Cursor Flag Constants

Constant	Value	Reason
TPU$K_OFF_LEFT	1	The editing point is off the left side of the current window.
TPU$K_OFF_RIGHT	2	The editing point is off the right side of the current window.
TPU$K_INVISIBLE	4	The editing point is on a record that is invisible in the current window.
TPU$K_DISJOINT	8	The current buffer is not mapped to the current window.
TPU$K_UNMAPPED	16	No current window exists.

Note that it is possible for TPU$K_INVISIBLE to be set in combination with either the TPU$K_OFF_LEFT or TPU$K_OFF_RIGHT flags.

"edit_mode"

Returns an integer (1 or 0) that indicates whether the terminal is set to edit mode.

"eightbit"

Returns an integer (1 or 0) that indicates whether the terminal uses 8-bit characters.

"event"

This parameter is used with DECwindows only.

When you use *"event"* as the second parameter, you must specify the keyword GLOBAL_SELECT as the third parameter. GLOBAL_SELECT indicates that GET_INFO is to supply information about a global selection.

If called from within a global selection grab or ungrab routine, GET_INFO (SCREEN, "event", GLOBAL_SELECT) identifies the global selection that was grabbed or lost. GET_INFO (SCREEN, "event", GLOBAL_SELECT) returns a keyword if the global selection was the primary or secondary selection. The built-in returns a string naming the global selection if the grab or ungrab involves a global selection other than the primary or secondary selection.

If called from within a routine that responds to requests for information about a global selection, GET_INFO (SCREEN, "event", GLOBAL_SELECT) returns an array. The array contains the information an application needs to respond to the request for information about the global selection. The array contains the following information:

array {1}	The keyword PRIMARY, the keyword SECONDARY, or a string. This element identifies the global selection about which information was requested.
array {2}	A string. This element identifies the global selection property about which information has been requested.

The GET_INFO (SCREEN, "event") built-in returns 0 if the built-in is not responding to a grab, an ungrab, or a selection information request.

For more information about grabbing and ungrabbing a global selection, see the *VMS DECwindows Guide to Application Programming*.

"global_select"

This parameter is used with DECwindows only.

Returns the integer 1 if VAXTPU currently owns the specified global selection; 0 if it does not.

You must specify one of the following parameters as a third parameter to GET_INFO (SCREEN, "global_select"):

PRIMARY	A keyword directing VAXTPU to get information on the primary global selection.
SECONDARY	A keyword directing VAXTPU to get information on the secondary global selection.
selection_name	A string identifying the global selection about which VAXTPU is to get information.

For more information about grabbing and ungrabbing a global selection, see the *VMS DECwindows Guide to Application Programming*.

"grab_routine"

This parameter is used with DECwindows only.

Returns the program or learn sequence designated as the application's global selection or input focus grab routine. Returns the integer 0 if the requested grab routine is not present.

You must specify one of the following keywords as a third parameter to GET_INFO (SCREEN, "grab_routine"):

GLOBAL_SELECT	A keyword indicating that GET_INFO is to return the global selection grab routine.
INPUT_FOCUS	A keyword indicating that GET_INFO is to return the input focus grab routine.

"icon_name"

This parameter is used with DECwindows only.

Returns the string used as the layered application's name in the DECwindows icon box.

"input_focus"

This parameter is used with DECwindows only.

Returns an integer (1 or 0) indicating whether VAXTPU currently owns the input focus. Input focus is the ability to process user input from the keyboard.

"length"

Returns an integer that is the current length of the screen (in rows).

"line_editing"

Returns an integer (1 or 0) indicating whether the line-editing terminal attribute is turned on. On a character-cell terminal, returns 1 if the line-editing terminal attribute is turned on, otherwise returns 0. In DECwindows VAXTPU, this parameter always returns 0.

"mouse"

Returns an integer (1 or 0) that indicates whether VAXTPU's mouse support capability is turned on.

"new_length"

This parameter is used with DECwindows only.

Returns an integer that is the length (in rows) of the screen after the resize action routine is executed.

Resize action routines should use the length returned by GET_INFO (SCREEN, "new_length") to determine the length of their windows. If it is used outside a resize action routine, this length is the same as the current length of the screen.

"new_width"

This parameter is used with DECwindows only.

Returns an integer that is the width (in columns) of the screen after the resize action routine is executed.

Resize action routines should use the length returned by GET_INFO (SCREEN, "new_width") to determine the width of their windows. If it is used outside a resize action routine, this width is the same as the current width of the screen.

"old_length"

This parameter is used with DECwindows only.

Returns an integer that is the length (in rows) of the screen before the most recent resize event.

The *"old_length"* value is initially set to the length of the screen at startup. This value is reset after VAXTPU processes a resize event and before VAXTPU executes the resize action routine.

"old_width"

This parameter is used with DECwindows only.

Returns the width (in columns) of the screen before the most recent resize event.

The *"old_width"* value is initially set to the width of the screen at startup. This value is reset after VAXTPU processes a resize event and before VAXTPU executes the resize action routine.

"original_length"

Returns an integer that is the number of lines the screen had when VAXTPU was invoked.

"original_width"

Returns an integer that is the width of the screen when VAXTPU was invoked.

"prompt_length"

Returns an integer that is the number of lines in the prompt area.

"prompt_row"

Returns an integer that is the screen line number at which the prompt area begins.

"read_routine"

This parameter is used with DECwindows only.

Returns the program or learn sequence that VAXTPU executes when it owns a global selection and another application has requested information about that selection. If the application has not specified a global selection read routine, 0 is returned.

You must specify the keyword GLOBAL_SELECT as the third parameter to GET_INFO (SCREEN, "read_routine"). GLOBAL_SELECT indicates that GET_INFO is to return the global selection read routine.

"screen_limits"

Returns an integer-indexed array specifying the minimum and maximum screen length and width.

An integer-indexed array uses four elements to specify the minimum and maximum screen width and length. The array indices and the contents of their corresponding elements are as follows.

Array Element	Contents
array {1}	The minimum screen width, in columns. This value must be at least 0 and less than or equal to the maximum screen width. The default value is 0.
array {2}	The minimum screen length, in lines. This value must be at least 0 and less than or equal to the maximum screen length. The default value is 0.
array {3}	The maximum screen width, in columns. This value must be greater than or equal to the minimum screen width and less than or equal to 255. The default value is 255.
array {4}	The maximum screen length, in lines. This value must be greater than or equal to the minimum screen length and less than or equal to 255. The default value is 255.

"screen_update"

Returns an integer (1 or 0) that indicates whether screen updating is turned on.

"scroll"

Returns an integer (1 or 0) that indicates whether the terminal has scrolling regions. For more information on scrolling regions, see the description of the built-in SET (SCROLLING).

"time"

This parameter is used with DECwindows only.

Returns a string in VMS delta time format indicating the amount of time after requesting global selection information that VAXTPU waits for a reply. When the time has expired, VAXTPU assumes the request will not be answered.

You must specify the keyword GLOBAL_SELECT as the third parameter to GET_INFO (SCREEN, "time").

"ungrab_routine"

This parameter is used with DECwindows only.

Returns the program or learn sequence that VAXTPU executes when it loses ownership of a global selection or of the input focus. Returns 0 if the requested ungrab routine is not present.

You must specify one of the following keywords as a third parameter to GET_INFO (SCREEN, "ungrab_routine"):

GLOBAL_SELECT	A keyword indicating that GET_INFO is to return the global selection ungrab routine
INPUT_FOCUS	A keyword indicating that GET_INFO is to return the input focus ungrab routine

"visible_length"

Returns an integer that is the page length of the terminal.

"vk100"

Returns an integer (1 or 0) that indicates whether the terminal is a GIGI.™

"vt100"

Returns an integer (1 or 0) that indicates whether the terminal is in the VT100 series.

"vt200"

Returns an integer (1 or 0) that indicates whether the terminal is in the VT200 series.

"vt300"

Returns an integer (1 or 0) that indicates whether the terminal is in the VT300 series.

"width"

Returns an integer that is the current physical width of the screen.

GET_INFO (string_variable)

Returns information about the specified string. The string must be the name of a keymap or keymap list.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ integer  }
{ keyword  } := GET_INFO (string_variable,
{ program  }
                          { "journal"             }
                          { "pre_key_procedure"   }
                          { "post_key_procedure"  }
                          { "self_insert"         }
                          { "shift_key"           }
                          { "undefined_key"       } )
```

PARAMETERS

"journal"

Returns an array containing information about the buffer change journal file whose name you specify with the string parameter. If the specified file is not a journal file, the integer 0 is returned.

The array indices and the contents of the corresponding elements of the returned array are as follows:

Index	Contents of Element
1	The name of the buffer whose contents were journaled.
2	The date and time the journal file was created.
3	The date and time the edit session started.
4	The name of the source file. A source file is a file to which the buffer has been written. The journal file maintains a pointer to the source file. This enables the journal file to retrieve from the source file the buffer contents as they were after the last write operation. If the buffer has not been written out or if none of the source files will be available during recovery, this element contains a null string.
5	The name of the output file associated with the buffer. This is the file name specified with the SET (OUTPUT) built-in.
6	The name of the original input file associated with the buffer by the CREATE_BUFFER built-in. If there is no associated input file or if the input file will not be available during a recovery, this element contains a null string.
7	The identification string for the version of VAXTPU that wrote the journal file.

Note that all elements are of type string.

"pre_key_procedure"
Returns the program, stored in the specified keymap or keymap list, that is called before execution of code bound to keys. Returns 0 if no procedure was defined by SET (PRE_KEY_PROCEDURE).

"post_key_procedure"
Returns the program, stored in the specified keymap or keymap list, that is called before execution of code bound to keys. Returns 0 if no procedure was defined by SET (POST_KEY_PROCEDURE).

"self_insert"
Returns an integer (1 or 0) that indicates whether printable characters are to be inserted into the buffer if they are not defined. This parameter is established or changed with the built-in procedure SET (SELF_INSERT).

"shift_key"
Returns a keyword that is the key name for the key currently used as the shift key. This parameter is established or changed with the built-in procedure SET (SHIFT_KEY).

"undefined_key"
Returns the program that is called when an undefined character is entered. Returns 0 if the program issues the default message. This parameter is established or changed with the built-in procedure SET (UNDEFINED_KEY).

GET_INFO (SYSTEM)

Returns information about the system.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ integer          }
{ keyword          }
{ learn_sequence   } := GET_INFO
{ program          }
{ string           }
```

```
(SYSTEM, { "bell"                    }
         { "column_move_vertical"    }
         { "display"                 }
         { "default_directory"       }
         { "enable_resize"           }
         { "facility_name"           }
         { "informational"           }
         { "journaling_frequency"    }
         { "journal_file"            }
         { "line_number"             }
         { "message_action_level"    }
         { "message_action_type"     }
         { "message_flags"           }
         { "pad_overstruck_tabs"     }
         { "recover"                 }
         { "resize_action"           }
         { "section_file"            }
         { "shift_key"               }
         { "success"                 }
         { "timed_message"           }
         { "timer"                   }
         { "traceback"               }
         { "update"                  }
         { "version"                 }
         { "work_file"               }
```

PARAMETERS

"bell"

Returns the keyword ALL if the bell is on for all messages. Returns the keyword BROADCAST if the bell is on for broadcast messages only. Returns 0 if the SET (BELL) feature is off. This parameter is established or changed with the built-in procedure SET.

"column_move_vertical"

Returns 1 if the MOVE_VERTICAL built-in is set to keep the cursor in the same column as the cursor moves from line to line. Returns 0 if the MOVE_VERTICAL built-in preserves the offset, rather than the column

position, from line to line. This parameter is established or changed with the built-in procedure SET (COLUMN_MOVE_VERTICAL).

"display"

Returns 1 if the /DISPLAY qualifier has been specified by the user or by default; otherwise, returns 0.

"default_directory"

Returns the name of the current default directory.

"enable_resize"

Returns 1 if resize operations are enabled, otherwise returns 0. By default, resize operations are not enabled. You can turn resizing on or off with the built-in SET (ENABLE_RESIZE).

"facility_name"

Returns a string that is the current facility name. This parameter is established or changed with the built-in procedure SET (FACILITY_NAME).

"informational"

Returns an integer (1 or 0) that indicates whether informational messages are displayed. This parameter is established or changed with the built-in procedure SET (INFORMATIONAL).

"journaling_frequency"

Returns an integer that indicates how frequently records are written to the journal file. This parameter is established or changed with the built-in procedure SET (JOURNALING).

"journal_file"

Returns a string that is the name of the journal file.

"line_number"

Returns an integer (1 or 0) that indicates whether VAXTPU displays the line number at which an error occurred. This parameter is established or changed with the built-in procedure SET (LINE_NUMBER).

"message_action_level"

Returns an integer that is the completion status severity level at which VAXTPU performs the message action you specify. The valid values, in ascending order of severity, are as follows: 1 (success), 3 (informational), 0 (warning), and 2 (error). This parameter is established or changed with the built-in procedure SET (MESSAGE_ACTION_LEVEL).

"message_action_type"

Returns a keyword describing the action to be taken when VAXTPU signals an error, warning, or message whose severity level is greater than or equal to the level set with SET (MESSAGE_ACTION_LEVEL). The possible keywords are NONE, BELL, and REVERSE. This parameter is established or changed with the built-in procedure SET (MESSAGE_ACTION_TYPE).

"message_flags"
Returns an integer that is the current value of the message flag setting. This parameter is established or changed with the built-in procedure SET (MESSAGE_FLAGS).

"pad_overstruck_tabs"
Returns an integer (1 or 0) that indicates whether VAXTPU preserves the white space created by a tab character. This parameter is established or changed with the built-in procedure SET (PAD_OVERSTRUCK_TABS).

"recover"
Returns an integer (1 or 0) that indicates whether a recovery using a keystroke journal file is currently in progress. Be careful when using this built-in—specifying different VAXTPU actions during a recovery than during an ordinary editing session may cause VAXTPU keystroke journaling to fail.

"resize_action"
Returns the program or learn sequence designated as the application's resize action routine. Returns 0 if the requested resize action routine is not present. You can designate a resize action routine using the SET (RESIZE_ACTION) built-in.

"section_file"
Returns a string that is the name of the section file used when the user invoked VAXTPU.

"shift_key"
Returns a keyword that is the value of the current shift key set with SET (SHIFT_KEY) for the current buffer.

"success"
Returns an integer (1 or 0) that indicates whether success messages are displayed. This parameter is established or changed with the built-in procedure SET (SUCCESS).

"timed_message"
Returns a string of text that VAXTPU displays at 1-second intervals in the prompt area if the SET (TIMER) feature is on.

"timer"
Returns the integer 1 if SET (TIMER) has been enabled, otherwise returns 0.

"traceback"
Returns an integer (1 or 0) that indicates whether VAXTPU displays the call stack for VAXTPU procedures when an error occurs. This parameter is established or changed with the built-in procedure SET (TRACEBACK).

"update"
Returns an integer that is the update number of this version of VAXTPU.

"version"
Returns an integer that is the version number of VAXTPU.

"work_file"
Returns a string that is the name of the work file opened during startup.

GET_INFO (WIDGET)

Returns information about VAXTPU widgets in general or about a specific widget whose name you do not know at the time you use the built-in.

The GET_INFO (WIDGET) built-in is used with DECwindows only.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ integer }
{ widget  } := GET_INFO (WIDGET,
{ array   }
{ NONE    }
        { "callback_parameters", array                          }
        { "children", { widget } , array                        }
        { "menu_position", mouse_down_button                    }
        { "widget_id", { parent_widget                          } )
        {              { SCREEN, widget_name                    }
```

PARAMETERS

"callback_parameters"

Returns the widget instance performing the callback, the closure value associated with the widget instance, and the reason for the callback. Note that in DECwindows documentation, the closure is called the *tag*.

array

An array used to return values for the callback, the closure, and the reason. The array has the following indices of type string: *"widget"*, *"closure"*, and *"reason_code"*. GET_INFO (WIDGET, "callback_parameters") places the corresponding values in the array elements. VAXTPU automatically creates the array in which the return values are placed.

To use this parameter, specify a variable that has been declared or initialized before you use it. The initial type and value of the variable are unimportant. When GET_INFO (WIDGET, "callback_parameters") places the return values in the array, the initial values are lost.

Note that the integer on the left side of the assignment operator indicates whether GET_INFO was used correctly.

GET_INFO (WIDGET, "callback_parameters") should be used in a widget callback procedure. If you use this built-in outside a widget callback procedure, the value returned is indeterminate. If you use the built-in inside a widget callback procedure and callback information is available, the built-in returns 1.

For more information about callbacks and closure values in DECwindows VAXTPU, see Chapter 4. For general information about using callbacks and closure values, see the *VMS DECwindows Guide to Application Programming*.

"children"

Returns the number of widget children controlled by the specified widget. The array parameter returns the children themselves. If the keyword SCREEN is specified instead of a widget, the built-in returns the number of children controlled by the VAXTPU main window widget.

"menu_position"

Returns information about any pop-up widgets that are set for menu positioning when you press the specified mouse button. If no pop-up widgets are set, returns the keyword NONE; otherwise, returns an integer-indexed array of all pop-ups set for menu positioning.

mouse_down_button	This keyword (M1DOWN, M2DOWN, M3DOWN, M4DOWN, or M5DOWN) indicates the mouse button associated with the pop-up menus.

"widget_id"

Returns the widget instance whose name matches the specified widget name. The remaining parameters are as follows:

parent_widget	The widget that is an ancestor of the widget instance returned by the GET_INFO (WIDGET) built-in.
SCREEN	A keyword indicating that VAXTPU's main window widget is the ancestor of the widget instance that you want the GET_INFO (WIDGET) built-in to return.
widget_name	A string that is the fully qualified name of the widget you want the built-in to return. To specify this parameter correctly, start the string with the name of the widget's parent. Use the same name you used to specify the *parent_widget* parameter. If you used the SCREEN parameter instead of the *parent_widget* parameter, start the string with the widget name *tpu$mainwindow*. Next, specify the names of the ancestors, if any, that occur in the widget hierarchy between the parent and the widget itself. Start with the ancestor just below the parent and progressively specify more immediate ancestors. Finally, specify the name of the widget you want the GET_INFO (WIDGET) built-in to return. Separate all widget names with periods. The fully qualified widget name is case sensitive.

GET_INFO (WIDGET, "widget_id") calls the X Toolkit routine NAME TO WIDGET.

For more information on DECwindows concepts such as parent widgets, ancestor widgets, and the distinction between widget classes and widget instances, see the *VMS DECwindows Guide to Application Programming*.

EXAMPLES

1

```
PROCEDURE eve$callback_dispatch

LOCAL   the_program,
        status,
        temp_array;

ON_ERROR
    [TPU$_CONTROLC]:
        eve$$x_state_array {eve$$k_command_line_flag} := eve$k_invoked_by_key;
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
        eve$$x_state_array {eve$$k_command_line_flag} := eve$k_invoked_by_key;
ENDON_ERROR

IF NOT eve$x_decwindows_active
THEN
    RETURN (FALSE);
ENDIF;

eve$$x_state_array {eve$$k_command_line_flag} := eve$k_invoked_by_menu;


status :=
     GET_INFO (WIDGET, "callback_parameters", temp_array); ! This statement using
                                                           ! GET_INFO (WIDGET)
                                                           ! returns the calling
                                                           ! widget, the closure,
                                                           ! and the reason code.


! The following statements make the contents of "temp_array"
! available to all the eve$$widget_xxx procedures

eve$x_widget := temp_array {"widget"};
                ! This array element contains the widget
                ! that called back.
eve$x_widget_tag := temp_array {"closure"};
                ! This array element contains the widget tag
                ! that is assigned to the widget in the UIL file.
eve$x_widget_reason := temp_array {"reason_code"};
                ! This array element contains callback reason code.

! The next line gets the callback routine from the array indexed
! by closure values.

the_program := eve$$x_widget_array {eve$x_widget_tag};

IF the_program <> 0
THEN
    EXECUTE (the_program);
ENDIF;

eve$$x_state_array {eve$$k_command_line_flag} := eve$k_invoked_by_key;
RETURN;

ENDPROCEDURE;
```

This procedure shows one possible way that a layered application can use GET_INFO (WIDGET, "callback_parameters", array). The procedure is a simplified version of the EVE procedure EVE$CALLBACK_DISPATCH. You can find the original version in SYS$EXAMPLES:EVE$MENUS.TPU.

(For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.)

This version of EVE$CALLBACK_DISPATCH handles callbacks from EVE widgets. The statement GET_INFO (WIDGET, "callback_parameters", temp_array) copies the following three items into elements of the array *temp_array*:

- The widget that is calling back
- The widget's integer closure
- The reason why the widget is calling back

The array *eve$$x_widget_array* contains pointers to all of EVE's callback routines in elements indexed by the appropriate integer closure values. This procedure locates the correct index in the array and executes the corresponding callback routine.

Warning: This simplified version of EVE$CALLBACK_DISPATCH does not completely replace the version in existing EVE code. Furthermore, Digital does not guarantee that this example will work successfully with future versions of EVE. This example is presented solely to illustrate how EVE uses the built-in GET_INFO (WIDGET, "callback_parameters", array) in a callback handling procedure.

2
```
the_text_widget := GET_INFO (WIDGET, "widget_id", new_dialog,
                             "NEW_DIALOG.NEW_TEXT");
```

This statement assigns to the variable *the_text_widget* the widget instance named by the string *NEW_DIALOG.NEW_TEXT*. The widget instance is the child of the widget instance assigned to the variable *new_dialog*.

3
```
PROCEDURE eve_show_widgets              ! Display the widget hierarchy

local
        loop_index,
        num_topmost,
        widget_array;

widget_array := 0;
num_topmost := GET_INFO (WIDGET, "children", SCREEN, widget_array);

IF num_topmost > 0
THEN
    loop_index := 1;
    LOOP
        EXITIF loop_index > num_topmost;
        show_widget_tree (widget_array, "");
        loop_index := loop_index + 1;
    ENDLOOP;
ENDIF;

ENDPROCEDURE;

PROCEDURE show_widget_tree      ! Recursively display the widget tree
    (the_array, the_string)
```

```
LOCAL
        child_array,
        highest,
        loop_index,
        num_children;

child_array := 0;
loop_index := 1;
highest := get_info (the_array, "high_index");
LOOP
    EXITIF loop_index > highest;
    MESSAGE (the_string + GET_INFO (the_array {loop_index}, "name")
             + ASCII (%o11)
             + GET_INFO (the_array {loop_index}, "class"));
    num_children := GET_INFO (WIDGET, "children",
                              the_array {loop_index}, child_array);
    IF num_children > 0
    THEN
        show_widget_tree (child_array, the_string + "    ");
    ENDIF;
    loop_index := loop_index + 1;
ENDLOOP;

ENDPROCEDURE;
```

This procedure shows how to use GET_INFO (WIDGET, "children") to display the entire hierarchy of widgets known to a VAXTPU session.

GET_INFO (widget_variable)

Returns information about a specified widget variable.

The GET_INFO (widget_variable) built-in is used with DECwindows only.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ integer        }
{ learn_sequence }
{ program        } := GET_INFO (widget_variable,
{ string         }
                 { "callback_routine"                                  }
                 { "class"                                             }
                 { "is_managed"                                        }
                 { "is_subclass", widget_class                         }
                 { "name"                                              }
                 { "parent"                                            }
                 { "resources"                                         }
                 { "text"                                              }
                 { "widget_info", { array                              }
                                  { arg_pair [[, arg_pair... ]]        }
                                                                      }
```

PARAMETERS

"callback_routine"

Returns the program or learn sequence designated as the application's callback routine for the specified widget. This is the program or learn sequence that VAXTPU should execute when a widget callback occurs for the specified widget instance. For more information about callbacks, see Chapter 4.

"class"

Returns the name of the class to which the specified widget instance belongs.

"is_managed"

Returns 1 (TRUE) if the specified widget is managed; otherwise, it returns 0 (FALSE). This built-in calls the DECwindows Toolkit routine IS MANAGED.

"is_subclass"

Returns 1 (TRUE) if the specified widget belongs to the class referred to by the specified integer or belongs to a subclass of that class. A TRUE value indicates only that the widget is equal to or is a subclass of the specified class; the value does not indicate how far down the class hierarchy the widget's class or subclass is. If the widget is not in the class, or one of its subclasses, this GET_INFO call returns 0 (FALSE).

widget_class	The integer specifying the widget class to use in the subclass test. This value is returned from the DEFINE_WIDGET_CLASS built-in procedure.

"name"

Returns a string that is the name of the specified widget instance.

"parent"

Returns the parent of the specified widget instance. If the widget has no parent, the call returns 0.

"resources"

Returns a string-indexed array in which each index is a valid resource name for the specified widget. The corresponding array element is a string containing the resource's data type and class, separated by a line feed (ASCII (10)).

"text"

Returns a string that is the value of the specified simple text widget. (The value of a text widget is the text entered into the text widget by the user in response to a prompt in a dialog box.) GET_INFO (widget_variable, "text") is equivalent to the XUI Toolkit routine *dwt$s_text_get_string*.

If the specified widget is not of class SText, VAXTPU signals the status TPU$_WIDMISMATCH.

"widget_info"

Returns the current values for one or more resources of the specified widget.

Note that the values are returned in the array or series of argument pairs that is passed as the third parameter. The integer on the left side of the assignment operator indicates whether the built-in executed successfully.

The third parameter is either an array or a series of paired arguments, specified as follows:

array	Each array index must be a string naming a valid resource for the specified widget. Note that resource names are case sensitive. The corresponding array element contains the value of the resource. The array can contain any number of elements.
arg_pair	A string naming a valid resource for the widget followed by a variable to store the value of the resource. Separate the resource name string from the variable with a comma and a space, as follows: resource_name_string, resource_value You can fetch as many resources as you want by using multiple pairs of arguments.

GET_INFO (widget_variable, "widget_info", array, arg_pair) is functionally equivalent to the X Toolkit routine GET VALUES.

If you specify the name of a resource that the widget does not support, VAXTPU signals the error TPU$_ARGMISMATCH.

If the requested resource is a list of items and the list contains no entries, the GET_INFO call uses either the element of the array parameter or uses the value parameter to return an array containing no elements.

For more information about specifying resources, see Chapter 4.

EXAMPLES

1
```
EXECUTE (GET_INFO (eve$x_replace_dialog,
                   "callback_routine"));
```

This statement executes the callback routine for the widget *eve$x_replace_dialog*. Note that this statement is valid only after the Replace dialog box has been used at least once, because EVE does not create any dialog box until you have invoked it.

2
```
PROCEDURE sample_return_name

LOCAL status;

status := GET_INFO (eve$x_replace_dialog,
                    "name");

MESSAGE ("The data type of status is: ");
MESSAGE (STR (GET_INFO(status, "type")));
MESSAGE ("The value of status is: ");
MESSAGE (STR (status));

ENDPROCEDURE;
```

This procedure displays the name of the widget instance specified by the variable *eve$x_replace_dialog*. To confirm that the widget has been created as expected, the procedure also displays a message identifying the data type of the variable's contents. Note that the procedure is valid only after the *Replace* dialog box has been used at least once, because EVE does not create any dialog box until you have invoked it.

A statement containing the built-in GET_INFO (widget, "name") can be useful in code implementing a debugging command that evaluates VAXTPU statements, expressions, and variables.

3
```
eve$x_needfilename_dialog := CREATE_WIDGET ("NEEDFILENAME_DIALOG",
                                            eve$k_widget_hierarchy,
                                            SCREEN,
                                            eve$kt_callback_routine);

the_value := "Type filename for writing buffer " +
                          get_info (the_buffer, "name");

child_of_box := get_info (WIDGET, "widget_id",
                          eve$x_needfilename_dialog,
                          "NEEDFILENAME_DIALOG.NEEDFILENAME_LABEL");

status := set (WIDGET, child_of_box, eve$dwt$c_nlabel, the_value);
```

This code fragment creates an EVE file name dialog box widget and assigns the widget to the variable *eve$x_needfilename_dialog*. Next, the fragment assigns to the variable *the_value* a string prompting you for the name of a file to which the buffer's contents should be written. The fragment uses the built-in GET_INFO (WIDGET, "widget_id") to assign

the dialog box's label widget to the variable *child_of_box*. Finally, the fragment assigns to the label widget's *evedwtc_nlabel* resource the string contained in *the_value*.

4
```
PROCEDURE user_widget_replace_all

CONSTANT
      user_k_widget_name := "REPLACE_DIALOG.REPLACE_ALL";

LOCAL the_value,
      parent_widget,
      replace_all_button;

parent_widget := eve$x_replace_dialog;

replace_all_button := GET_INFO (WIDGET, "widget_id",
                                parent_widget,
                                user_k_widget_name);

GET_INFO (replace_all_button,                ! This statement uses
          "widget_info", eve$dwt$c_nvalue,   ! GET_INFO (widget, "widget_info")
          the_value);                        ! to fetch the value of the
                                             ! dwt$c_nvalue resource.

IF the_value
THEN
    MESSAGE ("All instances will be replaced.");
ELSE
    MESSAGE ("Not all instances will be replaced.");
ENDIF;

ENDPROCEDURE;
```

This procedure, *user_widget_replace_all*, shows one possible way that a layered application can use GET_INFO (widget, "widget_info"). The procedure is a modified version of the EVE procedure EVE$$WIDGET_REPLACE_ALL. You can find the current version in SYS$EXAMPLES:EVE$MENUS.TPU. (For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.)

Procedure *user_widget_replace_all* determines what user message to display in response to the EVE command REPLACE. The procedure uses GET_INFO (widget, "widget_info") to fetch the value of the resource *dwt$c_nvalue*. A value of 0 means the *Replace All* toggle button appears unshaded while a value of 1 means the toggle button appears solid.

5
```
temp_array := create array;
temp_array {"selectedItems" + ascii (10) + "selectedItemsCount"} := 0;
status := get_info (the_widget_id, "widget_info", temp_array);
```

If *the_widget_id* is a variable containing a list box widget that has no items selected, then *temp_array{"selectedItems" + ascii (10) + "selectedItemsCount"}* contains an empty array when the built-in returns.

GET_INFO (WINDOW)

Returns a window from VAXTPU's internal list of windows or the current window on the screen.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

window := GET_INFO *(WINDOW[[S]],* { *"current"* | *"first"* | *"last"* | *"next"* | *"previous"* } *)*

PARAMETERS

"current"

Returns the current window on the screen. Returns 0 if there is none. GET_INFO (WINDOW[[S]], "current") always returns the current window, regardless of whether or you have first used GET_INFO (WINDOW[[S]], "first") or GET_INFO (WINDOW[[S]], "last").

"first"

Returns the first window in VAXTPU's internal list of windows. Returns 0 if there is none.

"last"

Returns the last window in VAXTPU's internal list of windows. Returns 0 if there is none.

"next"

Returns the next window in VAXTPU's internal list of windows. Returns 0 if there are no more windows in the list. Use string constants *"current"* or *"first"* before using *"next"*.

"previous"

Returns the preceding window in VAXTPU's internal list of windows. Returns 0 if there are no previous windows in the list. Use string constants *"current"* or *"last"* before using *"previous"*.

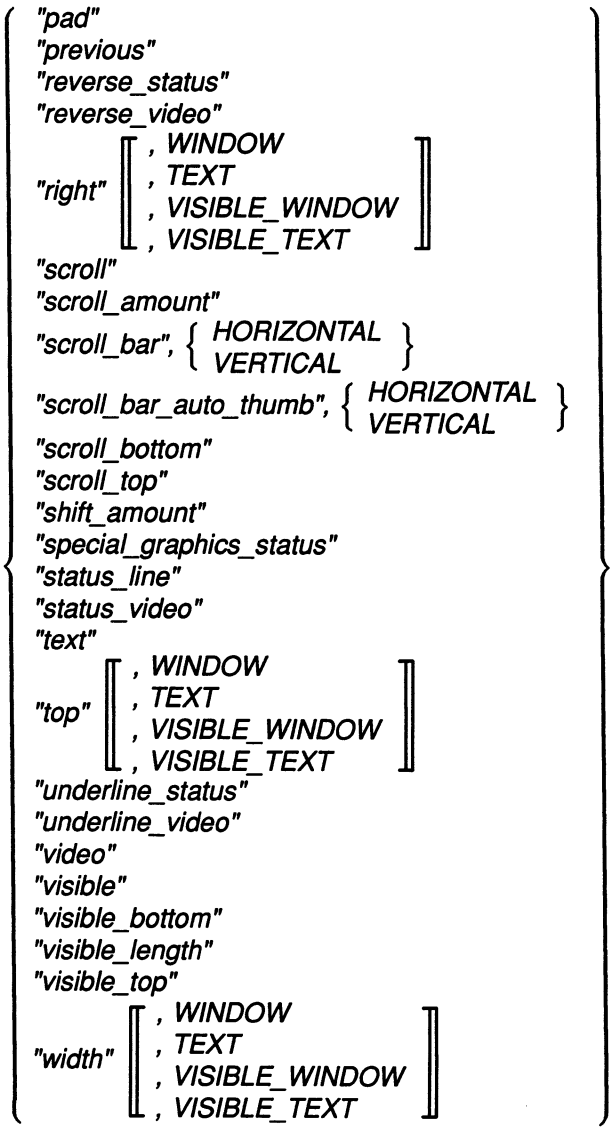
GET_INFO (window_variable)

Returns information about a specified window.

For general information about using all forms of GET_INFO built-ins, see the description of GET_INFO.

FORMAT

```
{ integer  }
{ buffer   }
{ keyword  } := GET_INFO  (window_variable,
{ string   }
{ window   }
{ widget   }
{ "before_bol"                                    }
{ "beyond_eob"                                    }
{ "beyond_eol"                                    }
{ "blink_status"                                  }
{ "blink_video"                                   }
{ "bold_status"                                   }
{ "bold_video"                                    }
{ "bound"                                         }
{ "bottom"  [[ , WINDOW                          ]]
{           [[ , TEXT                            ]]
{           [[ , VISIBLE_WINDOW                  ]]
{           [[ , VISIBLE_TEXT                    ]]
{ "buffer"                                        }
{ "current_column"                                }
{ "current_row"                                   }
{ "display_value"                                 }
{ "key_map_list"                                  }
{ "left"    [[ , WINDOW                          ]]
{           [[ , TEXT                            ]]
{           [[ , VISIBLE_WINDOW                  ]]
{           [[ , VISIBLE_TEXT                    ]]
{ "length"  [[ , WINDOW                          ]]
{           [[ , TEXT                            ]]
{           [[ , VISIBLE_WINDOW                  ]]
{           [[ , VISIBLE_TEXT                    ]]
{ "middle_of_tab"                                 }
{ "next"                                          }
{ "no_video"                                      }
{ "no_video_status"                               }
{ "original_bottom"                               }
{ "original_length"                               }
{ "original_top"                                  }
```



PARAMETERS

"before_bol"

Returns an integer (1 or 0) that indicates whether the cursor is to the left of the current line's left margin. The return value has no meaning if *"beyond_eob"* is true. This call returns 0 if the window you specified is not mapped.

"beyond_eob"

Returns an integer (1 or 0) that indicates whether the cursor is below the bottom of the buffer. This call returns 0 if the window you specified is not mapped.

"beyond_eol"

Returns an integer (1 or 0) that indicates whether the cursor is beyond the end of the current line. The return value has no meaning if *"beyond_eob"* is true. This call returns 0 if the window you specified is not mapped.

"blink_status"

Returns an integer (1 or 0) that indicates whether BLINK is one of the video attributes of the window's status line. This parameter is established or changed with the built-in procedure SET (STATUS_LINE).

"blink_video"

Returns an integer (1 or 0) that indicates whether BLINK is one of the video attributes of the window. This parameter is established or changed with the built-in procedure SET (VIDEO).

"bold_status"

Returns an integer (1 or 0) that indicates whether BOLD is one of the video attributes of the window's status line. This parameter is established or changed with the built-in procedure SET (STATUS).

"bold_video"

Returns an integer (1 or 0) that indicates whether BOLD is one of the video attributes of the window. This parameter is established or changed with the built-in procedure SET (VIDEO).

"bound"

Returns an integer (1 or 0) that indicates whether the cursor is located on a character.

"bottom"

Returns an integer that is the number of the last row or last visible row of the specified window, or the specified window's text area. The window row whose number is returned depends on the keyword you specify as the third parameter. If you do not specify a keyword, the default is TEXT. Valid keywords are as follows:

Table 7–5 Valid Keywords for the Third Parameter When the Second Parameter is "Bottom", "Left", "Length", "Right", "Top", or "Width"

Keyword	Definition
TEXT	A keyword directing the built-in to return the specified (left, right, top, or bottom) window row or column or the number of window rows or columns on which text can be displayed. By specifying TEXT instead of VISIBLE_TEXT, you obtain information about a window's rows and columns even if they are invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_TEXT.
VISIBLE_TEXT	A keyword directing the built-in to return the specified (left, right, top, or bottom) visible window row or column or the number of visible window rows or columns on which text can be displayed. When VAXTPU determines a window's last visible text row, VAXTPU does not consider the status line or the bottom scroll bar to be a text row.

(continued on next page)

Table 7–5 (Cont.) Valid Keywords for the Third Parameter When the Second Parameter is "Bottom", "Left", "Length", "Right", "Top", or "Width"

Keyword	Definition
VISIBLE_WINDOW	A keyword directing the built-in to return the specified (left, right, top, or bottom) visible window row or column or the number of visible window rows or columns in the window.
WINDOW	A keyword directing the built-in to return the specified (left, right, top, or bottom) window row or column or the number of window rows or columns in the window. By specifying WINDOW instead of TEXT, you obtain the window's last row or column, even if it cannot contain text because it contains a scroll bar or status line. By specifying WINDOW instead of VISIBLE_WINDOW, you obtain information about a window's rows and columns even if they are invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_WINDOW.

GET_INFO (window_variable, "bottom", TEXT) is a synonym for GET_INFO (window_variable, "original_bottom"). The call GET_INFO (window_variable, "bottom", VISIBLE_TEXT) is a synonym for GET_INFO (window_variable, "visible_bottom").

"buffer"

Returns the buffer that is associated with the window. Returns 0 if there is none.

"current_column"

Returns an integer that is the column in which the cursor was most recently located.

"current_row"

Returns an integer that is the row in which the cursor was most recently located.

"display_value"

Returns the display value of the specified window.

"key_map_list"

Returns the string that is the name of the key map list associated with the window you specify.

"left"

Returns an integer that is the number of the leftmost column or leftmost visible column of the specified window, or the specified window's text area. The column whose number is returned depends on the keyword you specify as the third parameter. If you do not specify a keyword, the default is TEXT. Valid keywords are shown in Table 7–5.

"length"
Returns an integer that is the number of rows or visible rows in the specified window or the specified window's text area. The number of rows returned depends on the keyword you specify as the third parameter. If you do not specify a keyword, the default is TEXT. Valid keywords are shown in Table 7–5.

"middle_of_tab"
Returns an integer (1 or 0) that indicates whether the cursor is in the middle of a tab. The return value has no meaning if *"beyond_eob"* is true. This call returns 0 if the window you specified is not mapped.

"next"
Returns the next window in VAXTPU's internal list of windows. Returns 0 if there are no more windows in the list.

"no_video"
Returns an integer (1 or 0) that indicates whether the video attribute of the window is NONE. This parameter is established or changed with the built-in procedure SET (VIDEO).

"no_video_status"
Returns an integer (1 or 0) that indicates whether the video attribute of the window's status line is NONE. This parameter is established or changed with the built-in procedure SET (STATUS).

"original_bottom"
Returns an integer that is the screen line number of the bottom of the window when it was created or last adjusted (does not include status line or scroll bar). Digital recommends that you retrieve this information using GET_INFO (window, "bottom", text).

"original_length"
Returns an integer that is the number of lines in the window when it was created. The value returned includes the status line.

Digital recommends that you retrieve this information using GET_INFO (window, "length", window).

"original_top"
Returns an integer that is the screen line number of the top of the window when it was created.

"pad"
Returns an integer (1 or 0) that indicates whether padding blanks have been displayed from column 1 to the left margin (if the left margin is greater than 1) and from the ends of lines to the right margin. This parameter is established or changed with the built-in procedure SET (PAD).

"previous"
Returns the previous window in VAXTPU's internal list of windows. Returns 0 if there are no previous windows in the list.

"reverse_status"

Returns an integer (1 or 0) that indicates whether REVERSE is one of the video attributes of the window's status line. This parameter is established or changed with the built-in procedure SET (STATUS).

"reverse_video"

Returns an integer (1 or 0) that indicates whether REVERSE is one of the video attributes of the window. This parameter is established or changed with the built-in procedure SET (VIDEO).

"right"

Returns an integer that is the number of the last column or last visible column of the specified window or the specified window's text area. The window column whose number is returned depends on the keyword you specify as the third parameter. If you do not specify a keyword, the default is TEXT. Valid keywords are shown in Table 7–5.

"scroll"

Returns an integer (1 or 0) that indicates whether scrolling is enabled for the window. This parameter is established or changed with the built-in procedure SET (SCROLLING).

"scroll_amount"

Returns an integer that is the number of lines to scroll. This parameter is established or changed with the built-in procedure SET.

"scroll_bar"

This parameter is used with DECwindows only.

Returns the specified scroll bar widget instance implementing the scroll bar associated with a window if it exists, otherwise returns 0.

You must specify the keyword HORIZONTAL or VERTICAL as the third parameter to GET_INFO (window_variable, "scroll_bar"). HORIZONTAL directs VAXTPU to return the window's horizontal scroll bar; VERTICAL directs VAXTPU to return the window's vertical scroll bar.

"scroll_bar_auto_thumb"

This parameter is used with DECwindows only.

Returns an integer (1 or 0) indicating whether automatic adjustment of the specified scroll bar slider is enabled. Returns 1 if automatic adjustment is enabled, 0 if it is disabled.

You must specify the keyword HORIZONTAL or VERTICAL as the third parameter to GET_INFO (window_variable, "scroll_bar_auto_thumb"). HORIZONTAL directs VAXTPU to return information about the window's horizontal scroll bar; VERTICAL directs VAXTPU to return information about the window's vertical scroll bar.

"scroll_bottom"

Returns an integer that is the bottom of the scrolling area, an offset from the bottom screen line. This parameter is established or changed with the built-in procedure SET (SCROLLING).

"scroll_top"
Returns an integer that is the top of the scrolling area, an offset from the top screen line. This parameter is established or changed with the built-in procedure SET (SCROLLING).

"shift_amount"
Returns an integer that is the number of columns the window is shifted to the left.

"special_graphics_status"
Returns an integer (1 or 0) that indicates whether SPECIAL_GRAPHICS is one of the video attributes of the window's status line. This parameter is established or changed with the built-in procedure SET (STATUS_LINE).

"status_line"
Returns a string that is the text of the status line. Returns 0 if there is none. This parameter is established or changed with the built-in procedure SET (STATUS_LINE).

"status_video"
If there is no video attribute or only one video attribute for the window's status line, the appropriate video keyword (NONE, BLINK, BOLD, REVERSE, UNDERLINE or SPECIAL_GRAPHICS) is returned. If there are multiple video attributes for the window's status line, the integer 1 is returned. If there is no status line for the window, the integer 0 is returned. This parameter is established or changed with the built-in procedure SET (STATUS_LINE).

"text"
Returns a keyword that indicates which keyword was used with SET (TEXT). SET (TEXT) controls text display in a window. SET (TEXT) returns any of the following keywords: BLANK_TABS, GRAPHIC_TABS, or NO_TRANSLATE.

"top"
Returns an integer that is the number of the first row or first visible row of the specified window or the specified window's text area. The window row whose number is returned depends on the keyword you specify as the third parameter. If you do not specify a keyword, the default is TEXT. Valid keywords are shown in Table 7–5.

"underline_status"
Returns an integer (1 or 0) that indicates whether UNDERLINE is one of the video attributes of the window's status line. This parameter is established or changed with the built-in procedure SET (STATUS_LINE).

"underline_video"
Returns an integer (1 or 0) that indicates whether UNDERLINE is one of the video attributes of the window. This parameter is established or changed with the built-in procedure SET (VIDEO).

"video"
If there is no video attribute or only one video attribute for the window, the appropriate video keyword (NONE, BLINK, BOLD, REVERSE, or UNDERLINE) is returned. If there are multiple video attributes for the window, the integer 1 is returned. If you get the return value 1 and you

want to know more about the window's video attributes, use the specific parameters such as *"underline_video"* and *"reverse_video"*.

This parameter is established or changed with the built-in procedure SET (VIDEO).

"visible"

Returns an integer (1 or 0) that indicates whether or not the window is mapped to the screen and whether it is occluded.

"visible_bottom"

Returns an integer that is the screen line number of the visible bottom of the window (does not include status line). This value can be changed using the ADJUST_WINDOW built-in, by creating other windows, or by mapping a window.

Digital recommends that you retrieve this information using GET_INFO (window, "bottom", visible_text).

"visible_length"

Returns an integer that is the visible length of the window (includes status line). This value differs from the value returned by GET_INFO (window_variable, "original_length") in that the value returned by *"visible_length"* is the original length minus the number of window lines (if any) hidden by occluding windows. This value can be changed using the ADJUST_WINDOW built-in, by creating other windows, or by mapping a window.

Digital recommends that you retrieve this information using GET_INFO (window, "length", visible_window).

"visible_top"

Returns an integer that is the screen line number of the visible top of the window. This value can be changed using the ADJUST_WINDOW built-in, by creating other windows, or by mapping a window on top of the current window.

Digital recommends that you retrieve this information using GET_INFO (window, "top", visible_window).

"width"

Returns an integer that is the number of columns or the number of visible columns in the specified window or the specified window's text area. The number of columns returned depends on the keyword you specify as the third parameter. If you do not specify a keyword, the default is TEXT. Valid keywords are shown in Table 7–5.

This parameter is established or changed with the built-in procedure SET.

EXAMPLES

```
1  last_line := GET_INFO (bottom_window, "bottom", WINDOW);
```

This statement returns the last line of the window *bottom_window*. The value returned is the line containing the status line or scroll bar, whichever comes last, if the window has a status line or scroll bar. For another example of code using GET_INFO (window_variable, "bottom", WINDOW) see Example B–5.

2
```
current_list := GET_INFO (CURRENT_WINDOW, "key_map_list");
```

This statement returns the key map list associated with the current window. For an example of code using GET_INFO (window_variable, "key_map_list", WINDOW) see Example B–6.

3
```
first_column := GET_INFO (CURRENT_WINDOW, "left", TEXT);
```

This statement returns the leftmost column where text can be displayed in the current window. Note that changing the left margin setting has no effect on the value returned.

4
```
the_length := the_length + GET_INFO (the_window, "length", WINDOW);
```

This statement adds the length of the window (the value in *the_window*) to the value in *the_length*. Note that the length of the window includes the length added by the scroll bar and status line, if the window has them. For another example of code using GET_INFO (window_variable, "length", WINDOW) see Example B–5.

5
```
last_column := GET_INFO (CURRENT_WINDOW, "right", WINDOW);
```

This statement returns the number of the rightmost column in the current window. Note that the column whose number is returned can be occupied by a vertical scroll bar if one is present. Note, too, that the returned value changes if you widen the window, but not if you move the window without widening it.

6
```
first_row := GET_INFO (CURRENT_WINDOW, "top", WINDOW);
```

This statement returns the number of the first row in the current window. Note that the row number returned is relative to the top of the VAXTPU screen. Thus, if the current window is not the top window on the VAXTPU screen, the row number returned is not 1. For another example of code using GET_INFO (window_variable, "top", WINDOW) see Example B–5.

7
```
the_width := GET_INFO (CURRENT_WINDOW, "width", WINDOW);
```

This statement returns the number of columns in the current window. For an example of code using GET_INFO (window_variable, "width", WINDOW) see Example B–6.

8
```
the_bar := GET_INFO (CURRENT_WINDOW, "scroll_bar", VERTICAL);
```

This statement returns the vertical scroll bar widget associated with the current window. For another example of code using GET_INFO (window_variable, "scroll_bar") see Example B–6.

9
```
status := GET_INFO (CURRENT_WINDOW,
                    "scroll_bar_auto_thumb", VERTICAL);
```

This statement returns an integer indicating whether automatic adjustment is enabled for the vertical scroll bar slider associated with the current window. For another example of code using GET_INFO (window_variable, "scroll_bar_auto_thumb", WINDOW) see Example B–6.

HELP_TEXT

Invokes the VMS Help Utility. You must specify the help library to be used for help information, the initial library topic, the prompting mode for the Help Utility, and the buffer to which the help information is to be written.

FORMAT

HELP_TEXT *(library-file, topic,* { *ON* / *OFF* } *,buffer)*

PARAMETERS

library-file

A string that is the file specification of the help library.

topic

A string that is the initial library topic. If this string is empty, the top level of help is displayed.

ON

A keyword specifying that the Help Utility should prompt the user for input.

OFF

Specifies that the prompting mode of the Help Utility should be turned off.

buffer

The buffer to which the help information is written.

DESCRIPTION

You can enter a complete file specification for the help library as the first parameter. However, if you enter only a file name, the Help Utility provides a default device (SYS$HELP) and default file type (HLB).

If you do not specify an initial topic as the second parameter, you must enter a null string as a place holder. The Help Utility then displays the top level of help available in the specified library.

When the prompting mode is ON for the built-in procedure HELP_TEXT, the following prompt appears if the help text contains more than one window of information:

```
Press RETURN to continue ...
```

Before VAXTPU invokes the Help Utility, VAXTPU erases the buffer specified as the help buffer. (In EVE the buffer to which the help information is written is represented by the variable *help_buffer*.) If the help buffer is associated with a window that is mapped to the screen, the window is updated each time VAXTPU prompts the user for input. If you set the prompting mode to OFF, then the window is not updated by the built-in procedure HELP_TEXT.

If *help_buffer* is not associated with a window that is mapped to the screen, the information from the Help Utility is not visible.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The HELP_TEXT built-in requires four parameters.
TPU$_TOOMANY	ERROR	You specified more than four parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	Only ON and OFF are allowed.
TPU$_NOTMODIFIABLE	WARNING	The output buffer is currently unmodifiable.
TPU$_SYSERROR	ERROR	Error activating the help librarian.
TPU$_OPENIN	ERROR	Error opening help library.

EXAMPLES

1
```
HELP_TEXT ("tpuhelp", "", OFF ,help_buffer)
```

This statement causes the top level of help information from the SYS$HELP:TPUHELP.HLB library to be written to the help buffer. The Help Utility prompting mode is not turned on.

2
```
HELP_TEXT ("tpuhelp", (READ_LINE ("Topic: ")), OFF, second_buffer)
```

This statement prompts the user to provide the topic for the Help Utility. The information on that topic that is in the VAXTPU help library is written to *second_buffer*.

3
```
! Interactive HELP
PROCEDURE user_help
   SET (STATUS_LINE, info_window, UNDERLINE,
       "Press CTRL/Z to leave prompts then CTRL/F to resume editing");
   MAP (info_window, help_buffer);
   HELP_TEXT ("USERHELP", READ_LINE ("Topic: "), ON, help_buffer);
ENDPROCEDURE;
```

This procedure displays information about getting out of help mode on the status line, prompts the user for input, and maps *help_buffer* to the screen.

INDEX

Locates a character or a substring within a string and returns its location within the string.

FORMAT

integer := INDEX *(string, substring)*

PARAMETERS

string

The string within which you want to find a character or a substring.

substring

A character or a substring whose leftmost character location you want to find within *string1*.

return value

An integer showing the character position within a string of the substring you specify.

DESCRIPTION

The built-in procedure INDEX finds the leftmost occurrence of *substring* within *string*. It returns an integer that indicates the character position in *string* at which *substring* was found. If *string* is not found, VAXTPU returns a 0. The character positions within *string* start at the left with 1.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	INDEX must be on the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	INDEX requires two arguments.
TPU$_TOOMANY	ERROR	INDEX accepts only two arguments.
TPU$_INVPARAM	ERROR	The arguments to INDEX must be strings.

EXAMPLES

```
1  loc := INDEX ("1234567","67")
```

This assignment statement stores an integer value of 6 in the variable *loc*, because the substring "67" is found starting at character position 6 within the string "1234567".

```
2  PROCEDURE user_is_character (c)
      LOCAL  symbol_characters;
      symbol_characters :=
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";
      RETURN INDEX( symbol_characters, c ) > 0;
ENDPROCEDURE;
```

This procedure uses the built-in procedure INDEX to return true if a given item is an alphanumeric character; otherwise, it returns false. (The list of characters in this example does not include characters that are not in the ASCII range of the DEC Multinational Character Set. However, you can write a procedure using such characters, because VAXTPU supports the DEC Multinational Character Set.) The parameter that is passed to this procedure is assumed to be a single character.

INT

Converts keyword or a string that consists of numeric characters into an integer.

FORMAT

integer3 := INT ({ *integer1* | *keyword* | *string* [[, *integer2*]] })

PARAMETERS

integer1

Any integer value. INT accepts a parameter of type integer so you need not check the type of the parameter you supply.

keyword

A keyword whose internal value you want.

string

A string that consists of numeric characters.

integer2

An integer specifying the radix (base) of the string being converted. The default radix is 10. The other allowable values are 8 and 16.

return value

The integer equivalent of the parameter you specify.

DESCRIPTION

You can use INT to store an integer value for a keyword or a string of numeric characters in a variable. You can then use the variable name in operations that require integer data types.

INT signals a warning and returns 0 if the string is not a number.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	INT returns a value that must be used.
TPU$_TOOFEW	ERROR	INT requires one parameter.
TPU$_TOOMANY	ERROR	INT accepts only one parameter.
TPU$_ARGMISMATCH	ERROR	The parameter to INT was not a keyword or string.
TPU$_INVNUMSTR	WARNING	The string you passed to INT was not a number.
TPU$_NULLSTRING	WARNING	You passed a string of length 0 to INT.

TPU$_BADVALUE	ERROR	You specified a value other than 8, 10, or 16 for the radix parameter.

EXAMPLES

1
```
user_int := INT ("12345")
```

This assignment statement converts the string "12345" into an integer value and stores it in the variable *user_int*.

2
```
! Parameters:
!
!   new_number          New integer value - output
!   prompt_string       Text of prompt - input
!   no_value_message    Message printed if user presses the
!                       RETURN key to get out of the command - input

PROCEDURE user_prompt_number (new_number, prompt_string,
                              no_value_message)
   LOCAL read_line_string;

   ON_ERROR
       IF ERROR = TPU$_NULLSTRING
       THEN
          MESSAGE (no_value_message);
       ELSE
          IF ERROR = TPU$_INVNUMSTR
          THEN
             MESSAGE (FAO ("Don't understand !AS",
                           read_line_string));
          ELSE
             MESSAGE (ERROR_TEXT);
          ENDIF;
       ENDIF;
       user_prompt_number := 0;
   ENDON_ERROR;

   user_prompt_number := 1;
   read_line_string := READ_LINE (prompt_string);

   EDIT (read_line_string, TRIM);
   TRANSLATE (read_line_string, "1", "l");

   new_number := INT (read_line_string);
ENDPROCEDURE;
```

This procedure is used by commands that prompt for integers. The procedure returns true if prompting worked or was not needed; it returns false otherwise. The number that is returned is returned in the output parameter.

JOURNAL_CLOSE

Closes an open keystroke journal file (if one exists for your session) and saves the journal file. Note that JOURNAL_CLOSE applies only to *keystroke* journaling.

FORMAT

JOURNAL_CLOSE

PARAMETERS

None.

DESCRIPTION

Once you specify JOURNAL_CLOSE, VAXTPU does not keep a keystroke journal of your work until you specify JOURNAL_OPEN. Calling the built-in procedure JOURNAL_OPEN causes VAXTPU to open a new keystroke journal file for your session.

To turn off buffer change journaling, see the description of the SET (JOURNALING) built-in procedure.

Caution: Journal files contain a record of *all* information being edited. Therefore, when editing files containing secure or confidential data, be sure to keep the journal files secure as well.

SIGNALED ERROR

TPU$_TOOMANY	ERROR	JOURNAL_CLOSE accepts no arguments.

EXAMPLE

```
JOURNAL_CLOSE
```

This statement causes VAXTPU to close the keystroke journal file, if one exists for your editing session.

JOURNAL_OPEN

Opens a keystroke journal file and starts making a copy of your editing session by recording every keystroke you make. If you invoked VAXTPU with the /RECOVER qualifier, then VAXTPU recovers the previous aborted section before recording new keystrokes. JOURNAL_OPEN optionally returns a string containing the file specification of the file journaled. Note that JOURNAL_OPEN applies only to *keystroke* journaling.

FORMAT

[string :=] JOURNAL_OPEN *(file-name)*

PARAMETER

file-name
A string that is the name of the keystroke journal file created for your editing session.

return value

The file specification of the file journaled.

DESCRIPTION

VAXTPU saves the keystrokes of your editing session by storing them in a buffer. VAXTPU writes the contents of this buffer to the file that you specify as a journal file. If for some reason VAXTPU should terminate unexpectedly, you can recover your editing session by using this journal file. To do this, invoke VAXTPU with the /RECOVER qualifier. See Chapter 5 for information on recovering files.

To turn on buffer change journaling, see the description of the SET (JOURNALING) built-in procedure.

By default, VAXTPU writes keystrokes to the journal file whenever the journal buffer contains 500 bytes of data. VAXTPU also tries to write keystrokes to the journal file when it aborts.

When you recover a VAXTPU session, your terminal characteristics should be the same as they were when the journal file was created. If they are not the same, VAXTPU informs you what characteristics are different and asks whether you want to continue recovering. If you answer yes, VAXTPU tries to recover; however, the different terminal settings may cause differences between the recovered session and the original session.

JOURNAL_OPEN succeeds if used in batch mode (NODISPLAY) but nothing is journaled as there are no keystrokes in batch mode.

Caution: Journal files contain a record of *all* information being edited. Therefore, when editing files containing secure or confidential data, be sure to keep the journal files secure as well.

SIGNALED ERRORS

TPU$_BADJOUFILE	ERROR	JOURNAL_OPEN could not open the journal file.
TPU$_TOOFEW	ERROR	JOURNAL_OPEN requires one argument.
TPU$_TOOMANY	ERROR	JOURNAL_OPEN accepts only one argument.
TPU$_INVPARAM	ERROR	The parameter to JOURNAL_OPEN must be a string.
TPU$_ASYNCACTIVE	WARNING	You cannot journal with asynchronous handlers declared.
TPU$_JNLOPEN	ERROR	A journal file is already open.

EXAMPLES

1

```
JOURNAL_OPEN ("test.fil")
```

This statement causes VAXTPU to open a file named TEST.FIL as the journal file for your editing session. VAXTPU uses your current default device and directory to complete the file specification.

2

```
PROCEDURE user_start_journal

! Default journal name
! Auxiliary journal name derived from file name

   LOCAL default_journal_name,
         aux_journal_name;

   IF (GET_INFO (COMMAND_LINE, "journal") = 1)
   AND
      (GET_INFO (COMMAND_LINE, "read_only") <> 1)
   THEN
      aux_journal_name := GET_INFO (CURRENT_BUFFER, "file_name");

      IF aux_journal_name = ""
      THEN
         aux_journal_name := GET_INFO (CURRENT_BUFFER, "output_file");
      ENDIF;

      IF aux_journal_name = 0
      THEN
         aux_journal_name := "";
      ENDIF;
```

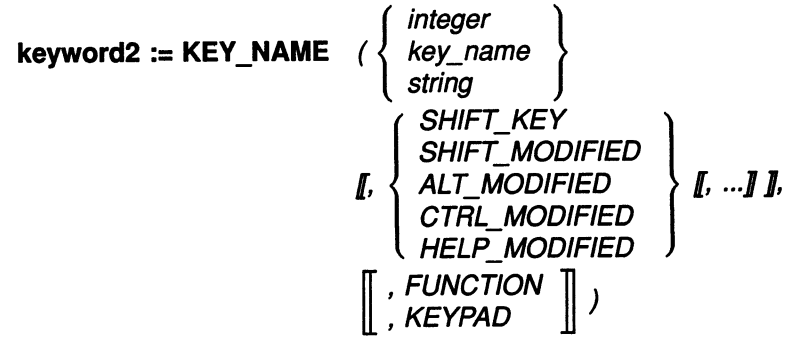
```
        IF aux_journal_name = ""
        THEN
           default_journal_name := "user.TJL";
        ELSE
           default_journal_name := ".TJL";
        ENDIF;

        journal_file := GET_INFO (COMMAND_LINE, "journal_file");
        journal_file := FILE_PARSE (journal_file, default_journal_name,
                                    aux_journal_name);
        JOURNAL_OPEN (journal_file);
     ENDIF;
ENDPROCEDURE;
```

This procedure starts journaling. It is called from the TPU$INIT_PROCEDURE after a file is read into the current buffer.

KEY_NAME

Returns a VAXTPU keyword for a key or a combination of keys, or creates a keyword used as a key name by VAXTPU.

FORMAT



PARAMETERS

integer

An integer that is either the integer representation of a keyword for a key, or is a value between 0 and 255 that VAXTPU interprets as the value of a character in the DEC Multinational Character Set.

key_name

A keyword that is the VAXTPU name for a key.

string

A string that is the value of a key from the main keyboard.

SHIFT_KEY

A keyword specifying that the key name created includes one or more shift keys. The keyword SHIFT_KEY specifies the VAXTPU shift key, not the key on the keyboard marked SHIFT. The shift key is also referred to as the GOLD key in EVE. (See the description of the SET (SHIFT_KEY) built-in in the *VAX Text Processing Utility Manual*.)

SHIFT_MODIFIED

A keyword specifying that the key name created by the built-in includes the key marked SHIFT on the keyboard. The keyword SHIFT_MODIFIED specifies the key that toggles between uppercase and lowercase, not the key known as the GOLD key.

SHIFT_MODIFIED only modifies function keys and keypad keys.

Digital recommends that you avoid using this keyword in the non-DECwindows version of VAXTPU. In non-DECwindows VAXTPU, when you create a key name with this keyword, the keyboard cannot generate a corresponding key.

ALT_MODIFIED

A keyword specifying that the key name created by the built-in includes the ALT key. Note that on most Digital keyboards the ALT key is labeled Compose Character.

ALT_MODIFIED only modifies function keys and keypad keys.

Digital recommends that you avoid using this keyword in the non-DECwindows version of VAXTPU. In non-DECwindows VAXTPU, when you create a key name with this keyword, the keyboard cannot generate a corresponding key.

CTRL_MODIFIED

A keyword specifying that the key name created by the built-in includes the CTRL key.

CTRL_MODIFIED only modifies function keys and keypad keys.

Digital recommends that you avoid using this keyword in the non-DECwindows version of VAXTPU. In non-DECwindows VAXTPU, when you create a key name with this keyword, the keyboard cannot generate a corresponding key.

HELP_MODIFIED

A keyword specifying that the key name created by the built-in includes the HELP key.

HELP_MODIFIED only modifies function keys and keypad keys.

Digital recommends that you avoid using this keyword in the non-DECwindows version of VAXTPU. In non-DECwindows VAXTPU, when you create a key name with this keyword, the keyboard cannot generate a corresponding key.

FUNCTION

A parameter that specifies that the resulting key name is to be that of a function key.

KEYPAD

A parameter that specifies that the resulting key name is to be that of a keypad key.

return value

A VAXTPU keyword to be used as the name of a key.

DESCRIPTION

Using the KEY_NAME built-in, you can create key names that are modified by more than one key. For example, it is possible to create a name for a key sequence consisting of the GOLD key, the CTRL key, and an alphanumeric or keypad key.

The built-in GET_INFO (key_name, "key_modifiers") returns a bit-encoded integer whose value represents the key modifier or combination of key modifiers used to create a given key name. For more information about interpreting the integer returned, see the description of GET_INFO (key_name, "key_modifiers").

The built-in GET_INFO (keyword, "name") has been extended to return a string including all the key modifier keywords used to create a key name. For more information about fetching the string equivalent of a key name, see the description of GET_INFO (keyword, "name").

SIGNALED ERRORS

TPU$_INCKWDCOM	WARNING	Inconsistent keyword combination.
TPU$_MUSTBEONE	WARNING	String must be one character long.
TPU$_NOTDEFINABLE	WARNING	Second argument is not a valid reference to a key.
TPU$_NEEDTOASSIGN	ERROR	KEY_NAME call must be on the right-hand side of an assignment statement.
TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to the KEY_NAME built-in.
TPU$_BADKEY	ERROR	KEY_NAME accepts SHIFT_KEY, FUNCTION, or KEYPAD as a keyword argument.
TPU$_TOOFEW	ERROR	Too few arguments passed to the KEY_NAME built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the KEY_NAME built-in.

EXAMPLES

1
```
new_key := KEY_NAME (KP4, CTRL_MODIFIED, SHIFT_KEY);
DEFINE_KEY ("eve_fill", new_key);
```

These statements create a name for the key sequence GOLD/CTRL/KP4 and bind the EVE command FILL to the resulting key sequence.

2
```
key1 := KEY_NAME ("Z")
```

This assignment statement creates the key name *key1* for the keyboard key Z.

3
```
key2 := KEY_NAME (KP5, SHIFT_KEY)
```

This example uses KEY_NAME to create a key name for a combination of keys.

4
```
key3 := KEY_NAME (ASCII (10))
```

This assignment statement creates the key name *key3* for the line-feed character.

5
```
! Procedure to define keys to emulate EDT
PROCEDURE user_define_edtkey
! Bind the EDT Fndnxt function to PF3
  DEFINE_KEY ("edt$search_next", PF3);
!  Bind the EDT Find function to SHIFT PF3
  DEFINE_KEY ("edt$search", KEY_NAME (PF3, SHIFT_KEY));
ENDPROCEDURE;
```

This example shows a portion of a command file that defines the keys for an editing interface that emulates EDT.

6 `key4 := KEY_NAME (90)`

This assignment statement creates the key name *key4* for the keyboard key Z. The key name is identical to *key1* in the first example, because 90 is the ASCII code for Z.

7 `key5 := KEY_NAME ("A", KEYPAD)`

This assignment statement creates the key name *key5* for the keypad key that is terminated by an A in the code that represents key names. This is identical to the key name *UP*, which VAXTPU uses to refer to the up arrow key.

VAXTPU defines a keypad key as a control sequence consisting of the code *SS3* followed by a character. The control sequence *SS3* can be represented as follows:

```
Esc O
```

For more information on the representation of keys, see the manual for your terminal.

8 `key6 := KEY_NAME (29, FUNCTION)`

This assignment statement creates the key name *key6* for the function key whose representation contains the number 29. This is identical to the VAXTPU keyword DO, which VAXTPU uses to identify the Do key.

VAXTPU defines a function key as a control sequence with the following format:

CSI decimal-number ~

The element CSI can be represented as follows:

ESC [

In this representation, the decimal number must be in the range 0 to 255. For more information on the representation of keys, see the manual for your terminal.

LAST_KEY

Returns a VAXTPU keyword for the last key that was entered, read, or executed.

FORMAT

keyword := LAST_KEY

PARAMETERS

None.

DESCRIPTION

When VAXTPU is replaying a learn sequence or executing the program bound to a key, LAST_KEY returns the last key replayed or processed so far, not the last key that was pressed to invoke the learn sequence or program.

When you invoke VAXTPU with the /NODISPLAY qualifier, the value 0 is returned for LAST_KEY, except in the following case. If you precede the LAST_KEY statement with a READ_LINE statement, LAST_KEY can return a key name representing the last key read by READ_LINE, CTRL/Z, or the RETURN key. See the description of READ_LINE for more information on the values that LAST_KEY can return when you use LAST_KEY while running VAXTPU in /NO_DISPLAY mode.

SIGNALED ERROR

TPU$_TOOMANY	ERROR	Too many arguments passed to the LAST_KEY built-in.

EXAMPLE

```
PROCEDURE user_define_key

   def := READ_LINE ("Definition: ");
   key := READ_LINE ("Press key to define.",1);
   IF LENGTH (key) > 0
   THEN
       key := KEY_NAME (key)
   ELSE
         key := LAST_KEY;
   ENDIF;
   DEFINE_KEY (def, key);
ENDPROCEDURE;
```

This procedure prompts the user for input for key definitions.

LEARN_ABORT

Causes a learn sequence being replayed to be terminated whether or not the learn sequence has completed.

FORMAT

[integer :=] LEARN_ABORT

PARAMETERS

None.

return value

An integer indicating whether a learn sequence was actually replaying at the time the LEARN_ABORT statement was encountered. The value 1 is returned if a learn sequence was being replayed, 0, otherwise.

DESCRIPTION

LEARN_ABORT aborts a learn sequence that is being replayed. Only the currently executing learn sequence is aborted.

Whenever you write a procedure that can be bound to a key, the procedure should invoke the LEARN_ABORT built-in in case of error. Using LEARN_ABORT prevents a learn sequence from finishing if the learn sequence calls the user-written procedure and the procedure is not executed successfully.

SIGNALED ERROR

TPU$_TOOMANY	ERROR	The LEARN_ABORT built-in takes no parameters.

EXAMPLE

```
ON_ERROR
   MESSAGE ("Aborting command because of error.");
   LEARN_ABORT;
   ABORT;
ENDON_ERROR
```

In this error handler, if an error occurs any executing learn sequence is aborted.

LEARN_BEGIN and LEARN_END

Saves all keystrokes typed between LEARN_BEGIN and LEARN_END. LEARN_BEGIN starts saving all keystrokes that you type. LEARN_END stops the "learn mode" of VAXTPU and returns a learn sequence consisting of all the keystrokes that you entered.

FORMAT

LEARN_BEGIN ({ *EXACT* / *NO_EXACT* })

.

.

.

learn := LEARN_END

PARAMETERS

EXACT

Causes VAXTPU to use the input that was entered for each READ_LINE, READ_KEY, or READ_CHAR built-in procedure when the learn sequence was created as the input for these built-in procedures when the learn sequence is replayed.

NO_EXACT

Causes VAXTPU to prompt for new input each time a READ_LINE, READ_KEY, or READ_CHAR built-in procedure is replayed within a learn sequence.

return value

A variable of type learn storing the keystrokes you specify.

DESCRIPTION

You can use the variable name that you assign to a learn sequence as the parameter for the built-in procedure EXECUTE to replay a learn sequence. You can also use the variable name with the built-in procedure DEFINE_KEY to bind the sequence to a key so that the learn sequence is executed when you press a key.

Learn sequences are different from other VAXTPU programs in that they are created with keystrokes rather than with VAXTPU statements. You create the learn sequence as you are entering text and executing VAXTPU commands. Because learn sequences make it easy to collect and execute a sequence of VAXTPU commands, they are convenient for creating temporary "programs." You can replay these learn sequences during the editing session in which you create them.

Learn sequences, created by collecting keystrokes, are not flexible enough to use for writing general programs. Learn sequences are best suited to saving a series of editing actions that you perform many times during a single editing session.

It is possible to save learn sequences from session to session so that you can replay them in an editing session other than the one in which you created them. To save a learn sequence, bind it to a key; before ending your editing session, use the built-in procedure SAVE to do an incremental save to the section file you are using. Using the built-in procedure SAVE causes the new definitions from the current session to be added to the section file with which you invoked VAXTPU. For more information, see the built-in procedure SAVE.

VAXTPU key definitions may change in future versions. You may lose learn sequences that you have saved when you run a new version of VAXTPU.

Note: You should not use built-in procedures that can return WARNING or ERROR messages as a part of a learn sequence because learn sequences do not stop on error conditions. Because the learn sequence continues executing after an error or warning condition, the editing actions that are executed after an error or a warning may not take effect at the character position you desire.

If, for example, a built-in procedure SEARCH that you use as a part of a learn sequence fails to find the string you specify and issues a warning, the learn sequence does not stop executing. This can cause the rest of the learn sequence to take inappropriate editing actions.

Pre- and postkey procedures interact with learn sequences in the following order:

1. When the user presses the key or key sequence to which the learn sequence is bound, VAXTPU executes the prekey procedure of that key if a prekey procedure has been set.
2. For each key in the learn sequence, VAXTPU executes procedures or programs in the following order:
   a. VAXTPU executes the prekey procedure of that key if a prekey procedure has been set.
   b. VAXTPU executes the code bound to the key itself.
   c. VAXTPU executes the postkey procedure of that key if a postkey procedure has been set.
3. When all keys in the learn sequence have been processed, VAXTPU executes the postkey procedure, if one has been set, for the key to which the entire learn sequence was bound.

SIGNALED ERRORS

TPU$_NOTLEARNING	WARNING	LEARN_BEGIN was not used since the last call to LEARN_END.
TPU$_ONELEARN	WARNING	A learn sequence is already in progress.

TPU$_TOOFEW	ERROR	LEARN_BEGIN requires one argument.
TPU$_TOOMANY	ERROR	LEARN_BEGIN accepts only one argument.
TPU$_INVPARAM	ERROR	The specified parameter has the wrong type.

EXAMPLE

```
LEARN_BEGIN (EXACT)
             .
             .
             .
This represents a typical editing session,
in which you perform commands that are
bound to keys.
             .
             .
             .
do_again := LEARN_END
```

This example shows how to combine LEARN_BEGIN and LEARN_END so that all of the keystrokes that you enter between them are saved. The keyword (EXACT) specifies that if you use READ_LINE, READ_CHAR, or READ_KEY within the learn sequence, any input that you enter for these built-in procedures is repeated exactly when you replay the learn sequence.

LENGTH

Returns an integer that is the number of character positions in a buffer, range, or string.

FORMAT

integer := LENGTH ({ *buffer* | *range* | *string* })

PARAMETERS

buffer

The buffer whose length you want to determine. If you specify a buffer, line terminators are not counted as character positions.

range

The range whose length you want to determine. If you specify a range, line terminators are not counted as character positions.

string

The string whose length you want to determine.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	LENGTH must be on the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	LENGTH requires one argument.
TPU$_TOOMANY	ERROR	LENGTH accepts only one argument.
TPU$_ARGMISMATCH	ERROR	The argument to LENGTH must be a string or a range.
TPU$_CONTROLC	ERROR	You pressed CTRL/C while LENGTH was executing.

EXAMPLES

1
```
str_len := LENGTH ("Don Quixote")
```

This assignment statement stores the number of characters in the string "Don Quixote" in the variable *str_len*. In this example, the integer value is 11.

2
```
user_how_long := LENGTH (my_range)
```

This assignment statement stores the number of character positions (excluding line terminators) in *my_range* in the variable *user_how_long*.

```
3   ! Parameters:
    !
    !   mark_parameter is user-supplied string,
    !   which is used as a mark name

    PROCEDURE user_mark_ (mark_parameter)

    ! Local copy of mark_parameter

       LOCAL mark_name;

       ON_ERROR
           MESSAGE (FAO ("Cannot use !AS as a mark name", mark_name));
           RETURN;
       ENDON_ERROR;

    ! 132 - length ("user_mark_")

       IF LENGTH (mark_parameter) > 122
       THEN
           mark_name := SUBSTR (mark_name, 1, 122);
       ELSE
           mark_name := mark_parameter;
       ENDIF;

       EXECUTE ("user_mark_" + mark_name + " := MARK (NONE)");
       MESSAGE (FAO ("Current position marked as !AS", mark_name));
    ENDPROCEDURE;
```

This procedure puts a marker without any video attributes at the current position. The marker is assigned to a variable that begins with *user_mark_* and ends with the string you pass as a parameter. The procedure writes a message to the message area verifying the mark name that comes from the input parameter.

LINE_BEGIN

Matches the beginning of a line.

FORMAT

LINE_BEGIN

PARAMETERS

None.

DESCRIPTION

When used as part of a complex pattern or as an argument to SEARCH, LINE_BEGIN matches the start of a line.

Although LINE_BEGIN behaves much like a built-in, it is actually a keyword.

LINE_BEGIN lets you search for complex strings by creating patterns that match certain conditions. For example, if you want to find all occurrences of the exclamation point (!) when it is the first character in the line, use LINE_BEGIN to create the following pattern:

```
pat1 := LINE_BEGIN + "!";
```

For more information on patterns, see Chapter 2.

SIGNALED ERROR

LINE_END is a keyword and has no completion codes.

EXAMPLES

1
```
pat1 := LINE_BEGIN
```

This assignment statement stores the beginning-of-line condition in the variable *pat1*.

2
```
POSITION (SEARCH (LINE_BEGIN, REVERSE));
```

This VAXTPU statement positions you at the beginning of the current line.

3
```
PROCEDURE user_remove_dsrlines

   LOCAL s1,
         pat1;

   pat1 := LINE_BEGIN + ".";

   LOOP
      s1 := SEARCH_QUIETLY (pat1, FORWARD);
      EXITIF s1 = 0;
      POSITION (s1);
      ERASE_LINE;
   ENDLOOP;
ENDPROCEDURE;
```

This procedure removes all DSR commands from a file by searching for a pattern that has a period (.) at the beginning of a line and then removing the lines that match this condition.

LINE_END

Matches the end of a line.

FORMAT

LINE_END

PARAMETERS

None.

DESCRIPTION

When used as part of a complex pattern or as an argument to SEARCH, LINE_END matches the end of a line.

Although LINE_END behaves much like a built-in, it is actually a keyword.

The end-of-line condition is one character position to the right of the last character on a line.

For more information on patterns, see Chapter 2.

SIGNALED ERROR

LINE_END is a keyword and has no completion codes.

EXAMPLES

1
```
pat1 := LINE_END
```

This assignment statement stores the keyword LINE_END in the variable *pat1*. *Pat1* can be used as an argument to the SEARCH built-in or as part of a complex pattern.

2
```
PROCEDURE user_end_of_line
   LOCAL eol_range;
   eol_range := SEARCH_QUIETLY (LINE_END, FORWARD);
   IF eol_range <> 0
   THEN
      POSITION (eol_range);
   ENDIF;
ENDPROCEDURE;
```

If you are not already at the end of the current line, the preceding procedure moves the editing point to the end of the line.

LOCATE_MOUSE

Locates the window position of the pointer at the time LOCATE_MOUSE is invoked. LOCATE_MOUSE returns the window name and the window position of the pointer and optionally returns a status indicating whether the pointer was found in a window.

FORMAT

[integer :=] LOCATE_MOUSE *(window, x_integer, y_integer)*

PARAMETERS

window

Returns the window in which the pointer is located. You can pass any data type except a constant in this parameter. If the pointer is not found, an unspecified data type is returned.

x_integer

Returns the column position of the pointer. You can pass any data type except a constant in this parameter. If the pointer is not found, an unspecified data type is returned.

y_integer

Returns the row position of the pointer. You can pass any data type except a constant in this parameter. If the pointer is not found, an unspecified data type is returned. This parameter returns 0 if the pointer is in the status line for a window.

return value

An integer indicating whether the pointer was found in a window. The value is 1 if VAXTPU finds a window position, 0, otherwise.

DESCRIPTION

When the user presses a mouse button, VAXTPU determines the location of the mouse pointer and makes that information available while the code bound to the mouse button is being processed. Mouse pointer location information is not available at any other time.

In DECwindows VAXTPU, you can use the built-in LOCATE_MOUSE anytime after the first keyboard or mouse-button event. The built-in returns the location occupied by the pointer cursor at the time of the most recent keyboard or mouse button event.

If there is no mouse information available (because no mouse button has been pressed or if the mouse has been disabled using SET (MOUSE)), LOCATE_MOUSE signals the status TPU$_MOUSEINV.

SIGNALED ERRORS

TPU$_MOUSEINV	WARNING	The mouse position is not currently valid.
TPU$_TOOFEW	ERROR	LOCATE_MOUSE requires three parameters.
TPU$_TOOMANY	ERROR	LOCATE_MOUSE accepts at most three parameters.
TPU$_BADDELETE	ERROR	You have specified a constant as one or more of the parameters.

EXAMPLES

1
```
LOCATE_MOUSE (abc_window, x_1, Y1);
```

The example returns the window and coordinate position of the pointer.

2
```
PROCEDURE user_move_to_mouse

   LOCAL my_window,
         x_1,
         y1;

   my_window := 0;
   x_1 := 0;
   y1 := 0;

   IF (LOCATE_MOUSE (my_window, x_1, Y1) <> 0)
   THEN
      IF (CURRENT_WINDOW <> my_window)
      THEN
         POSITION (my_window);
         UPDATE (my_window);
      ENDIF;
      CURSOR_VERTICAL (y1 - (CURRENT_ROW - GET_INFO
                      (my_window,"visible_top") + 1));
      CURSOR_HORIZONTAL (CURRENT_COLUMN - x_1);
   ENDIF;
ENDPROCEDURE;
```

Binding the *user_move_to_mouse* procedure to a mouse button moves the cursor to the mouse location. The *user_move_to_mouse* procedure is essentially equivalent to POSITION (MOUSE).

Note that CURRENT_ROW and CURRENT_COLUMN return screen-relative location information, while LOCATE_MOUSE returns window-relative location information.

3
```
status := LOCATE_MOUSE (new_window, x_value, y_value);
```

The previous statement returns an integer in the variable *status* indicating whether the pointer cursor was found in a window, the window in the parameter *new_window* where the mouse was found, an integer in the parameter *x_value* specifying the pointer cursor's location in the horizontal dimension, and an integer in the parameter *y_value* specifying the pointer cursor's location in the vertical dimension.

LOOKUP_KEY

Returns the executable code or the comment that is associated with the key you specify. The code can be returned as a program or as a learn sequence. The comment is returned as a string.

FORMAT

```
{ integer       }
{ learn_sequence } := LOOKUP_KEY
{ program       }
{ string3       }

(key-name, { COMMENT  } [[ { , string1 } ]])
           { KEY_MAP  }    { , string2 }
           { PROGRAM  }
```

PARAMETERS

key-name

A VAXTPU key name for a key or a combination of keys. See Table 2–1 for a list of the VAXTPU key names for the VT300-series, VT200-series, and VT100-series keyboards.

COMMENT

A keyword specifying that the LOOKUP_KEY built-in is to return the comment supplied when the key was defined. If no comment was supplied, the LOOKUP_KEY built-in returns the integer zero.

KEY_MAP

A keyword specifying that the LOOKUP_KEY built-in is to return the key map in which the key's definition is stored. If you specify a key that is not defined in any key map, LOOKUP_KEY returns a null string.

PROGRAM

A keyword specifying that the LOOKUP_KEY built-in is to return the program or learn sequence bound to the key specified. If the key is not defined, the LOOKUP_KEY built-in returns the integer 0.

string1

The name of the key map from which the LOOKUP_KEY built-in is to obtain the key definition. Use this optional parameter if the key is defined in more than one key map. If you do not specify a key map or a key map list for the third parameter, the first definition found for the specified key in the key map list bound to the current buffer is returned.

string2

The name of the key map list from which the LOOKUP_KEY built-in is to obtain the key definition. Use this optional parameter if the key is defined in more than one key map list. If you do not specify a key map or a key map list for the third parameter, the first definition found for the specified key in the key map list bound to the current buffer is returned.

return value

- **integer** — The integer 0. This value is returned if the key specified as a parameter has no definition.
- **learn_sequence** — The learn sequence bound to the key specified as a parameter.
- **program** — The program bound to the key specified as a parameter.
- **string3** — If you specified COMMENT as the second parameter, *string3* is the comment bound to the key specified as the first parameter. If you specified KEY_MAP as the second parameter, *string3* is the string naming the key map in which the key definition was found.

DESCRIPTION

The LOOKUP_KEY built-in procedure can return a program, a learn sequence, a string, or the integer 0 (0 means that the key has no definition).

LOOKUP_KEY is useful when you are defining keys temporarily during an editing session and you want to check the existing definitions of a key.

SIGNALED ERRORS

TPU$_NOTDEFINABLE	WARNING	Argument is not a valid reference to a key.
TPU$_NOKEYMAP	WARNING	Argument is not a defined key map.
TPU$_NOKEYMAPLIST	WARNING	Argument is not a defined key map list.
TPU$_KEYMAPNTFND	WARNING	The specified key map is not found.
TPU$_EMPTYKMLIST	WARNING	The specified key map list contains no key maps.
TPU$_TOOFEW	ERROR	Too few arguments passed to the LOOKUP_KEY built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the LOOKUP_KEY built-in.
TPU$_NEEDTOASSIGN	ERROR	LOOKUP_KEY must be on the right-hand side of an assignment statement.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the LOOKUP_KEY built-in.
TPU$_BADKEY	ERROR	An unknown keyword has been used as an argument. Only PROGRAM, COMMENT, and KEY_MAP are valid keywords.

EXAMPLES

1
```
programx := LOOKUP_KEY (key1, PROGRAM)
```

This assignment statement returns the executable code that is associated with *key1*. The second keyword, PROGRAM, indicates that the result is returned to a variable of type program or learn.

2
```
PROCEDURE user_what_is_comment
   MESSAGE (LOOKUP_KEY (LAST_KEY, COMMENT));
ENDPROCEDURE;
```

This procedure displays in the message area the comment that you included with your key definition for the last key that you typed.

3
```
PROCEDURE user_get_key_info
   LOCAL key_to_interpret,
         key_info;

   MESSAGE ("Press the key you want information on: ");

   key_to_interpret := READ_KEY;
   key_info := LOOKUP_KEY (key_to_interpret, COMMENT);

   IF key_info <> ""
   THEN
       MESSAGE ("Comment: " + key_info);
   ELSE
       MESSAGE ("No comment is associated with that key.");
   ENDIF;
ENDPROCEDURE;
```

This procedure returns the comment associated with a particular key.

4
```
key_map_name := LOOKUP_KEY (RET_KEY, KEY_MAP, "tpu$key_map_list");
IF LENGTH (key_map_name) = 0
THEN
     MESSAGE ("RET_KEY is undefined");
ELSE
     MESSAGE ("RET_KEY is defined in key map " + key_map_name);
ENDIF;
```

This procedure returns the key map within the key map list TPU$KEY_MAP_LIST in which the RETURN key is defined.

5
```
PROCEDURE shift_key_handler (key_map_list_name);

   LOCAL bound_program;

   bound_program := LOOKUP_KEY (READ_KEY, PROGRAM, "key_map_list_name");

   IF bound_program <> 0
   THEN
      EXECUTE (bound_program);
   ELSE
      MESSAGE ("Attempt to execute undefined key");
   ENDIF;
ENDPROCEDURE;
```

```
red_keys := CREATE_KEY_MAP ("red_keys");
red_key_map_list := CREATE_KEY_MAP_LIST ("red_key_map_list",
                    red_keys);
DEFINE_KEY ("shift_key_handler (red_key_map_list)", PF3,
           "RED shift key");
```

This procedure implements multiple shift keys.

MANAGE_WIDGET

Makes the specified widget instances visible, provided that the specified widgets' parent is also visible.

FORMAT

MANAGE_WIDGET *(widget [, widget...])*

PARAMETERS

widget
The widget instance to be managed.

DESCRIPTION

This built-in performs the same functions as the X Toolkit MANAGE CHILD and MANAGE CHILDREN routines.

If you have multiple children of a single widget that you want to manage, include them in a single call to MANAGE_WIDGET. Managing several widgets at once is more efficient than managing one widget at a time.

All widgets passed in the same MANAGE_WIDGET operation must have the same parent.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	You specified a parameter of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to the MANAGE_WIDGET bulit-in.
TPU$_NORETURNVALUE	ERROR	MANAGE_WIDGET cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the MANAGE_ WIDGET built-in only if you are using DECwindows VAXTPU.
TPU$_WIDMISMATCH	ERROR	You have specified a widget whose class is not supported.

EXAMPLE

For a sample procedure using the MANAGE_WIDGET built-in, see Example B–2.

MAP

Associates a buffer with a window and causes the window or widget to become visible on the screen. Before using MAP, you must already have created the widget, buffer, and window that you specify as parameters. See CREATE_WIDGET, CREATE_BUFFER, and CREATE_WINDOW.

FORMAT

MAP ({ *window, buffer* / *widget* })

PARAMETERS

window

The window you want to map to the screen.

buffer

The buffer you want to associate with the window.

widget

The widget instance you want to make visible.

DESCRIPTION

The window and buffer that you use as parameters become the current window and the current buffer, respectively. The map operation synchronizes the cursor position with the editing point in the buffer. If the window is not already mapped to the buffer when you use MAP, VAXTPU puts the cursor back in the last position the cursor occupied the last time the window was the current window.

MAP may cause other windows that are mapped to the screen to be partially or completely occluded. If MAP causes the new window to segment another window into two pieces, only the upper part of the segmented window remains visible and continues to be updated. The lower part of the segmented window is erased on the next screen update. If you remove the window that is segmenting another window, VAXTPU repaints the screen so that the window that was segmented regains its original size and position on the screen.

In DECwindows, MAP also maps the VAXTPU main widget if it has not already been mapped.

Note that if you execute MAP within a procedure, the screen is not updated to reflect such operations as window repainting, line erasure, or new mapping until the procedure has finished executing and control has returned to the screen manager. If you want the screen to reflect the changes before the entire program is executed, you can force the immediate update of a window by including the following statement in the procedure before any statements containing the MAP built-in:

```
UPDATE(WINDOW);
```

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	MAP requires at least two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_MAXMAPPEDBUF	WARNING	The buffer is already mapped to the maximum number of windows allowed by VAXTPU.

EXAMPLES

1
```
MAP (main_window, main_buffer)
```

This statement associates the main buffer with the main window and maps the main window to the screen. You must have established the main buffer and the main window with CREATE_BUFFER and CREATE_WINDOW before you can use them as parameters for MAP.

2
```
PROCEDURE user_message_window

   message_buffer := CREATE_BUFFER ("message");
   SET (EOB_TEXT, message_buffer, "");
   SET (NO_WRITE, message_buffer);
   SET (SYSTEM, message_buffer);

   message_window := CREATE_WINDOW (23, 2, OFF);
   SET (VIDEO, message_window, NONE);
   MAP (message_window, message_buffer);
ENDPROCEDURE;
```

This procedure creates a message buffer and a message window. It then associates the message buffer with the message window and maps the message window to the screen.

3
```
MAP (example_widget);
```

This statement causes the widget assigned to the variable *example_widget* to become visible if the widget has been created and managed but not mapped. For more information on how to map widgets without managing them, see the description of the SET (MAPPED_WHEN_MANAGED) built-in.

MARK

Returns a marker for the editing point in the current buffer. You must specify how the marker is to be displayed on the screen (no special video, reverse video, bolded, blinking, or underlined).

FORMAT

marker := MARK ({ *BLINK* | *BOLD* | *FREE_CURSOR* | *NONE* | *REVERSE* | *UNDERLINE* })

PARAMETERS

BLINK
A keyword directing VAXTPU to display the marker in blinking rendition.

BOLD
A keyword directing VAXTPU to display the marker in bold rendition.

FREE_CURSOR
A keyword directing VAXTPU to create a free marker (that is, a marker not bound to a character). Specifying the parameter FREE_CURSOR does not create a free marker unless the editing point is before the beginning of a line, after the end of a line, in the middle of a tab, or below the bottom of a buffer when the statement MARK (FREE_CURSOR) is executed. If the editing point is on a character when the statement is executed, the marker is bound. A free marker has no video attribute.

NONE
A keyword directing VAXTPU to apply no video attributes to the marker.

REVERSE
A keyword directing VAXTPU to display the marker in reverse video.

UNDERLINE
A keyword directing VAXTPU to underline the marker.

DESCRIPTION

This built-in procedure can be used to establish place holders, or "bookmarks."

A marker can be either **bound** or **free**. For more information on how these markers differ, see Chapter 2.

To create a bound marker, use the MARK built-in with any of its parameters except FREE_CURSOR. This operation creates a bound marker even if the editing point is beyond the end of a line, before the beginning of a line, in the middle of a tab, or beyond the end of a buffer. To create a bound cursor in a location where there is no character, VAXTPU fills the space between the marker and the nearest character with padding space characters.

A bound marker is tied to the character at which it is created. If the character tied to the marker moves, the marker moves also. If the character tied to the marker is deleted, the marker moves to the nearest character position. The nearest character position is determined in the following way:

1 If there is a character position on the same line and to the right, the marker moves to this position, even if the position is at the end of the line.

2 If the line on which the marker is located is deleted, the marker moves to the first position on the following line.

You can move one column past the last character in a line and place a marker there. However, the video attribute for the marker is not visible unless a subsequent operation puts a character under the marker.

If you use a marker at the end of a line as part of a range, it is included in the range even though the marker is not positioned on a character.

A marker is free if the following conditions are true:

- You used the statement *marker_variable* := MARK(FREE_CURSOR) to create the marker.
- There was no character in the position marked by the editing point at the time you created the marker.

VAXTPU keeps track of the location of a free marker by measuring the distance between the marker and the character nearest to the marker. If you move the character from which VAXTPU measures distance to a free marker, the marker moves too. VAXTPU preserves a uniform distance between the character and the marker. If you collapse white space containing one or more free markers (for example, if you delete a tab or use the APPEND_LINE built-in), VAXTPU preserves the markers and binds them to the nearest character.

If the current buffer is mapped to a visible window, the MARK built-in causes the screen manager to synchronize the editing point, which is a buffer location, with the cursor position, which is a window location. Unless you specify the parameter FREE_CURSOR, using the MARK built-in may result in the insertion of padding spaces or lines into the buffer if the cursor position is before the beginning of a line, in the middle of a tab, beyond the end of a line, or after the last line in the buffer.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	MARK requires one parameter.
TPU$_TOOMANY	ERROR	MARK accepts only one parameter.
TPU$_NEEDTOASSIGN	ERROR	The MARK built-in must be on the right-hand side of an assignment statement.

TPU$_NOCURRENTBUF	WARNING	You must be positioned in a buffer to set a marker.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	The keyword must be NONE, BOLD, BLINK, REVERSE, UNDERLINE, or FREE_CURSOR.
TPU$_UNKKEYWORD	ERROR	You have specified an unknown keyword.
TPU$_INSVIRMEM	FATAL	There is not enough memory to create the marker.

EXAMPLES

1
```
user_mark := MARK (NONE)
```

This assignment statement places a marker at the editing point. There are no video attributes applied to the marker.

2
```
user_mark_under := MARK (UNDERLINE)
```

This assignment statement places a marker at the row and column position that corresponds to the editing point. The character tied to the marker is underlined.

3
```
my_mark1 := MARK (UNDERLINE);
my_mark2 := MARK (BLINK);
```

These assignment statements place a marker at the row and column position that corresponds to the editing point. The character tied to the marker is underlined and blinks.

4
```
PROCEDURE user_paste

   temp_pos := MARK (NONE);

   POSITION (END_OF (paste_buffer));
   MOVE_HORIZONTAL (-2);

   paste_text := CREATE_RANGE (BEGINNING_OF (paste_buffer),
                 MARK (NONE), NONE);
   POSITION (temp_pos);
   COPY_TEXT (paste_text);
ENDPROCEDURE;
```

This procedure marks a temporary position at the current character position, and then goes to the paste buffer and creates a range of the contents of the paste buffer. VAXTPU then goes to *temp_pos* and copies the text from the paste buffer at the temporary position.

MATCH

MATCH returns a pattern that matches from the editing point up to and including the sequence of characters specified in the parameter.

FORMAT

pattern := MATCH ({ *buffer* | *range* | *string* })

PARAMETERS

buffer

An expression that evaluates to a buffer. MATCH forms a string from the contents of the buffer and stops matching when it finds the resulting string.

range

An expression that evaluates to a range. MATCH forms a string from the contents of the range and stops matching when it finds the resulting string.

string

An expression that evaluates to a string. MATCH stops matching when it finds this string.

return value

A variable of type pattern that matches text from the editing point up to and including the characters specified in the parameter.

DESCRIPTION

MATCH returns a pattern that matches any string ending in the specified sequence of characters. The matched string does not contain line terminators.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	MATCH must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	MATCH requires at least one argument.
TPU$_TOOMANY	ERROR	MATCH requires no more than one argument.
TPU$_ARGMISMATCH	ERROR	Argument to MATCH has the wrong type.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of MATCH.

EXAMPLES

1
```
pat1 := MATCH ("abc")
```

This assignment statement stores in *pat1* a pattern that matches a string of characters starting with the editing point up to and including the characters "abc".

2
```
PROCEDURE user_double_parens

   paren_text    := "((' + MATCH ('))");
   found_range   := SEARCH_QUIETLY (paren_text, FORWARD, NO_EXACT);

   IF found_range = 0   ! No match
   THEN
       MESSAGE  ("No match found.");
   ELSE
       POSITION (found_range);
   ENDIF;
ENDPROCEDURE;
```

This procedure finds text within double parentheses. It moves the editing point to the beginning of the parenthesized text, if it is found.

MESSAGE

Depending on the format you choose, either puts the characters that you specify into the message buffer, or else fetches text associated with a message code, formats the text using FAO directives, and puts it in the message buffer.

If you use the first format, MESSAGE inserts the characters in the string, range, or buffer that you specify into the message buffer, if one exists. (By default, VAXTPU looks for a buffer variable that is named MESSAGE_BUFFER.) If there is no message buffer, VAXTPU displays the message at the current location on the device pointed to by SYS$OUTPUT (usually your terminal).

If you use the second format, MESSAGE fetches the text associated with a message code, formats the text using FAO directives, and displays the formatted message in the message buffer. (If there is no message buffer, VAXTPU displays the message on SYS$OUTPUT.)

FORMATS

MESSAGE ({ *buffer* / *range* } [, *integer1*])

MESSAGE ({ *integer2* / *keyword* / *string* / *buffer* } [, *integer3*
[, *FAO-parameter* [, *FAO-parameters...*]]])

PARAMETERS

buffer

The buffer containing the text that you want to include in the message buffer.

range

The range containing the text that you want to include in the message buffer.

integer1

An integer indicating the severity of the message placed in the message buffer. If you do not specify this parameter, no severity code is associated with the message. The allowable integer values and their meanings are as follows:

Integer	Meaning
0	Warning
1	Success

Integer	Meaning
2	Error
3	Informational

integer2
The integer representing the message code associated with the text to be fetched.

keyword
The VAXTPU keyword representing the message code associated with the text to be fetched. VAXTPU provides keywords for all of the message codes used by VAXTPU and EVE.

string
Either a quoted string or a variable representing the text you want to include in the message buffer.

integer3
A bit-encoded integer that specifies what fields of the message text associated with the message code from the first parameter are to be fetched. If the message flags are not specified or the value is zero, then the message flags set by the SET (MESSAGE_FLAGS) built-in procedure are used.

Table 7–6 shows the message flags:

Table 7–6 Message Flag Values

Bit	Constant	Meaning
0	TPU$K_MESSAGE_TEXT	Include text of message.
1	TPU$K_MESSAGE_ID	Include message identifier.
2	TPU$K_MESSAGE_SEVERITY	Include severity level indicator.
3	TPU$K_MESSAGE_FACILITY	Include facility name.

FAO-parameter
One or more expressions that evaluate to an integer or string. The MESSAGE_TEXT built-in procedure uses these integers and strings as arguments to the $FAO system service, substituting the values into the text associated with the message code to form the resultant string.

The FAO directives are listed in the description of $FAO in the *VMS System Services Reference Manual*.

DESCRIPTION

If you use the first format shown above, the MESSAGE built-in provides the user who is writing an editing interface with a method of displaying messages in a way that is consistent with the VAXTPU language.

If you have associated a message buffer with a message window, and if the message window is mapped to the screen, the range you specify appears immediately in the message window on the screen.

If you have not associated a message buffer with a message window, messages are written to the buffer, but do not appear on the screen.

If you use the second format shown above, the MESSAGE built-in places a formatted string in the message buffer. The difference between MESSAGE and MESSAGE_TEXT is that MESSAGE_TEXT simply returns the resulting string while MESSAGE places the resulting string in the message buffer. The string is specified by the message code passed as the first parameter and constructed according to the rules of the $FAO system service. The control string associated with the message code directs the formatting process, and the optional arguments are values to be substituted into the control string.

MESSAGE capitalizes the first character of the string placed in the message buffer. The MESSAGE_TEXT built-in, on the other hand, does not capitalize the first character of the returned string.

Some FAO directives you can include as part of the message text are the following:

!AS	Inserts a string as is
!OL	Converts an integer to octal notation
!XL	Converts an integer to hexadecimal notation
!ZL	Converts an integer to decimal notation
!UL	Converts an integer to decimal notation without adjusting for negative number
!SL	Converts an integer to decimal notation with negative numbers converted properly
!/	Inserts a new line character (carriage return/line feed)
!_	Inserts a tab
!}	Inserts a form feed
!!	Inserts an exclamation point
!%S	Inserts an s if the most recently converted number is not 1
!%T	Inserts the current time if you enter 0 as the parameter (you cannot pass a specific time because VAXTPU does not use quadwords)
!%D	Inserts the current date and time if you enter 0 as the parameter (you cannot pass a specific date because VAXTPU does not use quadwords)

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	MESSAGE requires at least one argument.
TPU$_TOOMANY	ERROR	MESSAGE cannot accept as many arguments as you have specified.
TPU$_ARGMISMATCH	ERROR	You have specified an argument of the wrong type.
TPU$_INVFAOPARAM	WARNING	Argument was not a string or integer.

TPU$_INVPARAM	ERROR	You have specified an argument of the wrong type.
TPU$_FLAGTRUNC	INFORMATIONAL	Message flag truncated to 4 bits.
TPU$_SYSERROR	ERROR	Error fetching the message text.
TPU$_ILLSEVERITY	WARNING	Illegal severity specified; VAXTPU used the severity "error."
TPU$_MSGNOTFND	WARNING	Message not found. VAXTPU returned default message.

EXAMPLES

1
```
MESSAGE ("Hello")
```

This statement writes the text "Hello" in the message area.

2
```
PROCEDURE user_on_eol
! test if at eol, return true or false
   MOVE_HORIZONTAL (1);
   IF CURRENT_OFFSET = 0              ! then we are on eol
   THEN
       user_on_end_of_line := 1;    ! return true
       MESSAGE ("Cursor at end of line");
   ELSE
       user_on_end_of_line := 0;    ! return false
       MESSAGE ("Cursor is not at the end of line");
   ENDIF;
   MOVE_HORIZONTAL (-1);              ! move back
ENDPROCEDURE;
```

This procedure determines whether the cursor is at the end of the line. It sends a text message to the message area on the screen about the position of the cursor.

3
```
MESSAGE (TPU$_OPENIN, TPU$K_MESSAGE_TEXT, "bat.bar");
```

The code fragment above fetches the text associated with the message code TPU$_OPENIN and substitutes the string "BAT.BAR" into the message. All of the text of the message is fetched. The following string is displayed in the message buffer:

```
Error opening BAT.BAR as input
```

MESSAGE_TEXT

The MESSAGE_TEXT built-in procedure lets you do the following:

- Fetch the text associated with a message code
- Use FAO directives to specify how strings and integers should be substituted into the text

For complete information on the $FAO and $GETMSG system services, see the *VMS System Services Reference Manual.*

FORMAT

string := MESSAGE_TEXT ({ *integer1* / *keyword* } [, *integer2* [, *FAO-parameter* [, *FAO-parameters...*]]])

PARAMETERS

integer1

The integer for the message code associated with the text that is to be fetched.

keyword

The keyword for the message code associated with the text that is to be fetched. VAXTPU provides keywords for all of the message codes used by VAXTPU and the EVE editor.

integer2

A bit-encoded integer that specifies what fields of the message text associated with the message code from the first parameter are to be fetched. If the message flags are not specified or the value is 0, then the message flags set by the SET (MESSAGE_FLAGS) built-in procedure are used.

Table 7–7 shows the message flags:

Table 7–7 Message Flag Values

Bit	Constant	Meaning
0	TPU$K_MESSAGE_TEXT	Include text of message.
1	TPU$K_MESSAGE_ID	Include message identifier.
2	TPU$K_MESSAGE_SEVERITY	Include severity level indicator.
3	TPU$K_MESSAGE_FACILITY	Include facility name.

FAO-parameter

One or more expressions that evaluate to an integer or string. The MESSAGE_TEXT built-in procedure uses these integers and strings as arguments to the $FAO system service, and substitutes the resultant values into the text associated with the message code to form the returned string.

return value

The text associated with a message code that is fetched and formatted by MESSAGE_TEXT.

DESCRIPTION

MESSAGE_TEXT returns a formatted string, specified by the message code passed as the first parameter, and constructed according to the rules of the $FAO system service. The control string associated with the message code directs the formatting process, and the optional arguments are values to be substituted into the control string.

MESSAGE_TEXT does not capitalize the first character of the returned string. The MESSAGE built-in, on the other hand, does capitalize the first character.

Some FAO directives you can include as part of the message text are the following:

!AS	Inserts a string as is
!OL	Converts an integer to octal notation
!XL	Converts an integer to hexadecimal notation
!ZL	Converts an integer to decimal notation
!UL	Converts an integer to decimal notation without adjusting for negative number
!SL	Converts an integer to decimal notation with negative numbers converted properly
!/	Inserts a new line character (carriage return/line feed)
!_	Inserts a tab
!}	Inserts a form feed
!!	Inserts an exclamation point
!%S	Inserts an s if the most recently converted number is not 1
!%T	Inserts the current time if you enter 0 as the parameter (you cannot pass a specific time because VAXTPU does not use quadwords)
!%D	Inserts the current date and time if you enter 0 as the parameter (you cannot pass a specific date because VAXTPU does not use quadwords)

SIGNALED ERRORS

TPU$_INVFAOPARAM	WARNING	Argument was not a string or integer.
TPU$_ NEEDTOASSIGN	ERROR	MESSAGE_TEXT must appear on the right-hand side of an assignment statement.
TPU$_INVPARAM	ERROR	You have specified an argument of the wrong type.
TPU$_TOOFEW	ERROR	MESSAGE_TEXT requires at least one parameter.
TPU$_TOOMANY	ERROR	MESSAGE_TEXT accepts up to 20 FAO directives.

TPU$_FLAGTRUNC	INFORMATIONAL	Message flag truncated to 4 bits.
TPU$_SYSERROR	ERROR	Error fetching the message text.

EXAMPLE

```
all_message_flags := TPU$K_MESSAGE_TEXT OR
                     TPU$K_MESSAGE_ID OR
                     TPU$K_MESSAGE_SEVERITY OR
                     TPU$K_MESSAGE_FACILITY;
openin_text := MESSAGE_TEXT (TPU$_OPENIN, all_message_flags,
               "bat.bar");
```

This code fragment fetches the text associated with the message code TPU$_OPENIN and substitutes the string "BAT.BAR" into the message. All of the text of the message is fetched. The following string is stored in the variable *openin_text*:

```
%TPU-E-OPENIN, error opening BAT.BAR as input
```

MODIFY_RANGE

Dynamically modifies a range.

FORMAT

MODIFY_RANGE *(range,* { *marker1* / *keyword1* }*,* { *marker2* / *keyword1* } *[[, keyword2]])*

PARAMETERS

range
The range to be modified.

marker1
The starting mark for the range.

marker2
The ending mark for the range.

keyword1
A keyword indicating the point in the buffer where you want the range to begin or end. Table 7–8 shows the valid keywords and their meanings. Use of the delimiting keywords is more efficient than the BEGINNING_OF and END_OF built-ins.

Table 7–8 MODIFY_RANGE Keyword Parameters

Keyword	Meaning
LINE_BEGIN	The beginning of the current buffer's current line.
LINE_END	The end of the current buffer's current line.
BUFFER_ BEGIN	Line 1, offset 0 in the current buffer. This is the first position where a character could be inserted, regardless of whether there is a character there. This is the same as the point referred to by BEGINNING_OF (CURRENT_BUFFER).
BUFFER_END	The last position in the buffer where a character could be inserted, regardless of whether there is a character there. This is the same as the point referred to by END_OF (CURRENT_BUFFER).

keyword2
A keyword specifying the new video attribute for the range. By default, the attribute is not modified. You can use the keywords NONE, REVERSE, UNDERLINE, BLINK, or BOLD to specify this parameter.

DESCRIPTION

You can use MODIFY_RANGE to specify a new starting mark and ending mark for an existing range.

MODIFY_RANGE can also change the characteristics of the range without deleting, re-creating, and repainting all the characters in the range. Using MODIFY_RANGE, you can direct VAXTPU to apply or remove the range's video attribute to or from characters as you select and unselect text.

Ranges are limited to one video attribute at a time. Specifying a video attribute different from the present attribute causes VAXTPU to apply the new attribute to the entire visible portion of the range.

If the video attribute stays the same and only the markers move, the only characters that are refreshed are those visible characters newly added to the range and those visible characters that are no longer part of the range.

SIGNALED ERRORS

TPU$_NOTSAMEBUF	WARNING	The first and second marker are in different buffers.
TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by the MODIFY_RANGE built-in.
TPU$_BADKEY	WARNING	You specified an illegal keyword.
TPU$_INVPARAM	ERROR	You specified a parameter of the wrong type.
TPU$_MODRANGEMARKS	ERROR	MODIFY_RANGE requires either two marker parameters or none.
TPU$_TOOFEW	ERROR	Too few arguments passed to the MODIFY_RANGE built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the MODIFY_RANGE built-in.
TPU$_NORETURNVALUE	ERROR	MODIFY_RANGE cannot return a value.

EXAMPLES

1

```
begin_mark := MARK (BOLD);
POSITION (MOUSE);
finish_mark := MARK (BOLD);
this_range := CREATE_RANGE (begin_mark, finish_mark, BOLD);
!  .
!  .  (User may have moved mouse)
!  .
POSITION (MOUSE);
new_mark := MARK (BOLD);
IF new_mark <> finish_mark
THEN
    MODIFY_RANGE (this_range, begin_mark, new_mark, BOLD);
ENDIF;
```

This code fragment creates a range between the editing point and the pointer cursor location. At a point in the program after you might have

moved the pointer cursor, the code fragment modifies the range to reflect the new pointer cursor location.

2
```
MODIFY_RANGE (this_range, , ,BLINK);
```

This statement sets the video attribute of the range *this_range* to BLINK.

3
```
PROCEDURE move_mark (place_to_start, direction);
    POSITION (place_to_start);
    IF direction = 1
    THEN
        MOVE_HORIZONTAL (1);
    ELSE
        MOVE_HORIZONTAL (-1);
    ENDIF;
    RETURN MARK (NONE);
ENDPROCEDURE;
PROCEDURE user_shrink_and_enlarge_range
    LOCAL  start_mark,
           end_mark,
           direction,
           dynamic_range,
           rendition,
           remembered_range;
                                          ! The following lines
                                          ! create a range that
                                          ! shrinks and grows and
                                          ! a range that defines
                                          ! the limits of the dynamic
                                          ! range.
    POSITION (LINE_BEGIN);
    start_mark := MARK (NONE);
    POSITION (LINE_END);
    end_mark := MARK (NONE);
    rendition := REVERSE;
    remembered_range := CREATE_RANGE (start_mark, end_mark, NONE);
    dynamic_range := CREATE_RANGE (start_mark, end_mark, rendition);
                                           ! The following lines
                                           ! shrink and enlarge
                                           ! the dynamic range.
    direction := 1;
    LOOP
        UPDATE (CURRENT_WINDOW);
        start_mark := move_mark (BEGINNING_OF (dynamic_range), direction);
        end_mark := move_mark (END_OF (dynamic_range), 1 - direction);
        MODIFY_RANGE (dynamic_range, start_mark, end_mark);
```

```
        IF start_mark > end_mark
        THEN
            EXITIF READ_KEY = CTRL_Z_KEY;
            direction := 0;
            IF rendition = REVERSE
            THEN
                rendition := BOLD;
            ELSE
                rendition := REVERSE;
            ENDIF;
            MODIFY_RANGE (dynamic_range, , , rendition);
        ENDIF;

        IF (start_mark = BEGINNING_OF (remembered_range)) OR
           (end_mark = END_OF (remembered_range))
        THEN
            direction := 1;
        ENDIF;
    ENDLOOP;
ENDPROCEDURE;
```

These procedures cause the range *dynamic_range* to shrink to one character, then grow until it becomes as large as the range *remembered_range*.

4
```
PROCEDURE line_up_characters (text_range, lined_chars_pat)
LOCAL
    range_start,
    range_end,
    temp_range,
    max_cols;


range_end := END_OF (text_range);           ! These statements store
                                            ! the ends of the range
                                            ! containing the text operated on.
range_start := BEGINNING_OF (text_range);

                                          ! The following statements
                                          ! locate the portions of
                                          ! text that match the pattern
                                          ! and determine which is
                                          ! furthest to the right.
max_cols := 0;
LOOP
    temp_range := SEARCH_QUIETLY (lined_chars_pat, REVERSE, EXACT, text_range);
    EXITIF temp_range = 0;
    POSITION (temp_range);
    IF GET_INFO (MARK (NONE), "offset_column") > max_cols
    THEN
        max_cols := GET_INFO (MARK (NONE), "offset_column");
    ENDIF;
    MOVE_HORIZONTAL (-1);
    MODIFY_RANGE (text_range, BEGINNING_OF (text_range), MARK (NONE));
ENDLOOP;
```

```
                                                        ! The following lines
                                                        ! locate matches to the
text_range := CREATE_RANGE (range_start, range_end);    ! pattern and align them
                                                        ! with the rightmost
                                                        ! piece of matching text.
LOOP
    temp_range := SEARCH_QUIETLY (lined_chars_pat, FORWARD, EXACT, text_range);
    EXITIF temp_range = 0;
    POSITION (temp_range);
    IF GET_INFO (MARK (NONE), "offset_column") < max_cols
    THEN
        COPY_TEXT (" " * (max_cols - GET_INFO (MARK (NONE), "offset_column")));
    ENDIF;
    MOVE_HORIZONTAL (1);
    MODIFY_RANGE (text_range, END_OF (text_range), MARK (NONE));
ENDLOOP;

!
! Restore the range to its original state, plus a reverse attribute.
!
text_range := CREATE_RANGE (range_start, range_end, REVERSE); ! This line
                                                              ! restores the
                                                              ! range to its
                                                              ! original state
                                                              ! and displays
                                                              ! the contents
                                                              ! in reverse video.

ENDPROCEDURE;
```

This procedure aligns text that conforms to the pattern specified in the second parameter. For example, if you want to align all comments in a piece of VAXTPU code, you would pass as the second parameter a pattern defined as an exclamation point followed by an arbitrary amount of text or whitespace and terminated by a line end.

The procedure is passed a range of text. As the procedure searches the range to identify the rightmost piece of text that matches the pattern, the procedure modifies the range to exclude any matching text. Next, the procedure searches the original range again and inserts padding spaces in front of each instance of matching text, making the text align with the rightmost instance of matching text.

MOVE_HORIZONTAL

Changes the editing point in the current buffer by the number of characters you specify.

FORMAT

MOVE_HORIZONTAL *(integer)*

PARAMETERS

integer

The signed integer value that indicates the number of characters the editing point should be moved. A positive integer specifies movement toward the end of the buffer. A negative integer specifies movement toward the beginning of the buffer.

VAXTPU does not count the column where the editing point is located when determining where to establish the new editing point. VAXTPU does count the end-of-line (the column after the last text character on the line) when determining where to establish the new editing point.

DESCRIPTION

The horizontal adjustment of the editing point is tied to text. MOVE_HORIZONTAL crosses line boundaries to adjust the current character position.

You cannot see the adjustment caused by MOVE_HORIZONTAL unless the current buffer is mapped to a visible window. If it is, VAXTPU scrolls text in the window, if necessary, so that the editing point you establish with MOVE_HORIZONTAL is within the scrolling limits set for the window.

If you try to move past the beginning or the end of a buffer, VAXTPU displays a warning message.

Using MOVE_HORIZONTAL may cause VAXTPU to insert padding spaces or blank lines in the buffer. MOVE_HORIZONTAL causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	MOVE_HORIZONTAL requires one parameter.
TPU$_TOOMANY	ERROR	You specified more than one parameter.

TPU$_INVPARAM	ERROR	The specified parameter has the wrong type.
TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.
TPU$_ENDOFBUF	WARNING	You are trying to move forward past the last character of the buffer.
TPU$_BEGOFBUF	WARNING	You are trying to move in reverse past the first character of the buffer.

EXAMPLES

1
```
MOVE_HORIZONTAL (+5)
```

This statement moves the editing point five characters toward the end of the current buffer.

2
```
PROCEDURE user_move_by_lines
   IF CURRENT_DIRECTION = FORWARD
   THEN
       MOVE_VERTICAL (8)
   ELSE
       MOVE_VERTICAL(- 8)
   ENDIF;
   MOVE_HORIZONTAL (-CURRENT_OFFSET);
ENDPROCEDURE;
```

This procedure moves the editing point by sections that are eight lines long, and uses MOVE_HORIZONTAL to put the editing point at the beginning of the line.

MOVE_TEXT

Depending on the mode of the current buffer, moves the text you specify and inserts or overwrites it in the current buffer. When you move text with range and buffer parameters, you remove it from its original location. For information on how to copy text instead of removing it, see the description of the COPY_TEXT built-in.

FORMAT

[range2 :=]MOVE_TEXT ({ *buffer* / *range1* / *string* })

PARAMETERS

buffer

The buffer from which text is moved.

range1

The range from which text is moved.

string

A string representing the text you want to move. Text is not removed from its original location with this argument.

return value

The range where the copied text has been placed.

DESCRIPTION

If the current buffer is in insert mode, the text you specify is inserted before the editing point in the current buffer. If the current buffer is in overstrike mode, the text you specify replaces text starting at the current position and continuing for the length of the string, range, or buffer.

Markers and ranges are not moved with the text. If the text of a marker or a range is moved, the marker or range structure and any video attribute that you specified for the marker or range are moved to the next closest character, which is always the character following the marker or range. To remove the marker or range structure, use the built-in procedure DELETE or set the variable to which the range is assigned to 0.

MOVE_TEXT is similar to COPY_TEXT. However, MOVE_TEXT erases the text from its original string, range, or buffer, while COPY_TEXT just makes a copy of the text and places the copy at the new location.

You cannot add a buffer or a range to itself. If you try to do so, VAXTPU issues an error message. If you try to insert a range into itself, part of the range is copied before VAXTPU signals an error. If you try to overstrike a range into itself, VAXTPU may or may not signal an error.

Using MOVE_TEXT may cause VAXTPU to insert padding spaces or blank lines in the buffer. MOVE_TEXT causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

SIGNALED ERRORS

TPU$_NOCACHE	ERROR	There is not enough memory to allocate a new cache.
TPU$_TOOFEW	ERROR	MOVE_TEXT requires one argument.
TPU$_TOOMANY	ERROR	MOVE_TEXT accepts only one argument.
TPU$_ARGMISMATCH	ERROR	The argument to MOVE_TEXT must be a buffer, range, or string.
TPU$_NOTMODIFIABLE	ERROR	You cannot copy text into an unmodifiable buffer.
TPU$_MOVETOCOPY	WARNING	MOVE_TEXT was able to copy the text into the current buffer but could not delete it from the source buffer because the source buffer is unmodifiable.

EXAMPLES

1
```
MOVE_TEXT (main_buffer)
```

If you are using insert mode for text entry, this statement causes the text from *main_buffer* to be placed in front of the current position in the current buffer. The text is removed from *main_buffer*.

2
```
PROCEDURE user_move_text
   LOCAL this_mode;
! Save mode of current buffer in this_mode
   this_mode := GET_INFO (CURRENT_BUFFER, "mode");
! Set current buffer to insert mode
   SET (INSERT, CURRENT_BUFFER);
! Move the scratch buffer text to the current buffer
   MOVE_TEXT (scratch_buffer);
! Reset current buffer to original mode
   SET (this_mode, CURRENT_BUFFER);
ENDPROCEDURE;
```

This procedure puts the text from the scratch buffer before the editing point in the main buffer. The text in the scratch buffer is removed; no copy of it is left there.

MOVE_VERTICAL

Modifies the editing point in the current buffer by the number of lines you specify.

FORMAT

MOVE_VERTICAL *(integer)*

PARAMETERS

integer

The signed integer value that indicates the number of lines that the editing point should be moved. A positive integer specifies movement toward the end of the buffer. A negative integer specifies movement toward the beginning of the buffer.

DESCRIPTION

The adjustment that MOVE_VERTICAL makes is tied to text. VAXTPU tries to retain the same character offset relative to the beginning of the line when moving vertically. However, if there are tabs in the lines, or the lines have different margins, the editing point does not necessarily retain the same column position on the screen.

By default, VAXTPU keeps the cursor at the same offset on each line. However, since VAXTPU counts a tab as one character regardless of how wide the tab is, the cursor's column position may vary greatly even though the offset is the same.

To keep the cursor in approximately the same column on each line, use the following statement:

```
SET (COLUMN_MOVE_VERTICAL, ON)
```

This statement directs VAXTPU to keep the cursor in the same column unless a tab character makes this impossible. If a tab occupies the column position, VAXTPU moves the cursor to the beginning of the tab.

You cannot see the adjustment caused by MOVE_VERTICAL unless the current buffer is mapped to a visible window. If it is, VAXTPU scrolls text in the window, if necessary, so that the editing point you establish with MOVE_VERTICAL is within the scrolling limits set for the window.

Using MOVE_VERTICAL may cause VAXTPU to insert padding spaces or blank lines in the buffer. MOVE_VERTICAL causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

If you try to move past the beginning or end of a buffer, VAXTPU displays a warning message.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	MOVE_VERTICAL requires at least one parameter.
TPU$_TOOMANY	ERROR	You specified more than one parameter.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BEGOFBUF	WARNING	You are trying to move backward past the first character of the buffer.
TPU$_ENDOFBUF	WARNING	You are trying to move forward past the last character of the buffer.
TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.

EXAMPLES

1
```
MOVE_VERTICAL (+5)
```

This statement moves the editing point in the current buffer down five lines toward the end of the buffer.

2
```
PROCEDURE user_move_8_lines
   IF CURRENT_DIRECTION = FORWARD
   THEN
       MOVE_VERTICAL (8);
   ELSE
       MOVE_VERTICAL (- 8);
   ENDIF;
   MOVE_HORIZONTAL(- CURRENT_OFFSET);
ENDPROCEDURE;
```

This procedure moves the editing point by sections that are eight lines long.

NOTANY

Returns a pattern that matches a specific number of characters not in the string, buffer, or range that is used as a parameter.

FORMAT

pattern := NOTANY ({ *buffer* / *range* / *string* } [, *integer1*])

PARAMETERS

buffer

An expression that evaluates to a buffer. NOTANY matches any character not in the resulting buffer.

range

An expression that evaluates to a range. NOTANY matches any character not in the resulting range.

string

An expression that evaluates to a string. NOTANY matches any character not in the resulting string.

integer1

This integer value indicates how many contiguous characters NOTANY matches. The default value for this integer is 1.

return value

A pattern that matches characters not in the string, buffer, or range used as a parameter.

DESCRIPTION

NOTANY returns a pattern that matches one or more contiguous characters. NOTANY only matches characters that do not appear in the string, range, or buffer used as the first parameter. The second parameter determines the number of characters NOTANY must match. NOTANY does not match across line breaks.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	NOTANY must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	NOTANY requires at least one argument.
TPU$_TOOMANY	ERROR	NOTANY accepts no more than two arguments.
TPU$_ARGMISMATCH	ERROR	NOTANY was given an argument of the wrong type.

TPU$_INVPARAM	ERROR	NOTANY was given an argument of the wrong type.
TPU$_MINVALUE	WARNING	NOTANY was given an argument less than the minimum value.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of NOTANY.

EXAMPLES

1
```
pat1 := NOTANY ("XYZ")
```

This assignment statement creates a pattern that matches the first character that is not an X, a Y, or a Z. The match fails if no character other than X, Y, or Z is found.

2
```
 pat1 := notany ("ABC", 2)
```

This assignment statement creates a pattern that matches two characters, neither of which can be an A, a B, or a C.

3
```
a_buf := CREATE_BUFFER ("new buffer");
POSITION (a_buf);
COPY_TEXT ("xy");
SPLIT_LINE;
COPY_TEXT ("abc");
pat1 := NOTANY (a_buf);
```

These VAXTPU statements create a pattern that matches any single character other than one of the characters a, b, c, x, and y.

4
```
!
! The following procedure returns a marker pointing to
! the next nonalphabetic character or the integer zero
! if there are no more nonalphabetic characters.  You
! call the procedure in the following way:
!
!   non_alpha_marker := user_search_for_nonalpha;

PROCEDURE user_search_for_nonalpha

   LOCAL pat,
         first_non_alpha;

   pat := NOTANY ("abcdefghijklmnopqrstuvwxyz");

   first_non_alpha := SEARCH_QUIETLY (pat, FORWARD, NO_EXACT);

   IF first_non_alpha <> 0
   THEN
       first_non_alpha := BEGINNING_OF (first_non_alpha);
   ENDIF;

   RETURN first_non_alpha;
ENDPROCEDURE;
```

This procedure starts at the current location and looks for the first nonalphabetic, nonlowercase character. The variable *non_alpha_range* stores the character that matches these conditions.

PAGE_BREAK

Specifies the form-feed character, ASCII(12), as a portion of a pattern to be matched.

FORMAT

PAGE_BREAK

PARAMETERS

None.

DESCRIPTION

PAGE_BREAK matches the next form-feed character. This character has an ASCII value of 12.

Although PAGE_BREAK behaves much like a built-in, it is actually a keyword.

If the form-feed character is the only character on a line, PAGE_BREAK matches the whole line. If the form-feed character is not the only character on a line, PAGE_BREAK matches only the form-feed character.

SIGNALED ERROR

PAGE_BREAK is a keyword and has no completion codes.

EXAMPLE

```
PROCEDURE user_next_page
   LOCAL next_page;
   next_page := SEARCH_QUIETLY (PAGE_BREAK, FORWARD);
   IF next_page <> 0
   THEN
       POSITION (next_page);
   ELSE
       POSITION (end_of (current_buffer));
   ENDIF;
ENDPROCEDURE;
```

This procedure places the cursor on the next page in the current buffer. If you are already on the last page of a document, it places the cursor at the end of that document.

POSITION

Ties the editing point to a specific character in a specific buffer, and moves the editing point to a specified record in the current buffer. The character and buffer in which POSITION establishes the editing point depend on which parameter you pass to POSITION.

FORMAT

POSITION { *buffer* | *BUFFER_BEGIN* | *BUFFER_END* | *integer* | *LINE_BEGIN* | *LINE_END* | *marker* | *MOUSE* | *range* | *TEXT* | *window* }

PARAMETERS

buffer

The buffer in which you want to establish the editing point.

VAXTPU maintains an editing point in each buffer even when the buffer is not the current buffer. When you position to a buffer, the editing point that VAXTPU maintains becomes the active editing point. The location at which POSITION establishes the editing point is the last character that the cursor was on when the buffer was most recently current.

BUFFER_BEGIN

A keyword directing VAXTPU to establish the editing point at the beginning of the current buffer. Note that this is more efficient than using POSITION (BEGINNING_OF (CURRENT_BUFFER)).

BUFFER_END

A keyword directing VAXTPU to establish the editing point at the end of the current buffer. Note that this is more efficient than using POSITION (END_OF (CURRENT_BUFFER)).

integer

The number of the record where you want VAXTPU to position the editing point.

A record number indicates the location of a record in a buffer. Record numbers are dynamic; as you add or delete records, VAXTPU changes the number associated with a particular record, as appropriate. VAXTPU counts each record in a buffer, regardless of whether the line is visible in a window, or whether the record contains text.

To position the editing point to a given record, specify the record number. The number can be in the range from 1 to the number of records in the buffer plus 1. For example, the following statement positions the editing point to record number 8 in the current buffer:

```
POSITION (8);
```

VAXTPU places the editing point on the first character of the record.

Specifying a value of 0 has no effect. Specifying a negative number or a number greater than the number of records in the buffer plus 1 causes VAXTPU to signal an error.

LINE_BEGIN

A keyword directing VAXTPU to establish the editing point at the beginning of the current line.

LINE_END

A keyword directing VAXTPU to establish the editing point at the end of the current line.

marker

The marker to which you want to tie the editing point. You can position either to a bound marker or a free marker. (For more information on the distinction between bound and free markers, see Chapter 2.) Positioning to a free marker does not cause VAXTPU to insert padding blanks between the nearest text and the free marker; such positioning establishes the editing point as free. (For more information on the distinction between free and detached editing points, see Chapter 6.)

MOUSE

A keyword directing VAXTPU to associate the editing point with the location of the pointer cursor.

In DECwindows VAXTPU, you can use the statement POSITION (MOUSE) at any point after the first keyboard or mouse button event. The statement positions the editing point to the location occupied by the pointer cursor at the time of the most recent keyboard or mouse-button event.

If the pointer cursor is on a window's status line when POSITION (MOUSE) is executed, VAXTPU positions the editing point at the line just above the status line.

If the pointer cursor is not located in a VAXTPU window at the time of the most recent keyboard or mouse-button event, POSITION (MOUSE) returns the status TPU$_NOWINDOW.

In non-DECwindows VAXTPU, POSITION (MOUSE) is only valid during a procedure that is executed as the result of a mouse click. At all other times, the mouse position is not updated.

The statement POSITION (MOUSE) makes the window in which the pointer cursor is located the current window, and the buffer in which the pointer cursor is located the current buffer.

range

The range in which you want to place the editing point. The editing point is established at the beginning of the range. To establish the editing point at the end of the range, use the statement POSITION (END_OF (range)).

TEXT

A keyword indicating that if the editing point is at a free-cursor location (a portion of the screen where there is no text), the POSITION built-in is to establish the editing point at the nearest location that has a text character in it. The character may be a space or an end of line. If you use POSITION (TEXT) when the editing point is already bound to a character, the built-in has no effect.

window

The window in which you want to establish the editing point. The window must be mapped to the screen.

The location at which POSITION establishes the editing point is the last character that the cursor was on when the window was most recently current. If that character has been deleted, the editing point is the character closest to the last character that the cursor was on when the window was current.

Positioning to a window causes the buffer associated with the window to become the current buffer. This is true whether you directly position to a window, or a new window is mapped as the result of a POSITION (MOUSE) statement.

DESCRIPTION

The editing point is the location in the current buffer where most editing operations are carried out. VAXTPU maintains a marker pointing to an editing point in each buffer, but only the editing point in the current buffer is active. An editing point, whose location is always tied to a character in a buffer, is not necessarily the same as the cursor position, whose location is always tied to a position in a window. For more information on the distinction between the editing point and the cursor position, see Chapter 6.

The POSITION built-in synchronizes the editing point and the cursor position if the current buffer is mapped to a visible window. POSITION also moves the editing point to the the specified record in the current buffer.

When you pass the keyword MOUSE to POSITION, the built-in establishes the mouse pointer's location as the cursor position. POSITION also establishes the window in which the mouse pointer is located as the current window, and establishes the buffer mapped to that window as the current buffer.

Positioning to a buffer, a marker, or a range does not necessarily move the cursor. VAXTPU does not change the cursor position unless the cursor is in a window that is mapped to the buffer specified or implied by the POSITION parameter. For example, if you use POSITION to establish the editing point in a buffer that is not mapped to a window, the cursor is unaffected by the POSITION operation. If you want to do visible editing, you should position to a window rather than a buffer.

If you try to position to an invisible window, VAXTPU issues a warning message.

For more information on the relationship between the editing point and the cursor position, see Chapter 6.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	POSITION requires one parameter.
TPU$_TOOMANY	ERROR	You specified more than one parameter.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to the built-in.
TPU$_BADKEY	WARNING	You have specified an invalid keyword.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.
TPU$_BADVALUE	ERROR	You specified a record number less than 0 or greater than the length of the buffer plus 1.
TPU$_MOUSEINV	WARNING	The mouse position is not currently valid.
TPU$_NOWINDOW	WARNING	The pointer cursor was not located in a VAXTPU window at the time of the most recent keyboard or mouse-button event.
TPU$_WINDNOTMAPPED	WARNING	Window is not mapped to the screen.
TPU$_WINDNOTVIS	WARNING	Window is totally occluded.

EXAMPLES

1
```
POSITION (message_window)
```

This statement establishes the editing point in the message window. Your position in the window is the same character position you occupied when you were last positioned in the window.

2
```
user_mark := MARK(NONE);
POSITION (user_mark)
```

These statements establish the editing point at the marker associated with the variable *user_mark*.

3
```
PROCEDURE user_change_windows
   IF CURRENT_WINDOW = main_window
   THEN
       POSITION (extra_window);
   ELSE
       POSITION (main_window);
   ENDIF;
ENDPROCEDURE;
```

This procedure toggles the active editing point between two windows.

QUIT

Leaves the editor without writing to a file.

FORMAT

QUIT *[({ ON / OFF } [, severity])]*

PARAMETERS

ON

A keyword indicating that VAXTPU should prompt to find out if the user really wants to quit with modified buffers. This is the default value.

OFF

A keyword indicating that VAXTPU should quit without asking the user whether to quit with modified buffers.

severity

If present, the least significant two bits of this integer are used as the severity of the status VAXTPU returns to whatever invoked it.

Value	Severity
0	Warning
1	Success
2	Error
3	Informational

It is not possible to force VAXTPU to return a fatal severity status.

DESCRIPTION

If you modify any buffers that are not set to NO_WRITE and you do not specify OFF as the first parameter to the QUIT built-in procedure, VAXTPU tells you that you have modified buffers and asks whether you want to quit. Enter Y (Yes) if you want to quit without writing out any modified buffers. Enter N (No) if you want to retain the modifications you have made and return to the editor. If you specify OFF as the first parameter to QUIT, VAXTPU quits without informing you that you have modified buffers. All modifications are lost because VAXTPU does not write out buffers when quitting.

Journal files (if any) are deleted upon quitting.

Use the EXIT built-in procedure when you have made changes and want to save them when you leave the editor. (For more information, see the description of EXIT.)

Normally, when VAXTPU quits it returns a status of TPU$_QUITTING to whatever invoked it. This is a success status.

This feature is useful if you are using VAXTPU to create an application in which quitting, especially before the end of a series of statements executing in batch mode, is an error.

A special use of the built-in procedure QUIT is at the end of your section file when you are compiling it for the first time. See Chapter 4 for information on creating section files.

SIGNALED ERRORS

TPU$_CANCELQUIT	WARNING	"NO" response was received from *". . . continue quitting?"* prompt.
TPU$_TOOMANY	ERROR	QUIT accepts no more than two arguments.
TPU$_INVPARAM	ERROR	One of the arguments to QUIT has the wrong data type.
TPU$_BADKEY	WARNING	QUIT accepts only the keywords ON and OFF.

EXAMPLES

1
```
QUIT;
```

This returns control of execution from an editor layered on VAXTPU to the program, application, or operating system that called VAXTPU. If you have modified any buffers, you see the following prompt:

```
Buffer modifications will not be saved, continue quitting (Y or N)?
```

Enter Yes if you want to quit and not save the modifications. Enter No if you want to return to the editor.

2
```
QUIT (OFF)
```

This returns control of execution from an editor layered on VAXTPU to the program, application, or operating system that called VAXTPU. VAXTPU does not alert you if you have modified buffers. All modifications since the last time you wrote out the buffer are discarded.

3
```
PROCEDURE user_quit
   SET (SUCCESS, OFF);
   QUIT;

! Turn message back on in case user answers "No" to the
! prompt "Buffer modifications will not be saved, continue
! quitting (Y or N)?"

   SET (SUCCESS, ON);
ENDPROCEDURE;
```

This procedure turns off the display of the success message, "Editor successfully quitting", when you use the built-in procedure QUIT to leave an editing interface.

READ_CHAR

Stores the next character entered from the keyboard in a string variable.

FORMAT

string := READ_CHAR

PARAMETERS

None.

return value

A variable of type string containing a character entered from the keyboard.

DESCRIPTION

The character read by READ_CHAR is not echoed on the screen; therefore, the cursor position does not move.

READ_CHAR does not process escape sequences. If a VAXTPU procedure uses READ_CHAR for an escape sequence, only part of the escape sequence is read. The remaining part of the escape sequence is treated as text characters. If control then returns to VAXTPU, or a READ_KEY or READ_LINE built-in procedure is executed, the results may be unpredictable.

In DECwindows VAXTPU, READ_CHAR maps the main window if it is not already mapped.

In the DECwindows environment, READ_CHAR cannot read a keypad or function key. If a VAXTPU procedure uses READ_CHAR and you press a keypad or function key, READ_CHAR returns a null string and signals the warning TPU$_NOCHARREAD.

If you invoke VAXTPU with the /NODISPLAY qualifier, do not use READ_CHAR during the session. READ_CHAR causes VAXTPU to abort when VAXTPU is running in NODISPLAY mode.

SIGNALED ERRORS

TPU$_NOCHARREAD	WARNING	READ_CHAR did not read a character.
TPU$_NEEDTOASSIGN	ERROR	READ_CHAR must be on the right-hand side of an assignment statement.
TPU$_TOOMANY	ERROR	READ_CHAR takes no arguments.

EXAMPLES

1

```
new_char := READ_CHAR
```

This assignment statement stores the next character that is entered on the keyboard in the string *new_char*.

```
2   PROCEDURE user_quote
       COPY_TEXT (READ_CHAR);
    ENDPROCEDURE;
```

This procedure enters the next character that is entered from the keyboard in the current buffer. If a key that sends an escape sequence is pressed, the first character of the escape sequence is copied into the buffer. Subsequent keystrokes are interpreted as self-inserting characters, defined keys, or undefined keys, as appropriate.

READ_CLIPBOARD

Reads string format data from the clipboard and copies it into the current buffer, at the editing point, using the buffer's current text mode (insert or overstrike).

FORMAT

[range | UNSPECIFIED] := READ_CLIPBOARD

return value

A range containing the text copied into the current buffer, or an unspecified data type indicating that no data was obtained from the clipboard.

DESCRIPTION

If VAXTPU finds a line-feed character in the data, it removes the line feed and any adjacent carriage returns and puts the data after the line feed on the next line of the buffer. If VAXTPU must truncate the data from the clipboard, VAXTPU copies the truncated text into the current buffer.

All text read from the clipboard is copied into the buffer starting at the editing point. If VAXTPU must start a new line to fit all the text into the buffer, the new line starts at column 1, even if the current left margin is not set at column 1.

SIGNALED ERRORS

TPU$_CLIPBOARDLOCKED	WARNING	VAXTPU cannot read from the clipboard because some other application has locked it.
TPU$_CLIPBOARDNODATA	WARNING	There is no string format data in the clipboard.
TPU$_CLIPBOARDFAIL	WARNING	The clipboard has not returned any data.
TPU$_REQUIRESDECW	ERROR	You can use the READ_CLIPBOARD built-in only if you are using DECwindows TPU.
TPU$_STRTOOLARGE	ERROR	The amount of data in the clipboard exceeds 65,535 characters.
TPU$_TOOMANY	ERROR	Too many arguments passed to the READ_CLIPBOARD built-in.

EXAMPLE

```
PROCEDURE eve$$insert_clipboard

ON_ERROR
    [TPU$_CLIPBOARDNODATA]:
        eve$message (EVE$_NOINSUSESEL);
        eve$learn_abort;
        RETURN (FALSE);
    [TPU$_CLIPBOARDLOCKED]:
        eve$message (EVE$_CLIPBDREADLOCK);
        eve$learn_abort;
        RETURN (FALSE);
    [TPU$_TRUNCATE]:
    [OTHERWISE]:
        eve$learn_abort;
ENDON_ERROR;

IF eve$test_if_modifiable (CURRENT_BUFFER)
THEN
    READ_CLIPBOARD;                         ! This statement using
                                            ! READ_CLIPBOARD reads
                                            ! data from the clipboard
                                            ! and copies it into the
                                            ! current buffer.
    RETURN (TRUE);
ENDIF;

eve$learn_abort;
RETURN (FALSE);

ENDPROCEDURE;
```

This procedure shows one possible way that an application can use the READ_CLIPBOARD built-in. This procedure is a modified version of the EVE procedure EVE$$INSERT_CLIPBOARD. You can find the original version in SYS$EXAMPLES:EVE$DECWINDOWS.TPU.

Procedure EVE$$INSERT_CLIPBOARD fetches the contents of the clipboard and places them in the current buffer.

READ_FILE

Reads a file and inserts the contents of the file immediately before the current line in the current buffer. READ_FILE optionally returns a string containing the file specification of the file read.

FORMAT

[string2 :=] READ_FILE *(string1)*

PARAMETER

string1
A quoted string, a variable name representing a string constant, or an expression that evaluates to a string, that is the name of the file you want to read and include in the current buffer.

return value

A string that is the specification of the file read.

DESCRIPTION

If the current buffer is mapped to a visible window, the READ_FILE built-in causes the screen manager to synchronize the editing point, which is a buffer location, with the cursor position, which is a window location. This may result in the insertion of padding spaces or lines into the buffer if the cursor position is before the beginning of a line, in the middle of a tab, beyond the end of a line, or after the last line in the buffer.

VAXTPU writes a message indicating how many records (lines) were read.

If you try to read a file containing lines longer than 32767 characters, VAXTPU truncates the line to the first 32767 characters and issues a warning.

SIGNALED ERRORS

TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.
TPU$_CONTROLC	ERROR	The execution of the read terminated because you pressed CTRL/C.
TPU$_NOCACHE	ERROR	There is not enough memory to allocate a new cache.
TPU$_TOOFEW	ERROR	READ_FILE requires at least one parameter.
TPU$_TOOMANY	ERROR	READ_FILE accepts no more than one parameter.
TPU$_INVPARAM	ERROR	The parameter to READ_FILE must be a string.
TPU$_TRUNCATE	WARNING	One of the lines in the file was too long to fit in a VAXTPU buffer.

The following errors, warnings, and messages can be signaled by VAXTPU's file I/O routine. You can provide your own file I/O routine by using VAXTPU's callable interface. If you do so, READ_FILE's signaled errors, warnings, and messages depend upon what status you signaled in your file I/O routine.

TPU$_OPENIN	ERROR	READ_FILE could not open the file you specified.
TPU$_READERR	ERROR	READ_FILE did not finish reading the file because it encountered a file system error.
TPU$_CLOSEIN	ERROR	READ_FILE did not finish closing the file because it encountered a file system error.

EXAMPLES

1
```
READ_FILE ("login.com")
```

This statement reads the file LOGIN.COM and adds it to your current buffer.

2
```
PROCEDURE user_two_windows

   w := CREATE_WINDOW (1, 10, ON);
   b := CREATE_BUFFER ("buf2");

   MAP (w, b);

   READ_FILE (READ_LINE ("Enter file name for 2nd window : "));

   POSITION (BEGINNING_OF (b));

   DEFINE_KEY ("POSITION (w)", KEY_NAME ("W", SHIFT_KEY));
ENDPROCEDURE;
```

This procedure creates a second window and a second buffer and maps the window to the screen. The procedure also prompts the user for a file name to include in the buffer and defines the key sequence SHIFT/W as the sequence with which to move to the second window. (The default shift key is PF1.)

READ_GLOBAL_SELECT

Requests information about the specified global selection from the owner of the global selection. If the owner provides the information, READ_GLOBAL_SELECT reads it and copies it into the current buffer at the editing point, using the buffer's current text mode (insert or overstrike). The READ_GLOBAL_SELECT built-in also puts line breaks in the text copied into the buffer.

FORMAT

[{ **unspecified** / **range** } :=] *READ_GLOBAL_SELECT* ({ *PRIMARY* / *SECONDARY* / *selection_name* }, *selection_property_name*)

PARAMETERS

PRIMARY

A keyword indicating that the application is requesting information about a property of the primary global selection.

SECONDARY

A keyword indicating that the application is requesting information about a property of the secondary global selection.

selection_name

A string identifying the global selection whose property is the subject of the application's information request. Specify the selection name as a string if the layered application needs information about a selection other than the primary or secondary global selection.

selection_property_name

A string specifying the property whose value the application is requesting.

return value

unspecified	A data type indicating that the information requested by the application was not available.
range	A range containing the text copied into the current buffer.

DESCRIPTION

Use READ_GLOBAL_SELECT to ask the application that owns the specified global selection for information about a property of the global selection. For example, you can ask about the global selection's font, the number of lines it contains, or the string-formatted data it contains, if any.

All text read from the global selection is copied into the current buffer starting at the editing point. If VAXTPU must start a new line to fit all the text into the buffer, the new line starts at column 1, even if the current left margin is not set at column 1.

If the global selection information requested is an integer, the built-in converts the integer into a string before copying it into the current buffer. If the information requested is a string, the built-in copies the string into the buffer, replacing any line feeds with line breaks. Carriage returns adjacent to line feeds are not copied into the buffer.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_GBLSELOWNER	WARNING	VAXTPU owns the global selection.
TPU$_INVGBLSELDATA	WARNING	The global selection owner provided data that VAXTPU cannot process.
TPU$_NOGBLSELDATA	WARNING	The global selection owner has indicated that it cannot provide the information requested.
TPU$_NOGBLSELOWNER	WARNING	You have requested information about an unowned global selection.
TPU$_TIMEOUT	WARNING	The global selection owner did not respond before the timeout period expired.
TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to the READ_GLOBAL_SELECT built-in.
TPU$_REQUIRESDECW	ERROR	You can use the READ_GLOBAL_SELECT built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the READ_GLOBAL_SELECT built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the READ_GLOBAL_SELECT built-in.

EXAMPLE

```
READ_GLOBAL_SELECTION (PRIMARY, "STRING");
```

This statement reads the string-formatted contents of the primary global selection and copies it into the current buffer at the current location.

For another example of code using the READ_GLOBAL_SELECT built-in, see Example B–9.

READ_KEY

Waits for you to press a key and then returns the key name for that key.

FORMAT

keyword := READ_KEY

PARAMETERS

None.

return value

A key name for the key just pressed.

DESCRIPTION

The READ_KEY built-in procedure should be used rather than READ_CHAR when you are entering escape sequences, control characters, or any characters other than text characters. READ_KEY processes escape sequences and VAXTPU's shift key (PF1 by default).

The key that is read by READ_KEY is not echoed on the terminal screen.

In DECwindows VAXTPU, READ_KEY maps the main window if it is not already mapped.

If you invoke VAXTPU with the /NODISPLAY qualifier, do not use READ_KEY during the session. READ_KEY causes VAXTPU to abort when VAXTPU is running in NODISPLAY mode.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	READ_KEY must be on the right-hand side of an assignment statement.
TPU$_TOOMANY	ERROR	READ_KEY accepts no arguments.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of READ_KEY.
TPU$_REQUIRESTERM	ERROR	You cannot use READ_KEY when VAXTPU is in NODISPLAY mode.

EXAMPLES

1
```
my_key := READ_KEY
```

This assignment statement reads the next key that is entered and stores the keyword for that key in the variable *my_key*.

```
2  PROCEDURE user_help_on_key
      LOCAL key_pressed,
            key_comment;
      MESSAGE ("Press the key you want help on.");
      key_pressed := READ_KEY;
      key_comment := LOOKUP_KEY (key_pressed, COMMENT);
      IF key_comment = 0
      THEN
          MESSAGE ("That key is not defined.");
      ELSE
          IF key_comment = ""
          THEN
              MESSAGE ("There is no comment for that key.");
          ELSE
              MESSAGE (key_comment);
          ENDIF;
      ENDIF;
   ENDPROCEDURE;
```

This procedure looks in the current key map list for the next key pressed. If the key is found, any comment associated with that key is put into the message buffer.

READ_LINE

Displays the text that you specify as a prompt for input and reads the information entered in response to the prompt. You can optionally specify the maximum number of characters to be read. READ_LINE returns a string that holds the data that is entered in response to the prompt.

FORMAT

string2 := READ_LINE *[(string1 [,integer])]*

PARAMETERS

string1

A quoted string, a variable name representing a string constant, or an expression that evaluates to a string, that is the text used as a prompt for input. This parameter is optional.

integer

The integer value that indicates how many characters to read from the input entered in response to the prompt. The maximum number is 132. This parameter is optional. If not present, control of execution passes from READ_LINE to VAXTPU's main loop when the user presses RETURN, CTRL/Z, or the one hundred thirty-second character.

return value

A string storing the user's response to a prompt.

DESCRIPTION

The terminators for READ_LINE are the standard VMS terminators such as CTRL/Z and RETURN. READ_LINE is not affected by VAXTPU key definitions; the built-in takes literally all keys except standard VMS terminators.

By default, the text you specify as a prompt is written in the prompt area on the screen. The prompt area is established with the built-in procedure SET (PROMPT_AREA). See SET (PROMPT_AREA) for more information. If no prompt area is defined, the text specified as a prompt is displayed at the current location on the device pointed to by SYS$OUTPUT (usually your terminal).

If READ_LINE terminates because it reaches the limit of characters specified as the second parameter, the last character read becomes the last key. Example 2 is a procedure that tests for the last key entered in a prompt string.

In DECwindows VAXTPU, READ_LINE maps the main widget if it is not already mapped.

When you invoke VAXTPU with the /NODISPLAY qualifier, terminal functions such as screen display and key definitions are not used. The built-in procedure READ_LINE calls the LIB$GET_INPUT routine to issue a prompt to SYS$INPUT and accept input from the user. A read done this way does not terminate when the number of keys you specified as the second parameter (*integer*) are entered. However, *string2* contains

the number of characters specified by the integer parameter and LAST_KEY contains the value of the key that corresponds to the integer specified as the last key to be read, except in the following cases. If the read is terminated by CTRL/Z, LAST_KEY has the value CTRL/Z. If the read is terminated by a carriage return before the specified integer limit is reached, LAST_KEY has the value of the RETURN key.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	READ_LINE must appear on the right-hand side of an assignment statement.
TPU$_TOOMANY	ERROR	READ_LINE accepts no more than two arguments.
TPU$_INVPARAM	ERROR	One of the arguments to READ_LINE has the wrong data type.

EXAMPLES

1
```
my_prompt := READ_LINE ("Enter key definition:", 1)
```

This assignment statement displays the text "Enter key definition:" in the prompt area, and stores the first character of the user's response in the variable *my_prompt*.

2
```
PROCEDURE user_test_lastkey

   LOCAL my_key,
         k;

   my_input := READ_LINE ("Enter 3 characters:", 3);

!  Press the keys "ABC"

   my_key := LAST_KEY;
   IF my_key = KEY_NAME ("C")
   THEN
       MESSAGE (" C key ");
   ELSE
       MESSAGE (" Error ");
   ENDIF;
ENDPROCEDURE;
```

This procedure prompts for three characters and stores them in the variable *my_input*. It then tests for the last key entered.

3
```
! Parameters:
!
!   old_number              Old integer value - input
!   new_number              New integer value - output
!   prompt_string           Text of prompt - input
!   no_value_message        Message printed if user hits RETURN to
!                           get out of the command - input

PROCEDURE user_prompt_number (old_number, new_number,
                              prompt_string, no_value_message)
! String read after prompt

   LOCAL read_line_string;
```

```
    new_number := old_number;
    IF old_number < 0
       THEN
          read_line_string := READ_LINE (prompt_string);
          EDIT (read_line_string, TRIM);
          IF read_line_string = ''
          THEN
              MESSAGE (no_value_message);
              new_number := 0;
              RETURN (0);
          ELSE
            !  Change lowercase l to #1
            TRANSLATE (read_line_string, "1", "l");
            new_number := INT (read_line_string);
               IF (new_number = 0) and (read_line_string <> "0")
               THEN
                   MESSAGE (FAO ("Don't understand !AS",
                            read_line_string));
                   RETURN (0);
                ELSE
                    RETURN (1);
                ENDIF;
          ENDIF;
       ELSE
          RETURN (1);
    ENDIF;
ENDPROCEDURE;
```

This procedure is used by commands that prompt for integers. The procedure returns true if prompting worked or was not needed; it returns false otherwise. The returned value is passed back as an output parameter.

REALIZE_WIDGET

Creates a window for the specified widget instance and maps the window to the display.

FORMAT

REALIZE_WIDGET *(widget)*

PARAMETER

widget
The widget instance you want VAXTPU to realize.

DESCRIPTION

Note that you can realize a widget only once during the widget's lifetime. For more information on realizing widgets, see the *VMS DECwindows Guide to Application Programming* and the *VMS DECwindows Toolkit Routines Reference Manual*.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	REALIZE_WIDGET must return a value.
TPU$_TOOMANY	ERROR	Too many arguments specified.
TPU$_TOOFEW	ERROR	Too few arguments specified.
TPU$_INVPARAM	ERROR	The argument to REALIZE_WIDGET has the wrong data type.
TPU$_REQUIRESDECW	ERROR	Requires the VAXTPU DECwindows screen updater.

EXAMPLE

```
REALIZE_WIDGET (example_widget);
```

This statement realizes the widget stored in *example_widget*.

RECOVER_BUFFER

Reconstructs the work done in the buffer whose name you specify. VAXTPU creates a new buffer using the specified buffer name and, using the information in the original buffer's journal file, recovers all the changes made to records in the original file. The resulting recovery is written to the newly created buffer.

FORMAT

buffer1 := RECOVER_BUFFER *(string1* { *[,string2]* / *[,buffer2]* } *)*

PARAMETERS

string1

The name of the buffer you are trying to recover.

string2

The name of the journal file you want VAXTPU to use to recover your buffer. If you did not set a journal file name using SET (JOURNALING), in most cases VAXTPU will have created the journal file using its default journal file naming algorithm. If the journal file was named by default, you need not specify a journal file name with RECOVER_BUFFER. If you specified a journal file name using SET (JOURNALING), use the same name with RECOVER_BUFFER.

Do not specify any directory name in this string. Specify only the buffer name and the extension, if any.

buffer2

The buffer whose attributes you want applied to the newly created buffer. For more information on using a buffer as a template, see the description of the CREATE_BUFFER built-in.

return value

The buffer containing the recovered text. If the recovery failed, an integer 0 is returned.

DESCRIPTION

Do not confuse the RECOVER_BUFFER built-in with the /RECOVER command qualifier in DCL. /RECOVER is used when invoking VAXTPU to recover a session or buffer. RECOVER_BUFFER, on the other hand, is used after VAXTPU has been invoked. It uses a buffer change journal file to recover the changes made to a specified buffer.

Note that RECOVER_BUFFER works on with buffer change journaling; you cannot recover a keystroke journal file with RECOVER_BUFFER.

Only the first parameter (the old buffer name) is required. If you want to specify the third parameter but not the second, you must use a comma as a placeholder, as follows:

```
RECOVER_BUFFER ("junk.txt", , template_buffer);
```

The third parameter is optional.

If VAXTPU returns a message that it cannot find the journal file and if the buffer you are trying to recover is small, the reason for the message might be that changes to the buffer were never written to the journal file because there were not enough changes to trigger the first write operation to the journal file. Similarly, if some text is missing after recovery, the reason might be that the last few changes did not trigger a write operation. For more information on how VAXTPU manages write operations to a journal file, see the description of the SET (JOURNALING) built-in.

Buffer change journaling does not journal changes in buffer attributes (such as modifiability of the buffer or visibility of tabs). Buffer change journaling only tracks changes to records in the buffer, such as addition, deletion, or modification of a record or changes in a record's attributes.

If you press CTRL/C during a recovery, VAXTPU terminates the recovery, closes the journal file, and deletes the newly created buffer.

If possible, after a successful recovery, VAXTPU continues journaling new changes into the journal file that was used during the recovery. However, it is likely that the journal file contains partial records at the end. In this case, VAXTPU cannot continue journaling to the same file. VAXTPU closes the journal file, marks the buffer unsafe for journaling, and signals an error.

If you have journal files created with the default naming algorithm as a result of editing multiple buffers with the same or similar names, RECOVER_BUFFER might not recover the buffer you intend. For more information on the default journal file naming algorithm, see Section 1.7.1. For example, suppose you were editing two buffers, one called TEST! and the other called TEST?. The default journal file naming algorithm creates for each buffer a journal file named TEST_.TPU$JOURNAL. The journal file for the buffer created first has the lower version number. If there were a system interruption while you were editing both buffers, and if the journal file for TEST! had the lower version number, then RECOVER_BUFFER would recover the journal file created for the buffer TEST?.

When you write the contents of a buffer to a file, VAXTPU erases the journal file. If you write the contents of the buffer to a file other than the default output file, the journal file contains a pointer to the file to which you last wrote the buffer. For example, if the buffer is called MAIN but you write the contents of the buffer to a file called OPUS.TXT, the journal file contains a pointer to the file OPUS.TXT. OPUS.TXT is known as the "source file" because, during a recovery VAXTPU uses OPUS.TXT as the source for the contents of the buffer as they were when the write operation was performed.

Caution: Since journal files often point to a source file that is not the same as the file originally read into the buffer, if you delete the source file, the buffer will be unrecoverable.

Similarly, if you have changed the name of the original file required for a recovery, the buffer will be unrecoverable. VAXTPU prompts for a new file name if it cannot find the original file. Be careful to specify the correct file name in this case.

SIGNALED ERRORS

TPU$_JRNLNOTSAFE	WARNING	The buffer is not safe for journaling.
TPU$_NOTJOURNAL	ERROR	The file specified is not a valid journal file.
TPU$_RECOVERABORT	WARNING	An inconsistency was found between the journal file and the currently executing procedure. Recovery is aborted and the journal file closed.
TPU$_RECOVERFAIL	ERROR	Recovery was terminated abnormally due to journal file inconsistency.
TPU$_RECOVERQUIT	WARNING	You did not specify a valid source file name.

EXAMPLES

1
```
RECOVER_BUFFER ("junk.txt");
```

This statement directs VAXTPU to find the buffer change journal file associated with the original buffer JUNK.TXT, to create a new buffer called JUNK.TXT, and, using the information from the journal file, to recover the changes made in the original JUNK.TXT buffer. The results of the recovery are placed in the new JUNK.TXT buffer.

2
```
defaults_buffer := CREATE_BUFFER ("Defaults");
SET (EOB_TEST, defaults_buffer, "[That's all, folks!]");
user_buffer := CREATE_BUFFER ("User1.txt", "", defaults_buffer);
SET (JOURNALING, user_buffer, ON, "user1_journal.tpu$journal");
    .
    .
    .
RECOVER_BUFFER ("User1.txt", "user1_journal.tpu$journal",
                defaults_buffer);
```

This code fragment creates a defaults buffer, changes an attribute of the defaults buffer, and creates a user buffer. The fourth statement turns on buffer change journaling and designates the file named USER1_JOURNAL.TPU$JOURNAL as the journaling file. At some later point in the session (represented by the ellipses) the RECOVER_BUFFER statement is used to recover the contents of the old USER1.TXT by using the journal file USER1_JOURNAL.TPU$JOURNAL. The attributes of the defaults buffer are applied to the newly created buffer USER1.TXT. In this case, the new buffer has the end-of-buffer text "[That's all, folks!]".

REFRESH

Repaints the whole screen. REFRESH erases any extraneous characters, such as those caused by noise on a communication line, and repositions the text so that the screen represents the last known state of the editing context.

FORMAT

REFRESH

PARAMETERS

None.

DESCRIPTION

REFRESH causes a redrawing of every line of every window that is mapped to the screen. The prompt area is erased. This built-in procedure causes the screen to change immediately. Even if REFRESH is issued from within a procedure, the action takes place immediately; VAXTPU does not wait until the entire procedure is completed to execute REFRESH.

If screen updating is disabled when VAXTPU executes the REFRESH command, VAXTPU performs the refresh operation when updating is enabled again.

VAXTPU reissues escape sequences as appropriate to do any of the following:

- To set the width of the terminal
- To set the scrolling region
- To set the keypad to applications mode
- To set the video attributes to a known state
- To clear the screen of a DIGITAL-supported terminal
- To reset the nonalphanumeric character sets

REFRESH repaints the whole screen. See UPDATE for a description of how to update a single window to make it reflect the current state of its associated buffer. If you want to update every visible window without erasing the screen, use the UPDATE (ALL) built-in.

See Chapter 6 for an explanation of how the screen is updated under various circumstances.

SIGNALED ERROR

TPU$_TOOMANY	ERROR	REFRESH takes no parameters.

EXAMPLES

1
```
REFRESH
```

This statement causes the screen manager to repaint the whole screen so that it reflects the current internal state of the editor.

2
```
PROCEDURE user_repaint
   ERASE (message_buffer);
   REFRESH;
ENDPROCEDURE;
```

This procedure removes the contents of the message buffer and then repaints the whole screen.

REMAIN

Specifies that all characters from the current position to the end of the line should be included in a pattern.

FORMAT

REMAIN

PARAMETERS

None.

DESCRIPTION

When used as part of a complex pattern or as an argument to SEARCH, REMAIN matches the rest of the characters on a line. REMAIN matches successfully even if there are no more characters on the line.

Although REMAIN behaves much like a built-in, it is actually a keyword.

SIGNALED ERROR

REMAIN is a keyword and has no completion codes.

EXAMPLES

1
```
pat1 := LINE_BEGIN + "!" + REMAIN
```

This assignment statement stores in the variable *pat1* a pattern that matches all lines that have an exclamation point at the beginning of the line.

2
```
PROCEDURE remove_comments

   LOCAL pat1,
         here,
         comment_range;

   here := MARK (NONE);    ! Remember our location
   pat1 := "!" + REMAIN;

   POSITION (BEGINNING_OF (CURRENT_BUFFER));
   LOOP
      comment_range := SEARCH_QUIETLY (pat1, FORWARD);
      EXITIF comment_range = 0;

      ERASE (comment_range);
      POSITION (comment_range);
   ENDLOOP;

   POSITION (here);
ENDPROCEDURE;
```

This procedure removes all comments from the current buffer. It does not correctly handle quoted strings containing exclamation points.

REMOVE_KEY_MAP

Removes key maps from key map lists.

FORMAT

REMOVE_KEY_MAP *(string1, string2 [, ALL])*

PARAMETERS

string1

A quoted string, or a variable name representing a string constant, that specifies the name of the key map list containing the key map to be removed.

string2

A quoted string, or a variable name representing a string constant, that specifies the name of the key map to be removed from the key map list.

ALL

This keyword is an optional argument. It specifies that all the key maps with the name specified by *string2* in the key map list are to be removed.

DESCRIPTION

This built-in procedure removes one or more key maps from a key map list. If the optional keyword ALL is specified, all of the key maps with the specified name in the key map list are removed from the list. Otherwise, only the first entry with the specified name is removed.

SIGNALED ERRORS

TPU$_NOKEYMAP	WARNING	You specified an argument that is not a defined key map.
TPU$_NOKEYMAPLIST	WARNING	You specified an argument that is not a defined key map list.
TPU$_KEYMAPNOTFND	WARNING	The key map you specified is not found.
TPU$_EMPTYKMLIST	WARNING	The key map list you specified contains no key maps.
TPU$_TOOFEW	ERROR	Too few arguments passed to the REMOVE_KEY_MAP built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the REMOVE_KEY_MAP built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the REMOVE_KEY_MAP built-in.

TPU$_UNKKEYWORD	ERROR	An unknown keyword has been used as an argument. Only the keyword ALL is allowed.
TPU$_BADKEY	ERROR	An unknown keyword has been used as an argument. Only the keyword ALL is allowed.

EXAMPLE

```
user$keymap_1 := CREATE_KEY_MAP ("keymap_1");
user$keymap_2 := CREATE_KEY_MAP ("keymap_2");
user$keymap_list := CREATE_KEY_MAP_LIST ("keymap_list", user$keymap_1,
                                         user$keymap_2);
ADD_KEY_MAP (user$keymap_list, "last", user$keymap_1);
     .
     .
     .
SHOW (KEY_MAP_LISTS);
     .
     .
     .
REMOVE_KEY_MAP (user$keymap_list, user$keymap_1, ALL);
     .
     .
     .
SHOW (KEY_MAP_LISTS);
```

In this example, a key map list named KEYMAP_LIST is created. The call to SHOW (KEY_MAP_LISTS) shows that the key map list contains three key maps: KEYMAP_1, KEYMAP_2, and KEYMAP_1 again. After the call to REMOVE_KEY_MAP, the call to SHOW (KEY_MAP_LISTS) shows that the key map list contains only KEYMAP_2.

RETURN

A VAXTPU language element. It returns control from the current procedure to its caller, optionally specifying the value the current procedure returns to the caller.

FORMAT

RETURN *[[expression]]*

RETURN is a VAXTPU language element. It does not take parameters. However, it is optionally followed by a VAXTPU expression.

PARAMETERS

expression

This expression may be any VAXTPU expression, variable, or built-in. It specifies what the current procedure should return to its caller.

DESCRIPTION

The RETURN statement returns control from the current procedure to its caller. It also provides a value for the current routine.

RETURN is evaluated for correct syntax at compile time. In contrast, VAXTPU procedures are usually evaluated for a correct parameter count and parameter types at execution time.

SIGNALED ERROR

RETURN is a language element and signals no errors or warnings.

EXAMPLES

1
```
PROCEDURE user_erase_message_buffer
   IF CURRENT_BUFFER = message_buffer
   THEN
       RETURN;
   ENDIF;

   ERASE (message_buffer);
ENDPROCEDURE;
```

This procedure erases the message buffer. If the current buffer is the message buffer, it returns without erasing it.

2
```
PROCEDURE user_find_string (look_for)
   ON_ERROR
       RETURN "String not found";
   ENDON_ERROR;

   RETURN SEARCH (look_for, FORWARD);
ENDPROCEDURE;
```

This procedure searches for a string. If it does not find the string, it returns the string *String not found*. Otherwise, it returns the range containing the found string.

SAVE

Writes the binary forms of all currently defined procedures, variables, key definitions, key maps, and key map lists to the section file you specify.

FORMAT

SAVE *(string1 [,"NO_DEBUG_NAMES"]*
[,"NO_PROCEDURE_NAMES"]
[,"IDENT", string2])

PARAMETERS

string1

A string that is a valid VMS file specification. If you supply only a file name, VAXTPU uses the *current* device and directory, not necessarily the SYS$LOGIN device and directory, in the file specification.

"NO_DEBUG_NAMES"

A string that prevents VAXTPU from writing debugging information to the section file. When you use "NO_DEBUG_NAMES", VAXTPU does not write procedure parameter names or local variable names. You can reduce the size of the section file by specifying this string. Do not specify this string if you intend to use the VAXTPU debugger on the section file.

"NO_PROCEDURE_NAMES"

A string, or a variable or constant name representing this string, that prevents VAXTPU from writing procedure names to the section file. You can reduce the size of the section file by specifying this string. However, the procedure names are required to display a meaningful traceback when an error occurs. Therefore, do not specify this string if you want to use the application created by the section file with the TRACEBACK or LINE_NUMBER function set to ON.

"IDENT"

A string specifying that you want to assign an identifying string, such as a version number, to the section file.

string2

The string (usually a version number) that you want to assign to the section file.

DESCRIPTION

SAVE is used to create VAXTPU section files. If you are adding to an existing section file, the new section file contains all of the items from the original section file and the new items from the current editing session. Section files enable VAXTPU interfaces to start up quickly because they contain the following items in binary form:

- All compiled PROCEDURE . . . ENDPROCEDURE statements
- Every variable created (only the variable's name is saved, not its contents)

- Every key definition that binds a statement, procedure, program, or learn sequence to a key, including the comments that you add to key definitions
- Every key map and key map list created
- All defined constants

When you use the built-in procedure SAVE during an editing session to add items to an existing section file, SAVE does not keep items that were established interactively with the built-in procedure SET (for example, margin settings for buffers, or setting the editor's shift key to something other than the PF1 key).

If you do not specify a device and directory in the string parameter, VAXTPU uses your current device and directory.

The default file type is TPU$SECTION.

When you use the built-in procedure SAVE, informational messages are generated for any undefined procedures or ambiguous symbols as they are written to the section file. If the display of informational messages has been disabled, these messages are not displayed.

SIGNALED ERRORS

TPU$_SAVEERROR	ERROR	The section cannot be created because of errors in the context being saved.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SAVE built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SAVE built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the SAVE built-in.
TPU$_SECTUNDEFPROC	WARNING	Undefined procedures or ambiguous symbols were found while the section file was being written.
TPU$_BADSYMTAB	ERROR	VAXTPU's symbol tables are inconsistent.
TPU$_SAVEUNDEFPROC	INFORMATIONAL	An undefined procedure is being written to the section file.
TPU$_SAVEAMBIGSYM	INFORMATIONAL	An ambiguous symbol is being written to the section file.

EXAMPLES

1

```
SAVE ("SYS$LOGIN:mysection.TPU$SECTION")
```

This statement, issued just before exiting from the editor, adds all of the procedure definitions, key definitions, and variables from your current editing session to the section file with which you invoked VAXTPU. The new file that you specify, SYS$LOGIN:mysection.TPU$SECTION, contains initialization items from the original section file and from your editing session.

To invoke VAXTPU with the new section file, enter the following command at the DCL level:

```
$ EDIT/TPU/SECTION=sys$login:mysection
```

2

```
PROCEDURE eve_next_paragraph
   LOCAL pat1,
         the_range;
   pat1 := LINE_BEGIN + LINE_BEGIN + ARB (1);
   the_range := SEARCH_QUIETLY (pat1, FORWARD, EXACT);
   IF the_range <> 0
   THEN
       POSITION (END_OF (the_range));
   ENDIF;
ENDPROCEDURE;
```

3

```
PROCEDURE tpu$local_init
   SET (SHIFT_KEY, KP0);
   DEFINE_KEY ("eve_next_paragraph", PERIOD, "Next Para");
ENDPROCEDURE;

SAVE ("my_section", "ident", "V1.5");
QUIT;
```

These procedures and statements show how SAVE can be used in a command file to extend an application. The first procedure moves the cursor to the beginning of the next paragraph. The second procedure defines a shift key and binds the procedure *eve_next_paragraph* to the period key on the keypad. The SAVE statement directs VAXTPU to write the binary form of *eve_next_paragraph* and the key definition to a section file called MY_SECTION.TPU$SECTION. The second and third parameters to the SAVE statement direct VAXTPU to assign the string "V1.5" to the section file. The QUIT statement terminates the VAXTPU session.

SCAN

Returns a pattern that matches only characters that do not appear in the string, buffer, or range used as its parameter. SCAN matches as many characters as possible.

FORMAT

pattern := SCAN ({ *buffer* / *range* / *string* } [, { *FORWARD* / *REVERSE* }])

PARAMETERS

buffer

An expression that evaluates to a buffer. SCAN does not match any of the characters that appear in the buffer.

range

An expression that evaluates to a range. SCAN does not match any of the characters that appear in the range.

string

An expression that evaluates to a string. SCAN does not match any of the characters that appear in the string.

FORWARD

A keyword directing VAXTPU to match characters in the forward direction. This is the default.

REVERSE

A keyword directing VAXTPU to match characters as follows: first, match characters in the forward direction until VAXTPU finds a character that is a member of the set of characters. Next, return to the first character matched and start matching characters in the reverse direction until VAXTPU finds a character that is in the set.

You can specify REVERSE only if you are using SCAN in the first element of a pattern being used in a reverse search. In all other contexts, specifying REVERSE has no effect.

The behavior enabled by REVERSE allows an alternate form of reverse search. By default, a reverse search stops as soon as a successful match occurs, even if there might have been a longer successful match in the reverse direction. By specifying REVERSE, you direct VAXTPU not to stop matching in either direction until it has matched as many characters as possible.

return value

A pattern matching only characters that do not appear in the buffer, range, or string used as the parameter.

DESCRIPTION

SCAN matches one or more characters, none of which appear in the string, buffer, or range passed as its parameter. SCAN matches as many characters as possible, stopping only if it finds a character that is present in its parameter or if it reaches the end of a line. If SCAN is part of a larger pattern, SCAN does not match a character if doing so prevents the rest of the pattern from matching.

SCAN does not cross line boundaries. To match a string of characters that may cross one or more line boundaries, use SCANL.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	SCAN must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	SCAN requires at least one argument.
TPU$_TOOMANY	ERROR	SCAN accepts no more than one argument.
TPU$_ARGMISMATCH	ERROR	SCAN was given an argument of the wrong type.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of SCAN.

EXAMPLES

1
```
pat1 := SCAN ("abc")
```

This assignment statement stores a pattern that matches the longest string of characters that does not contain a, b, or c in *pat1*.

2
```
PROCEDURE user_find_parens

   paren_text    := ANY("(') + SCAN (')");
   found_range   := SEARCH (paren_text, FORWARD, NO_EXACT);

   IF found_range = 0   ! No parentheses.
   THEN
       MESSAGE  ("No parentheses found.");
   ELSE
       POSITION (found_range);
   ENDIF;
ENDPROCEDURE;
```

This procedure identifies parenthesized text within a single line. It moves the editing point to the beginning of the parenthesized text, if it is found.

```
3  PROCEDURE user_remove_odd_characters
      LOCAL pat1,
            odd_text;

      pat1 := SCAN ("abcdefghijklmnopqrstuvwxyz 0123456789");
      POSITION (BEGINNING_OF (CURRENT_BUFFER));
      LOOP
         odd_text := SEARCH_QUIETLY (pat1, FORWARD);
         EXITIF odd_text = 0;
         ERASE (odd_text);
         POSITION (odd_text);
      ENDLOOP;
      POSITION (END_OF (CURRENT_BUFFER));
   ENDPROCEDURE;
```

This procedure goes through the current file, deleting all characters that are not numbers, letters, or spaces.

```
4  word := SCAN (' ', REVERSE);
```

This statement defines the variable *word* to mean the longest consecutive string of characters that does not include a space character. Suppose you are searching the text Xanadu, the cursor is on the n, and you use the following statement:

```
the_range := SEARCH (word, REVERSE);
```

The variable *the_range* contains the word Xanadu. The reason for this is when you use SCAN with REVERSE as the first element of a pattern, and then use that pattern in a reverse search, SCAN matches as many characters as possible in both the forward and reverse directions.

Suppose that the cursor is on the n of Xanadu, as before, but you define the variable *word* without the REVERSE keyword, as follows:

```
word := SCAN (' ');
```

If you do a reverse search, *the_range* contains the characters nadu.

SCANL

Returns a pattern matching a string of characters, including line breaks, none of which appear in the buffer, range, or string used as its parameter. The returned pattern contains as many characters and line breaks as possible.

FORMAT

pattern := SCANL ({ *buffer* / *range* / *string* } [, { *FORWARD* / *REVERSE* }])

PARAMETERS

buffer

An expression that evaluates to a buffer. SCANL does not match any of the characters that appear in the buffer.

range

An expression that evaluates to a range. SCANL does not match any of the characters that appear in the range.

string

An expression that evaluates to a string. SCANL does not match any of the characters that appear in the string.

FORWARD

A keyword directing VAXTPU to match characters in the forward direction. This is the default.

REVERSE

A keyword directing VAXTPU to match characters as follows: first, match characters in the forward direction until VAXTPU finds a character that is a member of the set of characters. Next, return to the first character matched and start matching characters in the reverse direction until VAXTPU finds a character that is in the set.

You can specify REVERSE only if you are using SCANL in the first element of a pattern being used in a reverse search. In all other contexts, specifying REVERSE has no effect.

The behavior enabled by REVERSE allows an alternate form of reverse search. By default, a reverse search stops as soon as a successful match occurs, even if there might have been a longer successful match in the reverse direction. By specifying REVERSE, you direct VAXTPU not to stop matching in either direction until it has matched as many characters as possible.

return value

A pattern that may contain line breaks and that matches only characters that do not appear in the buffer, range, or string used as the parameter.

DESCRIPTION

SCANL is similar to SCAN in that it matches one or more characters that do not appear in the string, buffer, or range used as its parameter. Unlike SCAN, however, SCANL does not stop matching when it reaches the end of a line. Rather, it successfully matches the line end and continues trying to match characters on the next line. If SCANL is part of a larger pattern, it does not match a character or line boundary if doing so prevents the rest of the pattern from matching.

SCANL must match at least one character.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	SCANL must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	SCANL requires at least one argument.
TPU$_TOOMANY	ERROR	SCANL requires no more than one argument.
TPU$_ARGMISMATCH	ERROR	Argument to SCANL has the wrong type.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of SCANL.

EXAMPLES

1
```
sentence_pattern := any ("ABCDEFGHIJKLMNOPQRSTUVWXYZ") + scanl (".!?);
```

This assignment statement creates a pattern that matches a sentence. It assumes that a sentence ends in one of the following characters: a period (.), an exclamation point (!), or a question mark (?). The matched text does not include the punctuation mark ending the sentence.

2
```
PROCEDURE user_remove_non_numbers
   LOCAL pat1,
         non_number_region;

   pat1 := SCANL ("0123456789");

   POSITION (BEGINNING_OF (CURRENT_BUFFER));

   LOOP
      non_number_region := SEARCH_QUIETLY (pat1, FORWARD);
      EXITIF non_number_region = 0;

      ERASE (non_number_region);
      POSITION (non_number_region);
   ENDLOOP;

   POSITION (BEGINNING_OF (CURRENT_BUFFER));
ENDPROCEDURE;
```

This procedure goes through the current buffer erasing anything that is not a number. The only line breaks it leaves in the file are those between a line ending with a number and one beginning with a number.

SCROLL

Moves the lines of text in the buffer up or down on the screen by the number of lines you specify.

FORMAT

[integer2 :=]SCROLL *(window [,integer1])*

PARAMETERS

window

The window associated with the buffer whose text you want to scroll.

integer1

The signed integer value that indicates how many lines you want the text to scroll. If you supply a negative value for the second parameter, the lines of text scroll off the top of the screen, leaving the cursor closer to the beginning of the buffer. If you supply a positive value for the second parameter, the lines of text scroll off the bottom of the screen, leaving the cursor closer to the end of the buffer. If you specify 0 as the integer value, no scrolling occurs.

This parameter is optional. If you omit the second parameter, the text scrolls continuously until it reaches a buffer boundary or until you press a key. If the current direction of the buffer is forward, the text scrolls to the end of the buffer. If the current direction of the buffer is reverse, the text scrolls to the beginning of the buffer. If you press a key that has commands bound to it, the scrolling stops and VAXTPU executes the commands bound to the key.

return value

An integer indicating the number and direction of lines actually scrolled as a result of using SCROLL.

DESCRIPTION

You can scroll text only in a visible window. If the window is not currently visible on the screen, VAXTPU issues an error message.

During scrolling, the cursor does not move but stays positioned at the same relative screen location. The current editing point is different from the editing point that was current before you issued the SCROLL built-in.

SCROLL optionally returns an integer that indicates the number and direction of lines actually scrolled. If you supply a negative value for the second parameter, the lines of text scroll off the bottom of the screen, leaving the cursor closer to the beginning of the buffer. If you supply a positive value for the second parameter, the lines of text scroll off the top of the screen, leaving the cursor closer to the end of the buffer. The value of *integer2* may differ from what was specified in *integer1*.

Note that SCROLL causes the screen to scroll immediately. It does not wait to take effect for the completion of a procedure.

If the buffer has been modified or the window display has altered since the last update, the window is updated before the scrolling operation begins.

SCROLL does not work in the following cases:

- If you have turned off the screen update flag with SET (SCREEN_UPDATE, OFF)
- If you used the /NODISPLAY qualifier when invoking VAXTPU on an unsupported device
- If the window that you specify is not visible on the screen

When the scrolling is complete, the editing point (record and offset) is set to match the cursor position (screen line and column position).

After the scrolling stops, the cursor may be located to the right of the last character in the new current record, to the left of the left margin, or in the middle of a tab. In this instance, any VAXTPU built-in procedure that requires a record offset (for example, CURRENT_OFFSET, MOVE_HORIZONTAL, MOVE_VERTICAL, MARK, and so on) causes the record to be blank-padded to the cursor location.

If the screen you are using does not have hardware scrolling regions, the window being scrolled is repainted for each scroll that would have occurred. For instance, the statement SCROLL (my_window,3) repaints the window three times.

If you use SCROLL while positioned after the end of the buffer, SCROLL completes successfully and returns 0 as the amount scrolled.

SIGNALED ERRORS

TPU$_CONTROLC	ERROR	You pressed CTRL/C to stop scrolling.
TPU$_WINDNOTMAPPED	WARNING	You are trying to scroll an unmapped window.
TPU$_TOOFEW	ERROR	SCROLL requires at least one parameter.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLES

1
```
SCROLL (main_window,+10)
```

This statement causes the text of the buffer that is mapped to the main window to scroll forward 10 lines.

2
```
SCROLL (my_window)
```

This statement causes the text in the buffer that is mapped to *my_window* to scroll in the direction that the buffer is set to until it reaches a buffer boundary or the user presses any key.

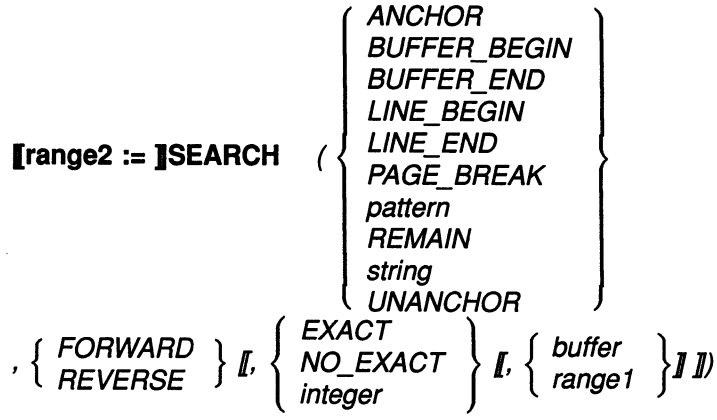
```
3   PROCEDURE user_scroll_buffer
       LOCAL scrolled_lines;
       MESSAGE ("Press any key to stop scrolling...");
       scrolled_lines := SCROLL (main_window);
       dummy_key := READ_KEY;
       RETURN scrolled_lines;
    ENDPROCEDURE;
```

This procedure scrolls the main buffer until the user presses a key. The procedure returns the number of lines scrolled.

SEARCH

Looks for a particular arrangement of characters in a buffer or range and returns a range that contains those characters.

FORMAT



PARAMETERS

ANCHOR
A keyword directing SEARCH to start a search at the current character position. Use this keyword as part of a complex pattern.

BUFFER_BEGIN
A keyword used to match the beginning of a buffer.

BUFFER_END
A keyword used to match the end of a buffer.

LINE_BEGIN
A keyword used to match the beginning of a line.

LINE_END
A keyword used to match the end of a line.

PAGE_BREAK
A keyword used to match a form-feed character.

pattern
The pattern that you want to match.

REMAIN
A keyword specifying a match starting at the current character and continuing to the end of the current line.

string
The string that you want to match.

UNANCHOR
A keyword specifying that the next pattern element can match anywhere after the previous pattern element. Use this keyword as part of a complex pattern.

For more information on these keywords, refer to the individual descriptions of them in this chapter.

FORWARD
Indicates a search in the forward direction.

REVERSE
Indicates a search in the reverse direction.

EXACT
Indicates that the characters SEARCH is trying to match must be the same case and have the same diacritical markings as those in the string or pattern used as the first parameter to SEARCH.

NO_EXACT
Indicates that the characters SEARCH is trying to match need not be the same case nor have the same diacritical markings as those in the string or pattern used as the first parameter to SEARCH. NO_EXACT is the default value for the optional third parameter.

integer
Specifies how SEARCH should handle case and diacritical information if you want to match one attribute and ignore the other. DIGITAL recommends that you use the defined constants available for specifying this integer. The defined constants are as follows:

- TPU$K_SEARCH_CASE — Equivalent to the integer 1. This specifies that the search should match the case of the first parameter but be insensitive to the diacritical markings of the first parameter.
- TPU$K_SEARCH_DIACRITICAL — Equivalent to the integer 2. This specifies that the search should match the diacritical markings of the first parameter but be insensitive to the case of the first parameter.

buffer
The buffer in which to search. SEARCH starts at the beginning of the buffer when doing a forward search and at the end of the buffer when doing a reverse search.

range1
The range in which to search. SEARCH starts at the beginning of the range when doing a forward search and at the end of the range when doing a reverse search.

To search a range for all occurrences of a pattern, you must define the range dynamically after each successful match. Otherwise, SEARCH positions to the beginning of the range and finds the same occurrence over and over. See the example section for a procedure that searches for all occurrences of a pattern in a range.

return value

The range containing characters that match the pattern or string specified as a parameter.

DESCRIPTION

SEARCH looks for text that matches the string, pattern, or keyword specified as its first parameter. If it finds such text, it creates a range containing this text and returns it. If SEARCH does not find a match, SEARCH returns 0 and signals the error TPU$_STRNOTFOUND. To perform a search that does not signal an error when there is no match, use the SEARCH_QUIETLY built-in.

The starting position for the search depends on the optional fourth parameter and the search direction. If you do not specify the fourth parameter, the search starts at the editing point.

If you specify a range for the fourth parameter, the search starts at the beginning of the range for a forward search, or the end of the range for a reverse search. When searching a range, SEARCH matches only text inside the range. It does not look at text outside the range.

If you specify a buffer for the fourth parameter, the search starts at the beginning of the buffer for a forward search, or the end of the buffer for a reverse search.

To determine whether the searched text contains a match, SEARCH examines the character at the starting position and attempts to match the character against the pattern, text, or keyword specified. By default, the search is unanchored. This allows SEARCH to move one character in the direction of the search if the character at the start position does not match. SEARCH continues in this manner until it finds a match or reaches the bounds of the buffer or range.

To prevent SEARCH from moving the starting position in the direction of the search, use the ANCHOR keyword when you define the pattern to be matched.

SEARCH does not change the current buffer or the editing point in that buffer.

For more information about searching, see Chapter 2.

SIGNALED ERRORS

TPU$_STRNOTFOUND	WARNING	Search for a string or pattern was unsuccessful.
TPU$_TOOFEW	ERROR	SEARCH requires at least two arguments.
TPU$_TOOMANY	ERROR	SEARCH accepts no more than four arguments.
TPU$_ARGMISMATCH	ERROR	One of the parameters to SEARCH is of the wrong type.
TPU$_INVPARAM	ERROR	One of the parameters to SEARCH is of the wrong type.

TPU$_BADKEY	WARNING	You specified an incorrect keyword to SEARCH.
TPU$_MINVALUE	WARNING	The integer parameter to SEARCH must be greater than or equal to –1.
TPU$_MAXVALUE	WARNING	The integer parameter to SEARCH must be less than or equal to 3.
TPU$_NOCURRENTBUF	ERROR	If you do not specify a buffer or range to search, you must position to a buffer before searching.
TPU$_CONTROLC	ERROR	You pressed CTRL/C while SEARCH was executing.
TPU$_ILLPATAS	ERROR	The pattern to SEARCH contained a partial pattern assignment to a variable not defined in the current context.

EXAMPLES

1
```
user_range := SEARCH ("Reflections of MONET", FORWARD, NO_EXACT)
```

If you search a buffer in which the string "Reflections of Monet" appears, this assignment statement stores the characters "Reflections of Monet" in the range *user_range*. The search finds a successful match even though the characters in the word "Monet" do not match in case, because you specified NO_EXACT.

2
```
PROCEDURE user_find_chap
   LOCAL chap,
         found_range;
   ON_ERROR
       IF ERROR = TPU$_STRNOTFOUND
       THEN
           MESSAGE ("CHAPTER not found.");
       ELSE
           MESSAGE (MESSAGE_TEXT (ERROR));
       ENDIF;
   ENDON_ERROR;

   chap := LINE_BEGIN + "CHAPTER";
   found_range := SEARCH (chap, FORWARD, NO_EXACT);

   IF found_range <> 0    ! No match found.
   THEN
       POSITION (found_range);
   ENDIF;
ENDPROCEDURE;
```

This procedure searches for the word "CHAPTER" appearing at the beginning of a line. If SEARCH finds the word, the built-in positions to the beginning of the string. If SEARCH does not find the word, the built-in writes an appropriate message in the message buffer.

```
3  PROCEDURE user_search_range
      LOCAL found_count;
      ON_ERROR
          [TPU$_STRNOTFOUND, TPU$_CONTROLC]:
          MESSAGE ( FAO ("Found !SL occurrences.", found_count));
          RETURN;
         [OTHERWISE]:ABORT;
      ENDON_ERROR;
      found_count := 0;
      the_pattern := "blue skies";
      the_range := CREATE_RANGE (BEGINNING_OF (CURRENT_BUFFER),
                                 END_OF (CURRENT_BUFFER),
                                 NONE);
      found_range := CREATE_RANGE (BEGINNING_OF (CURRENT_BUFFER),
                                   BEGINNING_OF (CURRENT_BUFFER),
                                   NONE);
       LOOP
          the_range := CREATE_RANGE (END_OF (found_range),
                                     END_OF (the_range), NONE);
          found_range := SEARCH (the_pattern, FORWARD, NO_EXACT,
                                 the_range);
          found_count := found_count + 1;
       ENDLOOP;
   ENDPROCEDURE;
```

This procedure searches the range *the_range* for all occurrences of the pattern "blue skies". If SEARCH finds the pattern, the procedure redefines *the_range* to begin after the end of the pattern just found. If the procedure did not redefine the range, SEARCH would keep finding the first occurrence over and over. The procedure reports the number of occurrences of the pattern.

SEARCH_QUIETLY

Looks for a particular arrangement of characters in a buffer or range and returns a range that contains those characters. Unlike the SEARCH built-in, SEARCH_QUIETLY does not signal TPU$_STRNOTFOUND when it fails to find a string.

FORMAT

```
[ range2 := ] SEARCH_QUIETLY ( { ANCHOR
                               | BUFFER_BEGIN
                               | BUFFER_END
                               | LINE_BEGIN
                               | LINE_END
                               | PAGE_BREAK
                               | pattern
                               | REMAIN
                               | string
                               | UNANCHOR } )
, { FORWARD } [, { EXACT    } [, { buffer } ] ])
  { REVERSE }    { NO_EXACT }    { range1 }
                 { integer  }
```

PARAMETERS

ANCHOR
A keyword directing SEARCH_QUIETLY to start a search at the current character position.

BUFFER_BEGIN
A keyword used to match the beginning of a buffer.

BUFFER_END
A keyword used to match the end of a buffer.

LINE_BEGIN
A keyword used to match the beginning of a line.

LINE_END
A keyword used to match the end of a line.

PAGE_BREAK
A keyword used to match a form-feed character.

pattern
The pattern that you want to match.

REMAIN
A keyword specifying a match starting at the current character and continuing to the end of the current line.

string
The string that you want to match.

UNANCHOR
A keyword specifying that the next pattern element can match anywhere after the previous pattern element. Use this keyword as part of a complex pattern.

For more information on these keywords, refer to the individual descriptions of them in this chapter.

FORWARD
Indicates a search in the forward direction.

REVERSE
Indicates a search in the reverse direction.

EXACT
Indicates that the characters SEARCH_QUIETLY is trying to match must be the same case and have the same diacritical markings as those in the string or pattern used as the first parameter to SEARCH_QUIETLY.

NO_EXACT
Indicates that the characters SEARCH_QUIETLY is trying to match need not be the same case nor have the same diacritical markings as those in the string or pattern used as the first parameter to SEARCH_QUIETLY. NO_EXACT is the default value for the optional third parameter.

integer
Specifies how SEARCH_QUIETLY should handle case and diacritical information if you want to match one attribute and ignore the other. Digital recommends that you use the defined constants available for specifying this integer. The defined constants are as follows:

- TPU$K_SEARCH_CASE — Equivalent to the integer 1. This specifies that the search should match the case of the first parameter but be insensitive to the diacritical markings of the first parameter.
- TPU$K_SEARCH_DIACRITICAL — Equivalent to the integer 2. This specifies that the search should match the diacritical markings of the first parameter but be insensitive to the case of the first parameter.

buffer
The buffer in which to search. SEARCH_QUIETLY starts at the beginning of the buffer when doing a forward search and at the end of the buffer when doing a reverse search.

range1
The range in which to search. SEARCH_QUIETLY starts at the beginning of the range when doing a forward search and at the end of the range when doing a reverse search.

To search a range for all occurrences of a pattern, you must define the range dynamically after each successful match. Otherwise, SEARCH_QUIETLY positions to the beginning of the range and finds the same occurrence over and over. See the example section for a procedure that searches for all occurrences of a pattern in a range.

return value

The range containing characters that match the pattern or string specified as a parameter.

DESCRIPTION

SEARCH_QUIETLY looks for text that matches the string, pattern, or keyword specified as its first parameter. If it finds such text, it creates a range containing this text and returns it. If SEARCH_QUIETLY does not find a match, the built-in returns 0.

The starting position for the search depends on the optional fourth parameter and the search direction. If you do not specify the fourth parameter, the search starts at the editing point.

If you specify a range for the fourth parameter, the search starts at the beginning of the range for a forward search, or the end of the range for a reverse search. When searching a range, SEARCH_QUIETLY matches only text inside the range. It does not look at text outside the range.

If you specify a buffer for the fourth parameter, the search starts at the beginning of the buffer for a forward search, or the end of the buffer for a reverse search.

To determine whether the searched text contains a match, SEARCH_QUIETLY examines the character at the starting position and attempts to match the character against the pattern, text, or keyword specified. By default, the search is unanchored. This allows SEARCH_QUIETLY to move one character in the direction of the search if the character at the start position does not match. SEARCH_QUIETLY continues in this manner until it finds a match or reaches the bounds of the buffer or range.

To prevent SEARCH_QUIETLY from moving the starting position in the direction of the search, use the ANCHOR keyword when you define the pattern to be matched.

SEARCH_QUIETLY does not change the current buffer or the editing point in that buffer.

For more information about searching, see Chapter 2.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SEARCH_QUIETLY requires at least two arguments.
TPU$_TOOMANY	ERROR	SEARCH_QUIETLY accepts no more than four arguments.
TPU$_ARGMISMATCH	ERROR	One of the parameters to SEARCH_QUIETLY is of the wrong type.
TPU$_INVPARAM	ERROR	One of the parameters to SEARCH_QUIETLY is of the wrong type.
TPU$_BADKEY	WARNING	You specified an incorrect keyword to SEARCH_QUIETLY.

TPU$_MINVALUE	WARNING	The integer parameter to SEARCH_QUIETLY must be greater than or equal to –1.
TPU$_MAXVALUE	WARNING	The integer parameter to SEARCH_QUIETLY must be less than or equal to 3.
TPU$_NOCURRENTBUF	ERROR	If you do not specify a buffer or range to search, you must position to a buffer before searching.
TPU$_CONTROLC	ERROR	You pressed CTRL/C while SEARCH_QUIETLY was executing.
TPU$_ILLPATAS	ERROR	The pattern to SEARCH_QUIETLY contained a partial pattern assignment to a variable not defined in the current context.

EXAMPLES

1
```
user_range := SEARCH_QUIETLY ("Reflections of MONET", FORWARD, NO_EXACT)
```

If you are searching a buffer in which the string "Reflections of Monet" appears, this assignment statement stores the characters "Reflections of Monet" in the range *user_range*. The search finds a successful match even though the characters in the word "Monet" do not match in case, because you specified NO_EXACT.

If the string "Reflections of Monet" does not appear in the buffer, SEARCH_QUIETLY assigns the value 0 to the variable *user_range*. It does not signal the TPU$_STRNOTFOUND error.

2
```
PROCEDURE user_find_chap

   LOCAL chap,
         found_range;

   chap := LINE_BEGIN + "CHAPTER";
   found_range := SEARCH_QUIETLY (chap, FORWARD, NO_EXACT);

   IF found_range = 0
   THEN
       MESSAGE ("Chapter not found.");
   ELSE
       POSITION (found_range);
   ENDIF;
ENDPROCEDURE;
```

This procedure searches for the word "CHAPTER" appearing at the beginning of a line. If the procedure finds the word, the procedure positions to the beginning of the string. If the procedure does not find the word, the procedure writes an appropriate message in the message buffer. Compare this example procedure to the corresponding procedure in the description of SEARCH.

```
PROCEDURE user_search_range
   LOCAL found_count;

   ON_ERROR
     [TPU$_CONTROLC]:
       MESSAGE ( FAO ("Found !SL occurrences.", found_count));
       RETURN;
     [OTHERWISE]:
       ABORT;
   ENDON_ERROR;

   found_count := 0;
   the_pattern := "blue skies";
   the_range := CREATE_RANGE (BEGINNING_OF (CURRENT_BUFFER),
                              END_OF (CURRENT_BUFFER), NONE);

   found_range := CREATE_RANGE (BEGINNING_OF (CURRENT_BUFFER),
                                BEGINNING_OF (CURRENT_BUFFER), NONE);

   LOOP
      the_range := CREATE_RANGE (END_OF (found_range),
                   END_OF (the_range), NONE);

      found_range := SEARCH_QUIETLY (the_pattern, FORWARD,
                     NO_EXACT, the_range);

      found_count := found_count + 1;
   ENDLOOP;
ENDPROCEDURE;
```

This procedure searches the range *the_range* for all occurrences of the pattern "blue skies". If SEARCH_QUIETLY finds the pattern, the procedure redefines *the_range* to begin after the end of the pattern just found. If the procedure did not redefine the range, SEARCH_QUIETLY would keep finding the first occurrence over and over. The procedure reports the number of occurrences of the pattern. Notice that a procedure using SEARCH_QUIETLY does not trap the TPU$_STRNOTFOUND error, because SEARCH_QUIETLY does not signal this error.

SELECT

Returns a marker for the editing point in the current buffer. You must specify how the marker is to be displayed on the screen (no special video, reverse video, bolded, blinking, or underlined).

The marker returned by SELECT indicates the first character position in a select range. The video attribute that you specify for the marker applies to all the characters in a select range. For information on creating a select range, see SELECT_RANGE.

FORMAT

marker := SELECT ({ *BLINK* | *BOLD* | *NONE* | *REVERSE* | *UNDERLINE* })

PARAMETERS

BLINK
Specifies that the selected characters are to blink.

BOLD
Specifies that the selected characters are to be bolded.

NONE
Applies no video attributes to selected characters.

REVERSE
Specifies that the selected characters are to be displayed in reverse video.

UNDERLINE
Specifies that the selected characters are to be underlined.

return value

A marker for the editing point in the current buffer.

DESCRIPTION

SELECT returns a special marker that establishes the beginning of a select range. The marker is positioned at the character position that is the editing point when the built-in procedure SELECT is executed. (The marker is actually positioned between character positions, rather than on a character position.) A select range includes all the characters between the select marker and the current position, but not the character at the current position.

Using SELECT may cause VAXTPU to insert padding spaces or blank lines in the buffer. SELECT causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a

line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

Only one select marker for a buffer can be active at any one time. If a buffer is associated with more than one visible window, the select range is displayed in only one window (the current or most recent window).

If the buffer in which you are selecting text is associated with the current window, as you move from the select marker to another character position in the same buffer, all the characters over which you move the cursor are included in the select range, and the video attribute that you specify for the select marker is applied to the characters in the range. The video attribute of a selected character is not visible when you are positioned on the character, but once you move beyond the character, the character is displayed with the attribute you specify.

If two or more windows are mapped to the same buffer and one of the windows is the current window, only the current window displays the select area. If two or more windows are mapped to different buffers, it is possible to have more than one visible select area on the screen at the same time. If none of the windows on the screen is the current window, the visible window that was most recently current displays the select area.

If the current character is deleted, the marker moves to the nearest character position. The nearest character position is determined in the following way:

1 If there is a character position on the same line to the right, the marker moves to this position, even if the position is at the end of the line.

2 If the line on which the marker is located is deleted, the marker moves to the first position on the following line.

If you are positioned at the select marker and you insert text, the select marker moves to the first character of the inserted text. You can move one column past the last character in a line. (With free cursor motion, you can move even further beyond the last character of a line.) However, if you establish a select marker beyond the last character in a line, no video attribute is visible for the marker.

SIGNALED ERRORS

TPU$_ONESELECT	WARNING	SELECT is already active in the current buffer.
TPU$_TOOFEW	ERROR	SELECT requires one argument.
TPU$_TOOMANY	ERROR	SELECT accepts only one argument.
TPU$_NEEDTOASSIGN	ERROR	SELECT must be on the right-hand side of an assignment statement.

TPU$_NOCURRENTBUF	ERROR	You must position to a buffer before using SELECT.
TPU$_BADKEY	WARNING	You specified the wrong keyword to SELECT.
TPU$_INVPARAM	ERROR	SELECT's parameter is not a keyword.

EXAMPLES

1
```
select_mark := SELECT (NONE)
```

This assignment statement places a marker at the editing point. Because NONE is specified, no video attributes are applied to a select range that has this marker as its beginning.

2
```
select_mark_under := SELECT (UNDERLINE)
```

This assignment statement places a marker at the editing point. Any characters included in a select range that has this marker as its beginning are underlined.

3
```
! Bold selected text
PROCEDURE user_start_select
   user_v_beginning_of_select := SELECT (BOLD);
ENDPROCEDURE;
```

This procedure creates a bold marker that is used as the beginning of a select region. As you move the cursor, the characters that you select are bolded. To turn off the selection of characters, set the variable *user_v_beginning_of_select* to 0.

SELECT_RANGE

Returns a range that contains all the characters between the marker established with the built-in procedure SELECT and the editing point. SELECT_RANGE does not include the current character.

FORMAT

range := SELECT_RANGE

PARAMETERS

None.

return value

A range containing all the characters between the marker established with SELECT and the editing point.

DESCRIPTION

If you select text in a forward direction, the select range includes the marked character and all characters up to but not including the current character. If you select text in a reverse direction from the marker, the select range includes the current character and all characters up to but not including the marked character.

SELECT_RANGE is used in conjunction with SELECT to allow the user to mark a section of text for treatment as an entity.

The procedure for selecting a section of text is the following:

1. Use the built-in procedure SELECT to place a marker at the beginning of the section you want to select. The following example illustrates:

```
m1 := SELECT (NONE);
```

2. Mark the characters that you want in the select region by moving from character to character with the cursor.

3. When all of the text is selected, create a range that contains the selected text. The following example illustrates:

```
r1 := SELECT_RANGE;
```

4. Stop the selection of characters by setting the marker that marks the beginning of the range to 0. The following example illustrates:

```
m1 := 0;
```

Using SELECT_RANGE may cause VAXTPU to insert padding spaces or blank lines in the buffer. SELECT_RANGE causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank

lines into the buffer to fill the space between the cursor position and the nearest text.

SIGNALED ERRORS

TPU$_NOSELECT	WARNING	There is no active select range in the current buffer.
TPU$_SELRANGEZERO	WARNING	The select range contains no selected characters.
TPU$_NEEDTOASSIGN	ERROR	SELECT_RANGE must be on the right-hand side of an assignment statement.
TPU$_TOOMANY	ERROR	SELECT_RANGE takes no arguments.
TPU$_NOCURRENTBUF	WARNING	There is no current buffer.

EXAMPLES

1
```
select_1 := SELECT_RANGE
```

This assignment statement puts the range for the currently selected characters in the variable *select_1*.

2
```
PROCEDURE user_select
! Start a select region
    user_select_position := SELECT (REVERSE);
    MESSAGE ("Selection started.");
! Move 5 lines and create a select region
    MOVE_VERTICAL (5);
    SR1 := SELECT_RANGE;
! Move 5 lines and create another select region
    MOVE_VERTICAL (5);
    SR2 := SELECT_RANGE;
! Stop the selection by setting the select marker to 0.
    user_select_position := 0;
ENDPROCEDURE;
```

This procedure shows the use of SELECT_RANGE multiple times in the same procedure.

SEND

Passes data to a subprocess.

FORMAT

SEND ({ *buffer* / *range* / *string* }, *process*)

PARAMETERS

buffer
The buffer whose contents you want to send to the subprocess.

range
The range whose contents you want to send to the subprocess.

string
The string that you want to send to the subprocess.

process
The process to which you want to send data.

DESCRIPTION

All output from the process is stored in the buffer that was associated with the process when you created it. See the CREATE_PROCESS built-in. Your editing stops until the process responds to what is sent.

If you specify a buffer or a range as the data to pass to a process, the lines of the buffer or range are sent as separate records.

SIGNALED ERRORS

TPU$_NOPROCESS	WARNING	Subprocess that you tried to send to has already terminated.
TPU$_SENDFAIL	WARNING	Unable to send input to a subprocess.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SEND built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SEND built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the SEND built-in.
TPU$_NOTMODIFIABLE	WARNING	Attempt to change unmodifiable buffer. The buffer to which a subprocess writes output must be modifiable.

TPU$_DELETEFAIL	WARNING	Unable to terminate the subprocess.
TPU$_NOSENDBUF	WARNING	Input buffer is the same as the output buffer.
TPU$_CONTROLC	ERROR	The execution of the command you sent terminated because you pressed CTRL/C.

EXAMPLES

1
```
SEND ("directory", user_process)
```

This statement sends the DCL command DIRECTORY to the process named *user_process*. The process must already be created with the built-in procedure CREATE_PROCESS so that the output can be stored in the buffer associated with the process.

2
```
PROCEDURE mail_subp

! Create a buffer and a window that a subprocess can run in

   v_mail_buffer := CREATE_BUFFER ("main_buffer");
   v_mail_window := CREATE_WINDOW (1, 22, ON);

! Map the mail window to the screen

   UNMAP (MAIN_WINDOW);
   MAP (v_mail_window, v_mail_buffer);

! Create a subprocess and send "mail" as the first command

   p1 := CREATE_PROCESS (v_mail_buffer, "MAIL");

! Position to the subprocess and use read_line for next command

   POSITION (v_mail_window);
   s1 := READ_LINE ("mail_subp> ", 20);
   SEND (s1, p1);

ENDPROCEDURE;
```

This procedure uses the built-in procedure SEND to pass a command to a process in which the Mail Utility is running. The command to be sent to the process is obtained with the built-in procedure READ_LINE.

SEND_CLIENT_MESSAGE

Sends either of two client messages—STUFF_SELECTION or KILL_SELECTION—to other DECwindows applications.

FORMAT

SEND_CLIENT_MESSAGE ({ *STUFF_SELECTION* / *KILL_SELECTION* })

PARAMETERS

KILL_SELECTION

A keyword indicating that the client message to be sent is the KILL_SELECTION client message.

SEND_CLIENT_MESSAGE (KILL_SELECTION) is used when the user wants to copy something from another application that owns the input focus to the VAXTPU/EVE-layered application without input focus. It also removes the text from that other application.

The user selects the text in the application that owns the input focus. The user positions the pointer cursor to the desired location in the VAXTPU /EVE-layered application and then presses the CTRL key and clicks on the MB3 mouse button. In this circumstance, the VAXTPU/EVE-layered application inserts the selected text and uses SEND_CLIENT_MESSAGE (KILL_SELECTION) to send a message directing the application that owns the input focus to delete the selected text.

STUFF_SELECTION

A keyword indicating that the client message to be sent is the STUFF_SELECTION client message.

SEND_CLIENT_MESSAGE (STUFF_SELECTION) is used when the user wants to copy something from an application layered on VAXTPU or EVE into some DECwindows application that owns the input focus.

To select the text to be copied, the user does an MB3DRAG operation in the VAXTPU/EVE-layered application. The VAXTPU/EVE-layered application grabs ownership of the secondary global selection. The selected text is the secondary global selection.

The VAXTPU/EVE-layered application then uses SEND_CLIENT_MESSAGE (STUFF_SELECTION) to send a STUFF_SELECTION client message to the application that owns the input focus. In response, the application requests to read the secondary global selection. This causes the VAXTPU/EVE-layered application to write to the secondary global selection, which is then received by the other application.

Note that if the user uses CTRL/MB3DRAG instead of MB3DRAG, the last step is that the text in the secondary global selection is deleted from the VAXTPU/EVE-based application.

DESCRIPTION

Note that the VAXTPU/EVE-layered application cannot designate the application that is to receive the client message. VAXTPU handles sending the message to the correct DECwindows application.

SIGNALED ERRORS

TPU$_NORETURNVALUE	ERROR	Does not return a value.
TPU$_TOOFEW	ERROR	SEND_CLIENT_MESSAGE requires one argument.
TPU$_TOOMANY	ERROR	SEND_CLIENT_MESSAGE accepts only one argument.
TPU$_BADKEY	WARNING	Keyword must be either KILL_SELECTION or STUFF_SELECTION.
TPU$_INVPARAM	ERROR	The parameter must be a keyword.
TPU$_NOGBLSELDATA	WARNING	There is no owner of the PRIMARY global selection to send a client message to.
TPU$_NOFOCUSOWNER	WARNING	There is no owner of the input focus to send a client message to.

SEND_EOF

Uses features of the VMS mailbox driver to send an end-of-file message (IO$_WRITEOF) to a process.

FORMAT

SEND_EOF *(process)*

PARAMETERS

process
The process to which the end-of-file message is being sent.

DESCRIPTION

The end-of-file message causes a pending read from a subprocess to be completed with an SS$_ENDOFFILE status. See the *VMS I/O User's Reference Volume* for more information on the Write End-of-File message.

SIGNALED ERRORS

TPU$_SENDFAIL	WARNING	Unable to send input to a subprocess.
TPU$_NOPROCESS	WARNING	No subprocess to send to.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SEND_EOF built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SEND_EOF built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the SEND_EOF built-in.
TPU$_NOTMODIFIABLE	WARNING	Attempt to change unmodifiable buffer. The buffer to which a subprocess writes output must be modifiable.
TPU$_DELETEFAIL	WARNING	Unable to terminate the subprocess.

EXAMPLE

```
SEND_EOF (sub_proc1)
```

This statement sends an end-of-file to *sub_proc1*.

SET

Lets you establish or change certain features of a VAXTPU session. SET requires a keyword as its first parameter. The keyword indicates which feature of the editor is being set. You can set the mode for entering text, the text that is to be displayed on certain parts of the screen, the direction of a buffer, the status of a buffer, and so on.

FORMAT

SET *(keyword, parameter [,...])*

PARAMETERS

keyword

The keyword used as the first parameter specifies which feature is being established or changed. Following are the valid keywords for SET:

ACTIVE_AREA
AUTO_REPEAT
BELL
CLIENT_MESSAGE
COLUMN_MOVE_VERTICAL
CROSS_WINDOW_BOUNDS
DEBUG
DEFAULT_DIRECTORY
DETACHED_ACTION
DISPLAY_VALUE
DRM_HIERARCHY
ENABLE_RESIZE
EOB_TEXT
ERASE_UNMODIFIABLE
FACILITY_NAME
FORWARD
GLOBAL_SELECT
GLOBAL_SELECT_GRAB
GLOBAL_SELECT_READ
GLOBAL_SELECT_TIME
GLOBAL_SELECT_UNGRAB
HEIGHT
ICON_NAME
ICON_PIXMAP
ICONIFY_PIXMAP
INFORMATIONAL
INPUT_FOCUS
INPUT_FOCUS_GRAB
INPUT_FOCUS_UNGRAB
INSERT
JOURNALING
KEY_MAP_LIST
KEYSTROKE_RECOVERY
LEFT_MARGIN
LEFT_MARGIN_ACTION
LINE_NUMBER
MAPPED_WHEN_MANAGED
MARGINS

MAX_LINES
MENU_POSITION
MESSAGE_ACTION_LEVEL
MESSAGE_ACTION_TYPE
MESSAGE_FLAGS
MODIFIABLE
MODIFIED
MOUSE
NO_WRITE
OUTPUT_FILE
OVERSTRIKE
PAD
PAD_OVERSTRUCK_TABS
PERMANENT
POST_KEY_PROCEDURE
PRE_KEY_PROCEDURE
PROMPT_AREA
RECORD_ATTRIBUTE
RESIZE_ACTION
REVERSE
RIGHT_MARGIN
RIGHT_MARGIN_ACTION
SCREEN_LIMITS
SCREEN_UPDATE
SCROLL_BAR
SCROLL_BAR_AUTO_THUMB
SCROLLING
SELF_INSERT
SHIFT_KEY
SPECIAL_ERROR_SYMBOL
STATUS_LINE
SUCCESS
SYSTEM
TAB_STOPS
TEXT
TIMER
TRACEBACK
UNDEFINED_KEY
VIDEO
WIDGET
WIDGET_CALL_DATA
WIDGET_CALLBACK
WIDTH

These keywords and the parameters that follow them are described on the following pages. The descriptions of the keywords are organized alphabetically.

parameter [, . . .]

The number of parameters following the first parameter varies according to the keyword you use. The parameters are listed in the format section of the applicable keyword description.

DESCRIPTION

The built-in procedure SET can be used by both the programmer creating an editing interface and the person using the interface. The programmer can establish certain default behavior and screen displays for an editing interface. The user can change the default behavior and do some simple customizing of an existing VAXTPU interface with the built-in procedure SET.

SET (ACTIVE_AREA)

Designates the specified area as the active area in a VAXTPU window. An active area is an area within which VAXTPU ignores movements of the pointer cursor.

FORMAT

SET *(ACTIVE_AREA, window, column, row [, width, height])*

PARAMETERS

ACTIVE_AREA
A keyword directing VAXTPU to set an attribute of the active area.

window
The window in which you want to define the active region.

column
An integer specifying the leftmost column of the active region.

row
An integer specifying the topmost row of the active region. If you use 0, the active row is the status line.

width
An integer specifying the width in columns of the active region. Defaults to 1.

height
An integer specifying the height in rows of the active region. Defaults to 1.

DESCRIPTION

The active area is the region in a window in which VAXTPU ignores movements of the pointer cursor for purposes of distinguishing clicks from drags. When you press down a mouse button, VAXTPU interprets the event as a click if the upstroke occurs in the active area with the downstroke. If the upstroke occurs outside the active area, VAXTPU interprets the event as a drag operation.

A VAXTPU layered application can have only one active area at a time, even if the application has more than one window visible on the screen. An active area is only valid if you are pressing a mouse button. The default active area occupies one character cell. By default, the active area is located on the character cell pointed to by the pointer cursor.

For information on mouse button clicks, see the *XUI Style Guide*.

Table 7–9 lists the keywords for referring to click and drag operations.

Table 7–9 VAXTPU Keywords Representing Mouse Events

M1UP	M2UP	M3UP	M4UP	M5UP
M1DOWN	M2DOWN	M3DOWN	M4DOWN	M5DOWN
M1DRAG	M2DRAG	M3DRAG	M4DRAG	M5DRAG
M1CLICK	M2CLICK	M3CLICK	M4CLICK	M5CLICK
M1CLICK2	M2CLICK2	M3CLICK2	M4CLICK2	M5CLICK2
M1CLICK3	M2CLICK3	M3CLICK3	M4CLICK3	M5CLICK3
M1CLICK4	M2CLICK4	M3CLICK4	M4CLICK4	M5CLICK4
M1CLICK5	M2CLICK5	M3CLICK5	M4CLICK5	M5CLICK5

SIGNALED ERRORS

TPU$_BADVALUE	ERROR	An integer parameter was specified with a value outside the valid range.
TPU$_EXTRANEOUSARGS	ERROR	One or more extraneous arguments has been specified for a DECwindows built-in.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (ACTIVE_AREA) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (ACTIVE_ AREA) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (ACTIVE_AREA) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (ACTIVE_AREA) built-in.

EXAMPLE

```
PROCEDURE eve$$m1down

LOCAL   the_window,
        the_column,
        the_row,
        the_width;

ON_ERROR
    [OTHERWISE]:
ENDON_ERROR;

eve$$x_pre_mb1_mark := MARK (FREE_CURSOR);
```

```
IF LOCATE_MOUSE (the_window, the_column, the_row)
THEN
    eve$x_mb1_in_progress := 1;
    IF the_row = 0
    THEN
        IF eve$current_indicator (the_window,
                                  the_column,
                                  the_width) <> 0
        THEN
            IF eve$x_decwindows_active
            THEN
                SET (ACTIVE_AREA,               ! This statement sets
                     the_window, the_column,    ! the active area.
                     0, the_width, 1);
            ENDIF;
        ELSE
            RETURN (FALSE);
        ENDIF;
    ELSE
        IF the_window = eve$choice_window
        THEN
            IF eve$$current_choice (the_column, eve$$x_chosen_range)
            THEN
                IF eve$x_decwindows_active
                THEN
                    SET (ACTIVE_AREA, the_window, the_column, the_row,
                         eve$$x_choices_column_width, 1);
                ENDIF;
            ENDIF;
        else
            POSITION (MOUSE);
            eve$$x_mb1_down_free := MARK (FREE_CURSOR);
            POSITION (TEXT);
            eve$clear_select_position;
            eve$clear_message;
            eve$$x_mb1_down_bound := MARK (NONE);
            POSITION (eve$$x_mb1_down_free);
        ENDIF;
    ENDIF;
    RETURN (TRUE);
ELSE
    RETURN (FALSE);
ENDIF;

ENDPROCEDURE;
```

This procedure shows one possible way that an application can use SET (ACTIVE_AREA). The procedure is a modified version of the EVE procedure EVE$$M1DOWN. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU.

Procedure EVE$$M1DOWN, when bound to M1DOWN, sets an active area when you press MB1.

SET (AUTO_REPEAT)

FORMAT

SET *(AUTO_REPEAT,* { *ON* / *OFF* } *)*

PARAMETERS

AUTO_REPEAT

A keyword indicating that SET is to control whether VAXTPU repeats keystrokes as long as you hold down a key.

By default, AUTO_REPEAT is set ON.

ON

Specifies that a key press should continue to generate characters until the key is released.

OFF

Requires a separate keystroke for each character generated.

DESCRIPTION

VAXTPU sends an escape sequence to the terminal to set AUTO_REPEAT on or off.

The autorepeat feature affects all keyboard keys on the VT100 series of terminals except the following:

- The SET-UP key
- The ESC key
- The NO SCROLL key
- The TAB key
- The RETURN key
- The CTRL key and another key

The autorepeat feature affects all keyboard keys on the VT300 series and VT200 series of terminals except the following:

- The keys F1, F2, F3, F4, F5
- The RETURN key

If you want to slow down the movement of the cursor, you can use SET (AUTO_REPEAT) within a procedure that causes cursor motion. Because of the time the terminal requires to process the escape sequence that VAXTPU sends, if you turn autorepeat off before moving the cursor and on after the movement, you slow down the cursor movement. You may find it useful to slow the cursor motion at the top or bottom of a window. The sample procedure in the Example section shows how to do this. See Example 2.

SET (AUTO_REPEAT) has no effect if you use it in DECwindows VAXTPU.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (AUTO_REPEAT) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	The keyword must be either ON or OFF.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.

EXAMPLES

1
```
SET (AUTO_REPEAT, OFF)
```

This statement turns autorepeat off.

2
```
! Two procedures that slow the scrolling action

PROCEDURE user_slow_up_arrow
   SET (AUTO_REPEAT, OFF);
   MOVE_VERTICAL (-1);
   SET (AUTO_REPEAT, ON);
ENDPROCEDURE;

PROCEDURE user_slow_down_arrow
   SET (AUTO_REPEAT, OFF);
   MOVE_VERTICAL (1);
   SET (AUTO_REPEAT, ON);
ENDPROCEDURE;
```

These procedures show how to turn AUTO_REPEAT off and on to slow the cursor movement.

SET (BELL)

FORMAT

SET *(BELL,* { *ALL* / *BROADCAST* }*,* { *ON* / *OFF* }*)*

PARAMETERS

BELL

The terminal bell.

ALL

Indicates that the second parameter (ON or OFF) applies to all messages.

BROADCAST

Indicates that the second parameter applies to broadcast messages only.

ON

Causes the terminal bell to ring when a message is written to the message window.

OFF

Turns off the audible signal of the terminal bell.

DESCRIPTION

When the bell is on, the terminal bell rings to signal the fact that a message is being written to the message window. When you use ALL, internal VAXTPU messages as well as broadcast messages cause the terminal bell to ring. To cause VAXTPU messages of success and informational severity level to be written to the message buffer, you must have used the built-in procedure SET ({INFORMATIONAL | SUCCESS}, ON). When you use BROADCAST, only broadcast messages such as mail notifications and REPLY messages cause the bell to ring.

SET (BELL, ALL, {ON | OFF}) affects the setting of SET (BELL, BROADCAST, {ON | OFF}). If you want the behavior of broadcast messages to be different from other messages, use the built-in procedure SET (BELL, BROADCAST, {ON | OFF}) after using SET (BELL, ALL, {ON | OFF}).

Note that VAXTPU causes the bell to ring as a signal that a message is being written to the message window, not as an interpretation of a bell character in the message text. Bell characters in the message text are not interpreted, they are displayed. Positioning to the message window and moving the cursor to a bell character in the message text do not cause the terminal bell to ring.

You can also use DCL commands to affect the display of broadcast messages within VAXTPU. If you use the command SET TERMINAL /NOBROADCAST at the DCL level, no broadcast messages are sent to your terminal. The DCL command SET BROADCAST allows you to enable or disable certain classifications of broadcast messages.

The bell is off by default.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (BELL) requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLES

1
```
SET (BELL, BROADCAST, ON)
```

This statement causes the terminal bell to ring when a broadcast message is written to the message window.

2
```
PROCEDURE user_ring_bell (msg_string)

   SET (BELL, ALL, ON);       ! Turn bell on
   MESSAGE (msg_string);      ! Write message text to message buffer
   SET (BELL, ALL, OFF);      ! Turn bell off
   SET (BELL, BROADCAST,ON); ! Turn bell on for broadcast messages
ENDPROCEDURE;
```

This procedure uses SET (BELL, ALL, ON) to cause the bell to ring for the message that is being sent in the second statement. After the message is written, the bell is turned off. SET (BELL, BROADCAST, ON) is used to cause broadcast messages to ring the terminal bell.

SET (CLIENT_MESSAGE)

Designates the action routine to be executed when DECwindows VAXTPU receives a client message from another DECwindows application.

FORMAT

SET *(CLIENT_MESSAGE,SCREEN,* { *buffer* | *learn_sequence* | *program* | *range* | *string* } *)*

PARAMETERS

CLIENT_MESSAGE
A keyword indicating that SET is being used to designate a client message action routine.

SCREEN
A keyword used to preserve compatibility with future versions of VAXTPU.

buffer
The buffer containing the code to be executed when VAXTPU receives a client message.

learn_sequence
The learn sequence to be executed when VAXTPU receives a client message.

program
The program to be executed when VAXTPU receives a client message.

range
The range containing the code to be executed when VAXTPU receives a client message.

string
The string containing the code to be executed when VAXTPU receives a client message.

DESCRIPTION

A client message is a communication from one DECwindows application to another. The message enables the sending application to generate an event on the queue of the receiving application.

SIGNALED ERRORS

TPU$_COMPILEFAIL	WARNING	Compilation failed.
TPU$_TOOFEW	ERROR	You specified too few parameters.

TPU$_TOOMANY	ERROR	You specified too many parameters.
TPU$_BADKEY	ERROR	You specified an invalid keyword.
TPU$_ARGMISMATCH	ERROR	Argument has the wrong type.

SET (COLUMN_MOVE_VERTICAL)

FORMAT

SET *(COLUMN_MOVE_VERTICAL,* { *ON* / *OFF* } *)*

PARAMETERS

COLUMN_MOVE_VERTICAL

Specifies that you want to use SET to control how the MOVE_VERTICAL built-in moves the cursor.

ON

Directs the MOVE_VERTICAL built-in to place the cursor in the same column on each new line unless doing so would put the cursor in the middle of a tab. If the cursor would be placed in a tab, MOVE_VERTICAL places the cursor at the beginning of the tab.

OFF

Directs the MOVE_VERTICAL built-in to place the cursor at the same offset in each new record to which the cursor moves. This behavior is the default for SET (COLUMN_MOVE_VERTICAL). Since VAXTPU counts a tab as one character when determining the offset, the cursor's column location can change dramatically after you use MOVE_VERTICAL.

DESCRIPTION

When SET (COLUMN_MOVE_VERTICAL) is set to ON, you can get a different result from using MOVE_VERTICAL (n) than from using MOVE_VERTICAL (1) *n* times. When you use MOVE_VERTICAL (3), for example, the built-in tries to keep the cursor in the column the cursor occupied just before execution of MOVE_VERTICAL (3). When you use MOVE_VERTICAL (1) three times, the built-in resets the column where VAXTPU is trying to keep the cursor. Thus, if the first MOVE_VERTICAL (1) moves the cursor leftward to the beginning of a tab, the second MOVE_VERTICAL (1) does not move the cursor rightward again.

When SET (COLUMN_MOVE_VERTICAL) is set to OFF, MOVE_VERTICAL (n) produces the same results as MOVE_VERTICAL (1) n times.

To determine whether COLUMN_MOVE_VERTICAL is set to ON or OFF, use the following statement:

```
boolean := GET_INFO (SYSTEM, "COLUMN_MOVE_VERTICAL")
```

This GET_INFO call returns 1 if COLUMN_MOVE_VERTICAL is set to ON, 0 if it is set to OFF.

If you have previously written extensions to EVE and want to layer the extensions on EVE, you may have to rewrite some procedures because EVE sets COLUMN_MOVE_VERTICAL to ON. For instance, if your extension contains the following code and if the first line has a left margin further to the right than the second line, the code may not work as intended:

```
MOVE_HORIZONTAL (-CURRENT_OFFSET); ! Go to beginning of line
MOVE_VERTICAL (1);                 ! Move down a line
```

To compensate for the fact that EVE sets COLUMN_MOVE_VERTICAL to ON, you can substitute the following code for the code shown above:

```
POSITION (LINE_END);          ! Go to end of existing line
MOVE_HORIZONTAL (1);          ! Advance to start of next line
```

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (COLUMN_MOVE_VERTICAL) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	The keyword must be either ON or OFF.

EXAMPLES

In the following example, the symbol ">" represents a tab character. The underscore shows the cursor location.

Suppose you have the following two lines of text in a buffer, with the cursor on the "c" in the first line:

```
abcdefg
a>......bcdefg
```

If you use the following code, the cursor ends up pointing to the "b" on the second line:

```
SET (COLUMN_MOVE_VERTICAL, OFF);
MOVE_VERTICAL (1);
```

After the MOVE_VERTICAL (1) statement, the cursor location is as follows:

```
abcdefg
a>......bcdefg
```

On the other hand, suppose you have the same text, as follows:

```
abcdefg
a>......bcdefg
```

If you use the following code, the cursor ends up pointing to the beginning of the tab on the second line:

```
SET (COLUMN_MOVE_VERTICAL, ON);
MOVE_VERTICAL (1);
```

After the MOVE_VERTICAL (1) statement, the cursor location is as follows:

```
abcdefg
a>......bcdefg
```

SET (CROSS_WINDOW_BOUNDS)

FORMAT

SET *(CROSS_WINDOW_BOUNDS,* { *ON* / *OFF* } *)*

PARAMETERS

CROSS_WINDOW_BOUNDS

A keyword specifying that SET is to control the way the CURSOR_VERTICAL built-in procedure behaves at a window boundary.

The default setting for CROSS_WINDOW_BOUNDS is ON (preserving the behavior from previous versions of VAXTPU).

ON

Causes the CURSOR_VERTICAL built-in procedure to cross window boundaries and to ignore scrolling regions. However, even when crossing of window bounds is enabled, the CURSOR_VERTICAL built-in procedure still obeys screen boundaries. That is, if CURSOR_VERTICAL brings the cursor to the edge of the screen, VAXTPU scrolls text into the window rather than making the cursor invisible.

OFF

Prevents the CURSOR_VERTICAL built-in procedure from crossing window boundaries and causes CURSOR_VERTICAL to obey scrolling regions.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (CROSS_WINDOW_BOUNDS) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (CROSS_WINDOW_BOUNDS, OFF)
```

This statement prevents subsequent invocations of the CURSOR_VERTICAL built-in procedure from crossing window boundaries and causes the screen to scroll if the cursor moves into a scrolling region. This is the setting that the EVE editor now uses.

SET (DEBUG)

Controls various attributes of a debugging program that helps locate VAXTPU programming errors.

Note that this built-in has five valid syntax permutations. You cannot use any combinations of parameters not shown in this description.

FORMAT

SET *(DEBUG, PROGRAM,* { *buffer* / *program* / *range* / *string1* } *)*

PARAMETERS

DEBUG

A keyword indicating that SET is to control various attributes of a debugging program that helps locate VAXTPU programming errors.

PROGRAM

A keyword indicating that VAXTPU is to use a user-written debugger.

buffer

An expression evaluating to a buffer that contains a procedure or program.

The statement SET (DEBUG, PROGRAM, *buffer*) directs VAXTPU to use the user-written debugger contained in the specified buffer during the current debugging session.

program

A variable of type program.

The statement SET (DEBUG, PROGRAM, *program*) directs VAXTPU to use the user-written debugger contained in the specified program during the current debugging session.

range

An expression evaluating to a range that contains a procedure or program.

The statement SET (DEBUG, PROGRAM, *range*) directs VAXTPU to use the user-written debugger contained in the specified range during the current debugging session.

string1

A string containing executable VAXTPU statements.

The statement SET (DEBUG, PROGRAM, *string1*) directs VAXTPU to use the VAXTPU statements in the specified string during the current debugging session.

FORMAT

SET *(DEBUG,* { *ON* / *OFF* } *)*

PARAMETERS

DEBUG

A keyword indicating that SET is to control various attributes of a debugging program that helps locate VAXTPU programming errors.

ON

A keyword that enables single-stepping.

The statement SET (DEBUG, ON) directs VAXTPU to execute just one line of code and then return control to the debugger.

OFF

A keyword that disables single-stepping.

The statement SET (DEBUG, OFF) disables single-step execution. Since single-stepping is off by default, this format is useful only to turn off single-stepping after single-stepping has been turned on.

FORMAT

SET *(DEBUG,* { *ON* / *OFF* }*, string2)*

PARAMETERS

DEBUG

A keyword indicating that SET is to control various attributes of a debugging program that helps locate VAXTPU programming errors.

ON

A keyword that sets a breakpoint.

The statement SET (DEBUG, ON, *string2*) directs VAXTPU to set a breakpoint at the procedure named by *string2*.

OFF

A keyword that cancels one or more breakpoints.

The statement SET (DEBUG, OFF, *string2*) cancels a breakpoint previously set at the procedure named by *string2*.

string2

The name of a procedure.

The format SET (DEBUG, ON, *string2*) or SET (DEBUG, OFF, *string2*) sets or cancels a breakpoint at the procedure specified by *string2*.

FORMAT

SET *(DEBUG, OFF, ALL)*

PARAMETERS

DEBUG

A keyword indicating that SET is to control various attributes of a debugging program that helps locate VAXTPU programming errors.

OFF

A keyword that cancels breakpoints.

The statement SET (DEBUG, OFF, ALL) cancels all breakpoints set during the debugging session.

ALL

A keyword indicating that all breakpoints are to be canceled.

The statement SET (DEBUG, OFF, ALL) clears all breakpoints.

FORMAT

SET *(DEBUG, string3, value)*

PARAMETERS

DEBUG

A keyword indicating that SET is to control various attributes of a debugging program that helps locate VAXTPU programming errors.

string3

The name of a global variable, local variable, or parameter. When you use *string3* to specify a local variable or a parameter, the variable or parameter must be in the procedure you are currently debugging.

The statement SET (DEBUG, *string3*, *value*) deposits the specified value in the variable or parameter specified by *string3*.

value

A value of any data type in VAXTPU.

The statement SET (DEBUG, *string*, *value*) deposits the specified value in the global variable, local variable, or parameter named by the string.

DESCRIPTION

You use the SET (DEBUG) built-in when you are writing or using user-written debuggers. You cannot freely mix parameters when using SET (DEBUG). The only valid usages are those shown in the format sections of this description.

SIGNALED ERRORS

TPU$_NOCURRENTBUF	WARNING	There is no current buffer.
TPU$_NONAMES	WARNING	No names match the one requested.
TPU$_BADKEY	ERROR	An unknown keyword has been used as an argument.
TPU$_ARGMISMATCH	ERROR	You have specified an unsupported data type.

EXAMPLES

1
```
SET (DEBUG, ON, "user_remove")
```

This statement causes the debugger to be invoked each time the procedure "user_remove" is called.

2
```
SET (DEBUG, PROGRAM, "user_debugger")
```

This statement causes the user-written program "user_debugger" to be called as the program to help locate programming errors.

3
```
PROCEDURE debugon

SET (DEBUG, PROGRAM, "tpu$$debug");
BREAK;
ENDPROCEDURE;

debugon;
```

This procedure and statement from the VAXTPU debugger are compiled and executed when the user specifies /DEBUG on the DCL command line. The BREAK statement suspends execution of the debugger program and directs the debugger to wait for a debugging command from the user.

4
```
SET (DEBUG, "user_x_count", 42);
```

This statement sets the value of the variable *user_x_count* to 42.

SET (DEFAULT_DIRECTORY)

Determines the directory that will be used as the default.

FORMAT

[old_default_string :=] SET *(DEFAULT_DIRECTORY, new_default_string)*

PARAMETERS

DEFAULT_DIRECTORY

A keyword indicating that the SET built-in is being used to control which directory is used as the default.

new_default_string

A string naming the directory to which you want the default changed.

DESCRIPTION

When the user exits from VAXTPU, the default directory is not restored to the default that was set when the user invoked VAXTPU.

Note that when the user issues the EVE command DCL SHOW DEFAULT, the default shown is not always the new default directory, even though the setting has actually been changed. To update DCL tracking of the current default directory, use the EVE command DCL SET DEFAULT instead of calling this built-in procedure directly.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_TOOFEW	ERROR	SET (DEFAULT_DIRECTORY) requires two parameters.
TPU$_SYSERROR	ERROR	One of the system routines used has failed. The system routine's error message will be in the message buffer.
TPU$_INVPARAM	ERROR	The second parameter must be a string.
TPU$_PARSEFAIL	WARNING	Parameter is not a valid RMS file specification.

EXAMPLE

```
prev_dir := SET (DEFAULT_DIRECTORY, "DISK1:[WALSH.PINK]");
```

This statement sets the default directory to [WALSH.PINK] on the device DISK1. The variable *prev_dir* contains the string naming the previous default directory.

SET (DETACHED_ACTION)

Specifies the code to be executed when the VAXTPU main input loop detects that the current cursor position is detached (that is, that the cursor position cannot accurately represent the editing point in the current window).

FORMAT

SET *(DETACHED_ACTION, SCREEN*
[[, { *buffer* / *learn* / *program* / *range* / *string* } *]])*

PARAMETERS

DETACHED_ACTION

A keyword indicating that the SET built-in is being used to designate the detached cursor action routine.

SCREEN

A keyword indicating that the detached action routine is being set for all buffers and windows used during the session.

buffer

The buffer containing the detached cursor action routine.

learn

The learn sequence that is executed as the detached cursor action routine.

program

The program containing the detached cursor action routine.

range

The range containing the detached cursor action routine.

string

The string containing the detached cursor action routine.

DESCRIPTION

If VAXTPU determines that the current editing point is on a record that is not visible in the current window, the screen updater positions the cursor on the next visible record, placing the cursor in the comparable screen column. This condition is known as a "detached cursor." Use SET (DETACHED_ACTION) to designate code to be executed when the cursor is detached.

There are five reasons for a detached cursor. The following table shows these reasons, along with their constants and integers.

Constant	Value	Reason
TPU$K_OFF_LEFT	1	The editing point is off the left side of the current window.
TPU$K_OFF_RIGHT	2	The editing point is off the right side of the current window.
TPU$K_INVISIBLE	4	The editing point is on a record that is invisible in the current window.
TPU$K_DISJOINT	8	The current buffer is not mapped to the current window.
TPU$K_UNMAPPED	16	No current window exists.

If you do not specify the optional third parameter, SET (DETACHED_ACTION) deletes the current detached action routine.

To fetch the current detached action routine, use GET_INFO (SCREEN, "detached_action"). To find out which of the five possible detached states the cursor is in, use GET_INFO (SCREEN, "detached_reason").

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	You specified too many parameters.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_INVPARAM	ERROR	The second parameter must be a keyword.
TPU$_ARGMISMATCH	ERROR	The third parameter must be a program or a learn key sequence.
TPU$_BADKEY	WARNING	The second parameter must be SCREEN.
TPU$_COMPILEFAIL	WARNING	The third parameter did not compile successfully.
TPU$_COMPILED	SUCCESS	The third parameter successfully compiled.

EXAMPLE

```
PROCEDURE detached_routine

LOCAL rightmost_column,
      the_offset;

IF GET_INFO (SCREEN, "detached_reason") < > tpu$k_off_right

   THEN RETURN;

ENDIF;

rightmost_column := GET_INFO (CURRENT_WINDOW, "right", VISIBLE_TEXT);

the_offset := GET_INFO (CURRENT_BUFFER, "offset_column");

IF the_offset > rightmost_column

   THEN SHIFT (CURRENT_WINDOW, the_offset - rightmost_column + 2)

ENDIF;

UPDATE (CURRENT_WINDOW);

ENDPROCEDURE;
```

Given this definition of the procedure "detached_routine", the following statement designates this procedure as an application's detached action routine:

```
SET (DETACHED_ACTION, SCREEN, "detached_routine");
```

This detached action routine shifts the current window to the right if the editing point is to the right of the last displayed column.

SET (DISPLAY_VALUE)

Sets the display value of the specified window.

FORMAT

SET *(DISPLAY_VALUE,window,display_value_integer)*

PARAMETERS

DISPLAY_VALUE

A keyword indicating that the SET built-in is being used to set the display value for a window.

window

The window whose display value you want to set.

display_value_integer

An integer from -127 to +127.

DESCRIPTION

VAXTPU uses a window's display value, which is an integer value, to determine if a given record in a buffer should be made visible in the window mapped to the buffer. If the record's display value is greater than or equal to the window's setting, VAXTPU makes the record visible in that window; otherwise, VAXTPU makes the record invisible.

The record's display values are set by using the SET (RECORD_ATTRIBUTES) built-in procedure.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	You specified too many parameters.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_INVPARAM	ERROR	The second parameter must be a window.
TPU$_BADDISPVAL	WARNING	Display values must be between -127 and +127.

EXAMPLE

```
SET (DISPLAY_VALUE, CURRENT_WINDOW, 10);
```

This statement gives the current window a display value of 10. This means that any record whose display value is less than 10 is invisible in the specified window.

SET (DRM_HIERARCHY)

Sets the User Interface Definition (UID) file or files to be used with VAXTPU.

FORMAT

integer := *SET (DRM_HIERARCHY, filespec*
[[, filespec...]])

PARAMETERS

filespec

A string specifying the UID file to be used. VAXTPU applies the VMS default file specification *"SYS$LIBRARY: .UID"* to the string you pass to SET (DRM_HIERARCHY). You must specify at least one file name.

return value

An integer that is the identification number for the XUI Resource Manager hierarchy.

DESCRIPTION

Using UID files to specify hierarchies makes it easy to translate the product into other languages and to modify an application's interface without recompiling all the code implementing the application.

For more information about UID files, see the *VMS DECwindows Guide to Application Programming*.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by the SET (DRM_HIERARCHY) built-in.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (DRM_HIERARCHY) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (DRM_HIERARCHY) built-in.
TPU$_FAILURE_STATUS	ERROR	The Digital Resource Manager returned an error status.
TPU$_INVPARAM	ERROR	You specified an invalid parameter.
TPU$_REQUIRESDECW	ERROR	Requires the VAXTPU DECwindows screen updater.

EXAMPLE

The following statement designates the User Interface Definition (UID) file MYNODE$DUA0:[SMITH]EXAMPLE.UID as a file to be used with VAXTPU to create widgets needed by the layered application:

```
example_hierarchy := SET (DRM_HIERARCHY, "mynode$dua0:[smith]example.uid");
```

For examples of how the SET (DRM_HIERARCHY) built-in is used in procedures, see Example B–1 and Example B–2.

SET (ENABLE_RESIZE)

Enables or disables resizing of the VAXTPU screen.

FORMAT

SET *(ENABLE_RESIZE,* { *ON* / *OFF* } *)*

PARAMETERS

ENABLE_RESIZE

A keyword directing VAXTPU to enable or disable screen resizing.

ON

A keyword enabling screen resizing.

OFF

A keyword disabling screen resizing.

DESCRIPTION

If you specify the ON keyword, VAXTPU gives the DECwindows window manager hints (parameters that the window manager is free to use or ignore) on the allowable maximum and minimum sizes for the VAXTPU screen. The hints are set by the SET (SCREEN_LIMITS, array) built-in. If you specify the OFF keyword, VAXTPU uses the screen's current width and length as the maximum and minimum size.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (ENABLE_RESIZE) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (ENABLE_RESIZE) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (ENABLE_RESIZE) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (ENABLE_RESIZE) built-in.

EXAMPLE

```
SET (ENABLE_RESIZE, ON);
```

This statement enables screen resizing. To see this statement used in an initializing procedure, see the example in the description of the SET (SCREEN_LIMITS) built-in.

SET (EOB_TEXT)

FORMAT

SET *(EOB_TEXT, buffer, string)*

PARAMETERS

EOB_TEXT

A keyword indicating that SET is to determine the text displayed at the end of a buffer. This text is merely a visual marker in a buffer and does not become part of the file that is written when a buffer is saved.

The default end-of-buffer text is "[EOB]."

buffer

The buffer in which the text for the end-of-buffer is being set.

string

The text that is displayed to indicate the end-of-buffer.

DESCRIPTION

You may specify ranges that include the end-of-buffer text, but you cannot set the record_attributes of the end-of-buffer "record." Therefore, the end-of-buffer text is always visible, is left-justified on the screen, and cannot be modified using normal editing operations.

Setting a blank EOB_TEXT is the only way to "remove" the end-of-buffer text. Note, however, that a blank line will still remain.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	This SET built-in requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_FAILURE	FATAL	VAXTPU could not create the record for the EOB text.

EXAMPLE

```
SET (EOB_TEXT, main_buffer, "[END OF MAIN EDITING BUFFER]")
```

This statement causes [END OF MAIN EDITING BUFFER] to be displayed as the end-of-buffer text for the main buffer.

SET (ERASE_UNMODIFIABLE)

Controls whether VAXTPU erases unmodifiable records in response to built-ins that delete lines from a buffer.

FORMAT

{ ON / OFF } := SET *(ERASE_UNMODIFIABLE, buffer* { *ON* / *OFF* }*)*

PARAMETERS

ERASE_UNMODIFIABLE

A keyword indicating that the SET built-in is being used to control whether unmodifiable records are deleted in response to built-ins that erase lines in a buffer.

buffer

The buffer for which you want to turn on or turn off erasing of unmodifiable records.

ON

A keyword enabling erasing of unmodifiable records.

OFF

A keyword disabling erasing of unmodifiable records.

return value

The keyword ON or OFF, indicating the previous setting of ERASE_UNMODIFIABLE.

DESCRIPTION

By default, unmodifiable records can be deleted from buffers by built-ins such as ERASE_LINE. For example, ERASE_LINE deletes an unmodifiable record only if ERASE_UNMODIFIABLE is turned on. If ERASE_UNMODIFIABLE is turned off when ERASE_LINE or a similar built-in encounters an unmodifiable record, the built-in returns an error and does not delete the record.

However, some built-ins delete records as a side effect of their normal action. Table 7–10 shows the built-ins that can delete records as a side effect and shows what these built-ins do instead when the ERASE_UNMODIFIABLE setting is turned off. The SET (ERASE_UNMODIFIABLE) built-in prevents these built-ins from unintentionally deleting unmodifiable records.

Table 7–10 Selected Built-in Actions When ERASE_UNMODIFIABLE is Turned Off

Built-in	Action
APPEND_LINE	Signals a warning if an attempt is made to append to an unmodifiable line.
CHANGE_CASE	Signals a warning if any of the lines in the range or buffer are unmodifiable.
COPY_TEXT	Copies all records, preserving modifiability attribute while in insert mode.
	Signals a warning if the current editing position is in an unmodifiable line.
	Signals a warning if in overstrike mode and any of the lines to be overstruck are unmodifiable.
EDIT	Signals a warning if any of the lines in the range or buffer are unmodifiable.
ERASE (buffer)	Signals a warning if any line in the buffer is unmodifiable.
ERASE (range)	Signals a warning if the start or the end of the range is in the middle of an unmodifiable line.
	Signals a warning if any of the lines in the range are unmodifiable.
ERASE_ CHARACTER	Signals a warning if the current character is unmodifiable.
ERASE_LINE	Signals a warning if the current line is unmodifiable.
FILL	Signals a warning if any of the lines in the range or buffer are unmodifiable.
MOVE_TEXT	Moves all records, preserving modifiability attribute while in insert mode.
	Signals a warning if the current editing point is in an unmodifiable line.
	Signals a warning if in overstrike mode and any of the lines to be overstruck are unmodifiable.
	If the start or the end of the range is in the middle of an unmodifiable line, the MOVE_TEXT is turned into a COPY_ TEXT and a warning is issued.
	If any of the lines in the buffer or range are unmodifiable, the MOVE_TEXT is turned into a COPY_TEXT and a warning is issued.
SPLIT_LINE	Signals a warning if the current editing position is in the middle of an unmodifiable line.
	If the current editing position is at the beginning of an unmodifiable line, a new modifiable line is created before it.
	If the current editing position is at the end of an unmodifiable line, a new modifiable line is created after it.

(continued on next page)

Table 7–10 (Cont.) Selected Built-in Actions When ERASE_UNMODIFIABLE is Turned Off

Built-in	Action
	If the current editing position is on an empty unmodifiable line, then a new modifiable line is created after it.
TRANSLATE	Signals a warning if any of the lines in the range or buffer are unmodifiable.

SET (ERASE_UNMODIFIABLE) optionally returns an integer (0 or 1) indicating whether ERASE_UNMODIFIABLE was turned on before the current call was executed. This makes it easier to return to the previous setting later in the program.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	You specified too many parameters.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	WARNING	The third parameter must be ON or OFF.

EXAMPLE

```
old_setting := SET (ERASE_UNMODIFIABLE, CURRENT_BUFFER, OFF);
```

This statement turns off erasing of unmodifiable records in the current buffer and returns the previous setting of ERASE_UNMODIFIABLE.

SET (FACILITY_NAME)

FORMAT

SET *(FACILITY_NAME, string)*

PARAMETERS

FACILITY_NAME

The facility name that is the first item in a message generated by VAXTPU.

string

The string that you specify as the facility name for messages. The maximum length of this name is 10 characters.

SIGNALED ERRORS

TPU$_FACTOOLONG	WARNING	Name specified is longer than maximum allowed.
TPU$_MINVALUE	WARNING	Argument specified is less than the minimum allowed.
TPU$_ARGMISMATCH	ERROR	The second parameter must be a string.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLE

```
SET (FACILITY_NAME, "new_editor")
```

This statement causes "new_editor" to be used as the facility name in messages.

SET (FORWARD)

FORMAT

SET *(FORWARD, buffer)*

PARAMETERS

FORWARD

A keyword specifying the direction of the buffer. FORWARD means to go toward the end of the buffer.

The default direction for a buffer is forward.

buffer

The buffer whose direction you want to set.

DESCRIPTION

The editor uses this feature to keep track of direction for searching or movement.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (FORWARD) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (FORWARD, my_buffer)
```

This statement causes the direction of the buffer to be toward the end of the buffer.

SET (GLOBAL_SELECT)

Requests ownership of the specified global selection property.

FORMAT

[integer :=] SET *(GLOBAL_SELECT, SCREEN,* { *PRIMARY* | *SECONDARY* | *selection_name* } *)*

PARAMETERS

GLOBAL_SELECT

A keyword indicating that the subject of the information request is a global selection.

SCREEN

A keyword used to preserve compatibility with future versions of VAXTPU.

PRIMARY

A keyword directing VAXTPU to request ownership of the primary global selection.

SECONDARY

A keyword directing VAXTPU to request ownership of the secondary global selection.

selection_name

A string naming the global selection whose ownership VAXTPU is to request.

return value

The value 1 if the global selection ownership request was granted; 0 otherwise.

DESCRIPTION

SET (GLOBAL_SELECT) returns the integer 1 if the request for ownership of a global selection was granted; otherwise 0.

The last parameter identifies the global selection of which VAXTPU is to grab ownership.

VAXTPU is notified immediately if its request is granted. Therefore, VAXTPU does not automatically execute the global selection grab routine when it encounters SET (GLOBAL_SELECT). VAXTPU executes the routine only when it automatically grabs the primary selection after it receives input focus.

For more information about the concept of global selection, see the *XUI Style Guide*.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_REQUIRESDECW	ERROR	You can use the SET (GLOBAL_SELECT) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (GLOBAL_SELECT) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (GLOBAL_SELECT) built-in.

EXAMPLE

```
SET (GLOBAL_SELECT, SCREEN, PRIMARY);
```

This statement requests ownership of the primary global selection. For another example of code using the SET (GLOBAL_SELECT) built-in, see Example B–10.

SET (GLOBAL_SELECT_GRAB)

Specifies the program or learn sequence VAXTPU should execute whenever it automatically grabs ownership of the primary selection.

FORMAT

SET *(GLOBAL_SELECT_GRAB, SCREEN*

[, { *buffer* | *learn_sequence* | *program* | *range* | *string* | *NONE* }])

PARAMETERS

GLOBAL_SELECT_GRAB

A keyword indicating that the subject of the information request is a global select grab routine.

SCREEN

A keyword used to preserve compatibility with future versions of VAXTPU.

buffer

The buffer that contains the grab routine.

learn_sequence

The learn sequence specifying the grab routine.

program

The program specifying the grab routine.

range

The range that contains the grab routine.

string

The string that contains the grab routine.

NONE

A keyword directing VAXTPU to delete the current global selection grab routine. This is the default if you do not specify the optional third parameter.

DESCRIPTION

For more information about VAXTPU global selection support, see Chapter 4.

If the optional parameter is not specified, NONE is the default. When NONE is specified or used by default, VAXTPU deletes the current global selection grab routine. When no global selection grab routine is defined, your application is not informed when VAXTPU grabs the primary global selection.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (GLOBAL_SELECT_GRAB) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (GLOBAL_SELECT_GRAB) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (GLOBAL_SELECT_GRAB) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (GLOBAL_SELECT_GRAB) built-in.

EXAMPLE

```
SET (GLOBAL_SELECT_GRAB, SCREEN, "user_grab_global");
```

This statement designates the procedure *user_grab_global* as a global selection read routine.

For another example of code using the SET (GLOBAL_SELECT_GRAB) built-in, see Example 7–1.

Sample Code Setting Various Global Selection and Input Focus Routines

Example 7–1 shows possible ways that a layered application can use statements setting global selection and input focus routines. The example contains portions of the procedure *eve$mouse_module_init*. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

The statements in Example 7–1 designate EVE's global selection read routine, global selection grab routine, global selection ungrab routine, input focus grab routine, and input focus ungrab routine.

Example 7–1 Initialization Procedure Using Variants of the SET Built-In

```
PROCEDURE eve$mouse_module_init

!  .
!  .
!  .

IF GET_INFO (SCREEN, "decwindows")
THEN

    SET (GLOBAL_SELECT_READ, SCREEN, "eve$write_global_select");
    SET (GLOBAL_SELECT_UNGRAB, SCREEN, "eve$global_select_ungrab");
    SET (GLOBAL_SELECT_GRAB, SCREEN, "eve$global_select_grab");
    SET (INPUT_FOCUS_GRAB, SCREEN, "eve$input_focus_grab");
    SET (INPUT_FOCUS_UNGRAB, SCREEN, "eve$input_focus_ungrab");
ENDIF;

ENDPROCEDURE;
```

SET (GLOBAL_SELECT_READ)

Specifies the program or learn sequence VAXTPU should execute whenever it receives a selection request event on a global selection it owns.

FORMAT

SET *(GLOBAL_SELECT_READ,* { *buffer1* | *SCREEN* }

[, { *buffer2* | *learn_sequence* | *program* | *range* | *string* | *NONE* }])

PARAMETERS

GLOBAL_SELECT_READ

A keyword indicating that the subject of the information request is a global select read routine.

buffer1

The buffer with which the global selection read routine is to be associated.

SCREEN

A keyword indicating that the specified routine is to be the application's default global selection read routine.

buffer2

The buffer that contains the global selection read routine.

learn_sequence

The learn sequence that specifies the global selection read routine.

program

The program that specifies the global selection read routine.

range

The range that contains the global selection read routine.

string

The string that contains the global selection read routine.

NONE

A keyword indicating that the global selection read routine should be deleted.

If you do not specify the optional third parameter, NONE is the default.

DESCRIPTION

To specify a buffer-specific global selection read routine, use the *buffer1* parameter. To specify a global selection read routine for the entire application, use the SCREEN keyword.

When VAXTPU receives a request for information about a global selection it owns, it checks to see if the current buffer has a global selection read routine. If so, it executes that routine. If not, it checks to see if there is an application-wide global selection read routine. If so, it executes that routine. If not, it tries to respond to the request itself.

If the optional parameter is not specified, NONE is the default. When NONE is specified or used by default, VAXTPU deletes the current global selection read routine.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (GLOBAL_SELECT_READ) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (GLOBAL_SELECT_READ) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (GLOBAL_SELECT_READ) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (GLOBAL_SELECT_READ) built-in.

EXAMPLE

```
SET (GLOBAL_SELECT_READ, SCREEN, "user_read_global");
```

The following statement designates the procedure *user_read_global* as a global selection read routine. For another example of code using the SET (GLOBAL_SELECT_READ) built-in, see Example 7–1.

SET (GLOBAL_SELECT_TIME)

Specifies how long VAXTPU should wait before it assumes that a request for information about a global selection will not be satisfied.

FORMAT

SET *(GLOBAL_SELECT_TIME, SCREEN,* { *integer* / *string* } *)*

PARAMETERS

GLOBAL_SELECT_TIME

A keyword directing VAXTPU to set the expiration time for a global selection information request.

SCREEN

A keyword used to maintain compatibility with future versions of VAXTPU.

integer

The number of seconds that VAXTPU should wait.

string

A string in VMS delta time format indicating how long VAXTPU should wait.

DESCRIPTION

The default waiting time is set by DECwindows. The maximum waiting time you can set is 24 days, 20 hours.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVTIME	WARNING	You specified an invalid time interval.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The SET (GLOBAL_SELECT_TIME) built-in cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (GLOBAL_SELECT_TIME) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (GLOBAL_SELECT_TIME) built-in.

TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (GLOBAL_SELECT_TIME) built-in.

EXAMPLE

```
SET (GLOBAL_SELECT_TIME, SCREEN, 3);
```

This statement sets the waiting time for a global selection response to 3 seconds.

SET (GLOBAL_SELECT_UNGRAB)

Specifies the program or learn sequence VAXTPU should execute whenever it loses ownership of a selection.

FORMAT

SET *(GLOBAL_SELECT_UNGRAB, SCREEN*
[, { *buffer* / *learn_sequence* / *program* / *range* / *string* / *NONE* } *])*

PARAMETERS

GLOBAL_SELECT_UNGRAB
A keyword indicating that the subject of the information request is a global select ungrab routine.

SCREEN
A keyword used to preserve compatibility with future versions of VAXTPU.

buffer
The buffer that contains the global selection ungrab routine.

learn_sequence
The learn sequence that specifies the global selection ungrab routine.

program
The program that specifies the global selection ungrab routine.

range
The range that contains the global selection ungrab routine.

string
The string that contains the global selection ungrab routine.

NONE
A keyword directing VAXTPU to delete the current global selection ungrab routine. This is the default if you do not specify the optional third parameter.

DESCRIPTION

For more information about VAXTPU global selection support, see Chapter 4.

If the optional parameter is not specified, NONE is the default. When NONE is specified or used by default, VAXTPU deletes the current global selection ungrab routine. When no global selection ungrab routine is defined, your application is not informed when VAXTPU loses ownership of the primary global selection.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (GLOBAL_SELECT_UNGRAB) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (GLOBAL_SELECT_UNGRAB) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (GLOBAL_SELECT_UNGRAB) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (GLOBAL_SELECT_UNGRAB) built-in.

EXAMPLE

```
SET (GLOBAL_SELECT_UNGRAB, SCREEN, "user_ungrab_global");
```

This statement designates the procedure *user_ungrab_global* as a global selection ungrab routine. For another example of code using the SET (GLOBAL_SELECT_UNGRAB) built-in, see Example 7–1, following the description of the SET (GLOBAL_SELECT_GRAB) built-in.

SET (HEIGHT)

Sets the height of the VAXTPU screen without modifying the height or location of any VAXTPU window.

FORMAT

SET *(HEIGHT, SCREEN, length)*

PARAMETERS

HEIGHT

A keyword indicating that the vertical dimension is being set.

SCREEN

A keyword indicating that the screen is being resized.

length

The length (in lines) that you want the screen to have. The value must be an integer between 1 and 255.

DESCRIPTION

Although SET (HEIGHT) does not alter any VAXTPU windows, the default EVE behavior when the screen is made smaller is to unmap windows from the screen, starting with the bottom-most window and working upward, until there is room on the screen for the remaining windows. If the screen is subsequently made larger, the unmapped windows are not remapped by default.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_TOOFEW	ERROR	SET (HEIGHT) requires three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	WARNING	The second parameter must be SCREEN.
TPU$_ BADLENGTHCHANGE	WARNING	The terminal's characteristics will not allow the height of the screen to change.
TPU$_BADVALUE	ERROR	The terminal cannot be set to the requested height.

EXAMPLE

```
SET (HEIGHT, SCREEN, 20);
```

This statement causes the screen to have a height of 20 lines.

SET (ICON_NAME)

Designates the string used as the layered application's name in the DECwindows icon box.

FORMAT

SET *(ICON_NAME, string)*

PARAMETERS

ICON_NAME

A keyword instructing VAXTPU to set the text of an icon.

string

The text you want to appear in the icon.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (ICON_NAME) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (ICON_NAME) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (ICON_NAME) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (ICON_NAME) built-in.

EXAMPLE

```
SET (ICON_NAME, "WordMonger");
```

This statement sets the text naming the layered application to be the string *WordMonger*.

SET (ICON_PIXMAP)

Determines the pixmap the application uses to create its icon in the DECwindows icon box if the user has selected the Large Window Manager Icon Style.

FORMATS

SET *(ICON_PIXMAP,integer,icon_pixmap [,widget])*

or

SET *(ICON_PIXMAP,bitmap_file_name [,widget])*

PARAMETERS

ICON_PIXMAP

A keyword indicating that the SET built-in is being used to determine the pixmap the application uses to create its icon in the DECwindows icon box if the user has selected the Large Window Manager Icon Style.

integer

The hierarchy identifier returned by the SET (DRM_HIERARCHY) built-in. This identifier is passed to the XUI Resource Manager, which uses the identifier to find the hierarchy's resource name in the resource database.

icon_pixmap

A case-sensitive string that is the name assigned to the icon in the UIL file defining the icon pixmap. The icon must be declared EXPORTED in the UIL file.

widget

The widget whose icon pixmap is to be set. By default, VAXTPU sets the icon pixmap of its top-level widget.

bitmap_file_name

The file specification of a bitmap file. SET (ICON_PIXMAP) requires these files to be in the format created by the Xlib routine WRITE BITMAP FILE. To create a file with the correct format, you can use the program SYS$SYSTEM:DECW$PAINT.EXE (the DECpaint application) or the program DECW$EXAMPLES:BITMAP.EXE. If you use DECpaint, use the Customize Picture Size option to set the picture size to nonstandard, the width to 32 pixels, and the height to 32 pixels. Use the Zoom option to manipulate this small image. Choose the X11 format when you save the file.

DESCRIPTION

To specify an icon pixmap defined in a UIL file, use the first format variant shown in the Format section. To specify an icon created in a bitmap file, use the second format variant shown in the Format section.

If an application uses SET (ICON_PIXMAP) so a large icon can be displayed, in most cases the application should also use SET (ICONIFY_PIXMAP) to create an iconify button in the title bar. An application also needs to use SET (ICONIFY_PIXMAP) so a small icon can be displayed if the user selects the Small Window Manager Icon Style from the Session Manager's Customize Window dialog box.

SIGNALED ERRORS

TPU$_FAILURE_STATUS	ERROR	The Digital Resource Manager returned an error status.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	Built-in does not return a value.
TPU$_REQUIRESDECW	ERROR	Requires the VAXTPU DECwindows screen updater.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_TOOMANY	ERROR	You specified too many parameters.

EXAMPLE

```
SET (ICON_PIXMAP, "DISK1:[SMITH]ICON_FLAMINGO.X11")
```

This statement causes the icon pixmap stored in the file ICON_FLAMINGO.X11 to be displayed in the application's icon if the Large Window Manager Icon Style has been selected.

SET (ICONIFY_PIXMAP)

Determines the pixmap the application uses to create its icon in the DECwindows icon box if the user has selected the Small Window Manager Icon Style. When you use SET (ICONIFY_PIXMAP), VAXTPU also automatically places the specified pixmap in the application's iconify button, in the title bar.

FORMATS

SET *(ICONIFY_PIXMAP,integer,icon_pixmap [,widget])*

or

SET *(ICONIFY_PIXMAP,bitmap_file_name [,widget])*

PARAMETERS

ICONIFY_PIXMAP

A keyword indicating that the SET built-in is being used to determine the pixmap the application uses to create its icon in the DECwindows icon box if the user has selected the Small Window Manager Icon Style.

integer

The hierarchy identifier returned by the SET (DRM_HIERARCHY) built-in. This identifier is passed to the XUI Resource Manager, which uses the identifier to find the hierarchy's resource name in the resource database.

icon_pixmap

A case-sensitive string that is the name assigned to the icon in the UIL file defining the iconify pixmap. The icon must be declared EXPORTED in the UIL file.

widget

The widget whose iconify pixmap is to be set. By default, VAXTPU sets the iconify pixmap of its top-level widget.

bitmap_file_name

The file specification of a bitmap file. SET (ICONIFY_PIXMAP) requires these files to be in the format created by the Xlib routine WRITE BITMAP FILE. To create a file with the correct format, you can use the program SYS$SYSTEM:DECW$PAINT.EXE (the DECpaint application) or the program DECW$EXAMPLES:BITMAP.EXE. If you use DECpaint, use the Customize Picture Size option to set the picture size to non-standard, the width to 16 pixels, and the height to 16 pixels. Use the Zoom option to manipulate this small image. Choose the X11 format when you save the file.

DESCRIPTION

To specify an iconify pixmap defined in a UIL file, use the first format variant shown in the Format section. To specify an icon created in a bitmap file, use the second format variant shown in the Format section.

If an application uses SET (ICONIFY_PIXMAP) so a small icon can be displayed, in most cases the application should also use SET (ICON_PIXMAP) so a large icon can be displayed if the user selects the Large Window Manager Icon Style.

Note that the user selects the Large or Small Window Manager Icon Style using the Session Manager's Customize Window dialog box.

SIGNALED ERRORS

TPU$_FAILURE_STATUS	ERROR	The Digital Resource Manager returned an error status.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	Built-in does not return a value.
TPU$_REQUIRESDECW	ERROR	Requires the VAXTPU DECwindows screen updater.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_TOOMANY	ERROR	You specified too many parameters.

EXAMPLE

```
SET (ICONIFY_PIXMAP, "DISK1:[SMITH]ICONIFY_FLAMINGO.X11")
```

This statement causes the iconify pixmap stored in the file ICONIFY_FLAMINGO.X11 to be displayed in the application's iconify button and in the application's icon if the small Window Manager Icon Style has been selected.

SET (INFORMATIONAL)

FORMAT

SET *(INFORMATIONAL,* { *ON* / *OFF* } *)*

PARAMETERS

INFORMATIONAL

Informational messages that VAXTPU writes.

ON

Causes the informational messages to be displayed.

OFF

Suppresses the display of informational messages.

DESCRIPTION

If you specify a section file when invoking VAXTPU (either by default, or by using the qualifier /SECTION), VAXTPU may not display informational messages. You can cause informational messages to be written by using SET (INFORMATIONAL, ON).

If you use the qualifier /NOSECTION when invoking VAXTPU, informational messages are written by default.

When you are developing VAXTPU programs, the informational messages help you find errors in your program, so it is a good idea to use the built-in procedure SET (INFORMATIONAL) to cause the messages to be displayed.

See Appendix D for a list of the VAXTPU informational messages.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (INFORMATIONAL) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (INFORMATIONAL, OFF)
```

This statement causes the display of informational messages to be turned off.

SET (INPUT_FOCUS)

Requests ownership of the input focus. Ownership of the input focus determines which application or widget processes user input from the keyboard.

FORMAT

SET *(INPUT_FOCUS* [[*, SCREEN* | *, widget*]] *)*

PARAMETERS

INPUT_FOCUS

A keyword directing VAXTPU to assign the input focus.

SCREEN

An optional keyword indicating that the top-level widget associated with VAXTPU's screen is to receive the input focus. This keyword is the default.

widget

The widget that is to receive the input focus. Note that if you specify a widget for this parameter, the VAXTPU key bindings are not available to process keyboard input into the specified widget.

DESCRIPTION

This built-in requests that input focus be given to VAXTPU or to a widget that is part of an application layered on VAXTPU. It does not guarantee that VAXTPU or the widget gets the input focus. If VAXTPU or the widget receives the input focus, it gets a focus-in event. When VAXTPU gets this event, it calls the input focus grab routine. For more information about the role of events in DECwindows applications, see the *VMS DECwindows Guide to Application Programming*.

When the top-level widget for VAXTPU's screen has the input focus, VAXTPU processes keystrokes normally. That is, undefined printable keys insert characters in the current buffer, and defined keys execute the code bound to them.

If a child widget in the widget hierarchy has the input focus, keystrokes are processed by that widget. For example, when a text widget in EVE's replace dialog box has the input focus, keystrokes are processed by the text widget, not by VAXTPU. No VAXTPU key bindings are in effect.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.

TPU$_NORETURNVALUE	ERROR	SET (INPUT_FOCUS) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (INPUT_FOCUS) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (INPUT_FOCUS) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (INPUT_FOCUS) built-in.

EXAMPLE

```
PROCEDURE eve$$widget_replace_ok

LOCAL   new_string,
        old_string,
        old_str_text_widget,
        new_str_text_widget;


SET (INPUT_FOCUS);      ! This statement grabs input focus
                        ! so CTRL/C events will be detected.

! Get the replace strings from the eve$$k_replace_new_[old]text widgets.

old_str_text_widget :=  GET_INFO (WIDGET, "widget_id", eve$x_replace_dialog,
                                  "REPLACE_DIALOG.REPLACE_OLD_TEXT")

old_string := GET_INFO (old_str_text_widget, "text");

! Test only the old string.
IF old_string = ""
THEN
    eve$message (EVE$_NOREPLSTR);
    RETURN;
ENDIF;

new_str_text_widget :=  GET_INFO (WIDGET, "widget_id", eve$x_replace_dialog,
                                  "REPLACE_DIALOG.REPLACE_NEW_TEXT")

new_string := GET_INFO (new_str_text_widget, "text");

IF new_string = ""
THEN
    eve$$replace1 (old_string, new_string, 1);
ELSE
    eve$$replace1 (old_string, new_string);
ENDIF;

ENDPROCEDURE;
```

This procedure shows one possible way that a layered application can use the SET (INPUT_FOCUS) built-in. The procedure is a modified version of the EVE procedure EVE$$WIDGET_REPLACE_OKAY. You can find the original version in SYS$EXAMPLES:EVE$MENUS.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

Procedure EVE$$WIDGET_REPLACE_OK fetches and tests the user's responses to prompts for old and new replace strings.

SET (INPUT_FOCUS_GRAB)

Specifies the program or learn sequence that VAXTPU should execute whenever it processes a focus-in event.

FORMAT

SET *(INPUT_FOCUS_GRAB , SCREEN* [, { *buffer* | *learn_sequence* | *program* | *range* | *string* | *NONE* }])

PARAMETERS

INPUT_FOCUS_GRAB

A keyword directing VAXTPU to set an attribute related to an input focus grab routine.

SCREEN

An keyword used for compatibility with future versions of VAXTPU.

buffer

The buffer that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

learn_sequence

The learn sequence that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

program

The program that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

range

The range that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

string

The string that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

NONE

A keyword directing VAXTPU to delete the input focus grab routine. If you specify this keyword or do not specify the parameter at all, the application is not notified when input focus is received.

DESCRIPTION

For more information about VAXTPU input focus support, see Chapter 4.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (INPUT_FOCUS_GRAB) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (INPUT_FOCUS_GRAB) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (INPUT_FOCUS_GRAB) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (INPUT_FOCUS_GRAB) built-in.

EXAMPLE

```
SET (INPUT_FOCUS_GRAB, SCREEN, "user_grab_focus");
```

This statement designates the procedure *user_grab_focus* as an input focus grab routine. For another example of code using the SET (INPUT_FOCUS_GRAB) built-in, see Example 7–1.

SET (INPUT_FOCUS_UNGRAB)

Specifies the program or learn sequence that VAXTPU should execute whenever it processes a focus-out event.

FORMAT

SET *(INPUT_FOCUS_UNGRAB, SCREEN* [[, { *buffer* | *learn_sequence* | *program* | *range* | *string* | *NONE* }]] *)*

PARAMETERS

INPUT_FOCUS_UNGRAB

A keyword directing VAXTPU to set an attribute related to an input focus ungrab routine.

SCREEN

A keyword used for compatibility with future versions of VAXTPU.

buffer

The buffer that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

learn_sequence

The learn sequence that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

program

The program that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

range

The range that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

string

The string that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

NONE

A keyword directing VAXTPU to delete the input focus ungrab routine. If you specify this keyword or do not specify the parameter at all, the application is not notified when input focus is lost.

DESCRIPTION

For more information about VAXTPU input focus support, see Chapter 4.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (INPUT_FOCUS_UNGRAB) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (INPUT_FOCUS_UNGRAB) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (INPUT_FOCUS_UNGRAB) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (INPUT_FOCUS_UNGRAB) built-in.

EXAMPLE

```
SET (INPUT_FOCUS_UNGRAB, SCREEN, "user_ungrab_focus");
```

This statement designates the procedure *user_ungrab_focus* as an input focus ungrab routine. For another example of code using the SET (INPUT_FOCUS_UNGRAB) built-in, see Example 7–1.

SET (INSERT)

FORMAT

SET *(INSERT, buffer)*

PARAMETERS

INSERT

A keyword specifying the mode of entering text. INSERT means that characters are added to the buffer immediately before the editing point. See also the description of the built-in procedure SET (OVERSTRIKE).

The default mode for text entry is insert.

buffer

The buffer whose mode of text entry you want to set.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (INSERT) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLE

```
SET (INSERT, my_buffer)
```

This statement causes the characters that you add to the buffer to be added immediately before the editing point.

SET (JOURNALING)

Turns buffer change journaling on or off, sets the journaling frequency, and specifies a journal file name.

FORMATS

SET *(JOURNALING, buffer* { *ON* / *OFF* }*, integer [,file_name_string])*

or

SET *(JOURNALING, integer)*

PARAMETERS

JOURNALING
A keyword indicating that the SET built-in is being used to enable or disable buffer change journaling, or set the frequency of journaling.

buffer
The buffer for which you want to enable or disable buffer change journaling.

ON
A keyword turning on buffer change journaling.

OFF
A keyword turning off buffer change journaling.

file_name_string
The string naming the file you want to use as the buffer's journal file. If the file does not exist, VAXTPU automatically creates it. You cannot specify this parameter if you have specified the keyword OFF for the third parameter. If you do not specify any file name when you turn journaling on, VAXTPU creates a journal file for you and names the file using the default naming algorithm. For more information on this algorithm, see Section 1.7.1.

integer
The integer that you specify that determines how frequently records are written to the journal file. The value of this integer must be between 1 and 10.

DESCRIPTION

Journaling can be turned on only if the buffer is safe for journaling. For a buffer to be safe for journaling, it must either be empty, have never been modified, or be unmodified since the last time it was written to a file. (Whether the buffer has been modified is not the same as whether the buffer is marked as modified. The *modified* flag can be set or cleared by the application or by the user.)

Once a buffer that is being journaled is written to a file, the journal file is closed and deleted, and a new journal file is started that references the newly created file. Similarly, reading a file into an empty buffer does not copy the file into the journal—it simply inserts a *reference* to the file in the journal. Note that this behavior must be taken into account when you perform operations that use temporary files. For example, writing a buffer to a temporary file (which is modified by an external program), then erasing the buffer and re-reading a (modified) temporary file will point the journal file at the temporary file. If you then delete the temporary file, the buffer will be unrecoverable.

A journal file name can be supplied only if journaling is being turned on. If a journal file name is supplied, VAXTPU creates a journal file using the name you specified. If this parameter is omitted, VAXTPU creates a journal file name based on the buffer's name using the algorithm outlined in Section 1.7.1.

If journaling is being turned off for the specified buffer, VAXTPU closes the journal file but does not delete it.

VAXTPU signals a warning or error if any of the following conditions apply:

- Journaling is being turned on and one or more of the following is also true:
  - The specified buffer is not safe for journaling.
  - The specified buffer is already being journaled.
  - An RMS error was returned when VAXTPU tried to create the journal file.
- Journaling is being turned off and a journal file name is specified in the built-in call.

Caution: Journal files contain a record of *all* information being edited. Therefore, when editing files containing secure or confidential data, be sure to keep the journal files secure as well.

The *integer* parameter specifies the journaling frequency. VAXTPU provides a 500-byte buffer for journaling keystrokes. If journaling is enabled, a write to the journal file occurs when the buffer is full. This built-in procedure lets you determine the frequency with which records are written to the journal file; the lower the integer you specify, the more often journal records are written to disk.

A value of 1 causes a record to be written for approximately every 10 keys pressed. A value of 10 causes a record to be written for approximately every 125 keys. If you are entering only text (rather than procedures that are bound to keys), the number of keystrokes included in a record is greater—for a value of 1, a record is written for approximately every 30 to 35 keystrokes; for a value of 10, a record is written for approximately every 400 keystrokes.

SIGNALED ERRORS

TPU$_MINVALUE	WARNING	Argument is less than minimum allowed.
TPU$_MAXVALUE	WARNING	Argument is greater than maximum allowed.
TPU$_TOOMANY	ERROR	You specified too many parameters.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_INVPARAM	ERROR	You specified a parameter with the wrong data type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.
TPU$_JRNLOPEN	ERROR	A journal file for this buffer is already open.

EXAMPLES

1
```
SET (JOURNALING, CURRENT_BUFFER, ON, "disk1:[jones]journal.jnl");
```

This statement turns on buffer change journaling for the current buffer and directs VAXTPU to use the file JOURNAL.JNL in the directory [JONES] as the journal file.

2
```
SET (JOURNALING, 1)
```

This statement causes a record to be written from the buffer to a journal file at intervals of approximately 10 user keystrokes. If all or most of the keys pressed have procedures bound to them, VAXTPU may write out the contents of the buffer after fewer than 10 keystrokes. The journaling interval shown in this statement is the shortest that you can specify.

SET (KEYSTROKE_RECOVERY)

Turns keystroke journal recovery on or off.

FORMAT

SET *(KEYSTROKE_RECOVERY* { *ON* / *OFF* } *)*

PARAMETERS

KEYSTROKE_RECOVERY

A keyword directing VAXTPU to enable or disable keystroke recovery, depending on whether /RECOVER or /NORECOVER was specified on the command line.

ON

A keyword enabling keystroke recovery. (This has the same effect as specifying the /RECOVER qualifier.)

OFF

A keyword disabling keystroke recovery. (This has the same effect as specifying the /NORECOVER qualifier.)

DESCRIPTION

If the /RECOVER qualifier is specified on the DCL command line, VAXTPU checks whether the application calls the JOURNAL_OPEN built-in to open a keystroke journal file. If the application does not call the JOURNAL_OPEN built-in, by default VAXTPU signals an error when the application starts accepting keyboard input.

In some circumstances, you may want your application to accept the /RECOVER qualifier without error, even though the application does not call the JOURNAL_OPEN built-in. For example, if your application uses only buffer-change journaling, the /RECOVER qualifier can be used when VAXTPU is invoked, but the JOURNAL_OPEN built-in is not used.

To disable the error caused by the lack of a call to JOURNAL-OPEN and concurrently to prevent VAXTPU from performing keystroke recovery (even if /RECOVER is specified and you use JOURNAL_OPEN) use SET (KEYSTROKE_RECOVERY, OFF). Conversely, to direct VAXTPU to perform keystroke recovery (even if /NORECOVER is specified and you use JOURNAL_OPEN), use SET (KEYSTROKE_RECOVERY, ON).

Note that SET (KEYSTROKE_RECOVERY) signals an error if the application code or the user calls the built-in after VAXTPU has started accepting keyboard input.

To determine whether a recovery using a keystroke journal file is currently in progress, use GET_INFO (SYSTEM, "recover"). This GET_INFO call returns FALSE (0) if no keystroke recovery is currently in progress, and returns TRUE (1) if a keystroke recovery is currently in progress. Note that SET (KEYSTROKE_RECOVERY) can determine the value returned by GET_INFO (SYSTEM, "recover") but cannot affect the value returned by GET_INFO (COMMAND_LINE, "recover"). GET_INFO (COMMAND_

LINE, "recover") returns a value indicating whether /RECOVER was specified when VAXTPU was invoked.

SIGNALED ERRORS

TPU$_JNLNOTOPEN	ERROR	No keystroke journal file is open from which to recover.
TPU$_RECJNLOPEN	ERROR	A keystroke journal file is already open.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_TOOMANY	ERROR	You specified too many parameters.
TPU$_INVPARAM	ERROR	You specified a parameter with the wrong data type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (KEYSTROKE_RECOVERY, ON)
```

This statement directs VAXTPU to do keystroke journal recovery even if /NORECOVER was specified on the command line.

SET (KEY_MAP_LIST)

FORMAT

SET *(KEY_MAP_LIST, string* { *[, buffer]* / *, window* } *)*

PARAMETERS

KEY_MAP_LIST

The key map list that you bind to a buffer or window.

The default key map list is TPU$KEY_MAP_LIST.

string

A quoted string, or a variable name representing a string constant, that specifies the key map list that you bind to a buffer or window.

buffer

Buffer to which you bind the specified key map list. The default is the buffer to which you are positioned.

window

The window with which you want to associate the key map list.

The key map list manipulated by SET (KEY_MAP_LIST) is used only to process mouse events in the specified window. Keystrokes are processed using the key map list associated with the buffer.

DESCRIPTION

The SET (KEY_MAP_LIST) built-in procedure binds a specified key map list to a buffer or window. If the buffer or window is not specified, the default is to bind the key map list to the current buffer. A buffer or window can be associated with only one key map list at a time. A key map list can be associated with many buffers or windows simultaneously.

SIGNALED ERRORS

TPU$_NOKEYMAPLIST	WARNING	Attempt to access an undefined key map list.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (KEY_MAP_LIST) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (KEY_MAP_LIST) built-in.
TPU$_NOCURRENTBUF	ERROR	You are not positioned in a buffer.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the SET (KEY_MAP_LIST) built-in.

EXAMPLE

```
SET (KEY_MAP_LIST, "tpu$_key_map_list")
```

This statement binds the key map list called TPU$_KEY_MAP_LIST to the current buffer.

```
PROCEDURE user_scratch_window

LOCAL scratch_window,
      scratch_buffer,
      scratch_map,
      scratch_list;

scratch_window := CREATE_WINDOW (20, 3, ON);
scratch_buffer := CREATE_BUFFER ("test", "junk.txt");
scratch_map := CREATE_KEY_MAP ("user_scratch_map");
DEFINE_KEY (eve$$kt_return + "sample_M1_DRAG", M1DRAG, "mouse_button_1",
            "user_scratch_map");
scratch_list := CREATE_KEY_MAP_LIST ("user_scratch_list", "user_scratch_map",
                                     eve$x_mouse_keys);
SET (KEY_MAP_LIST, "user_scratch_list", scratch_window);
MAP (scratch_window, scratch_buffer);

ENDPROCEDURE;
```

This procedure creates a small "scratch pad" window and maps it to a scratch buffer called *junk1.txt*. The procedure defines a key map list consisting of a user-defined key map redefining M1DRAG plus the standard EVE mouse key map. By setting the scratch window's key map list to be *user_scratch_list*, the procedure invokes *sample_m1_drag* when the user drags the mouse in the scratch window.

SET (LEFT_MARGIN)

FORMAT

SET *(LEFT_MARGIN, buffer, integer)*

PARAMETERS

LEFT_MARGIN

The left margin of a buffer.

buffer

The buffer in which the left margin is being set.

integer

The column at which the left margin is set.

DESCRIPTION

The SET (LEFT_MARGIN) built-in procedure allows you to change only the left margin of a buffer.

Newly created buffers receive a left margin of 1 (that is, the margin is set in column 1) if a template buffer is not specified in the call to the CREATE_BUFFER built-in procedure. If a template buffer is used, that buffer sets the left margin for all newly created buffers.

Use SET (LEFT_MARGIN) to override the default left margin.

The buffer margin settings are independent of the terminal width or window width settings.

The built-in procedure FILL uses these margin settings when it fills the text of a buffer.

When VAXTPU creates a new line, the line obtains its left margin value from the left margin of the current buffer setting. However, changing the left margin setting for the buffer does not change the left margin for any existing lines.

The value of the left margin must be at least 1 and less than the right margin value.

If you want to use the margin settings of an existing buffer in a user-written procedure, GET_INFO (buffer, "left_margin") and GET_INFO (buffer, "right_margin") return the values of the margin settings in the specified buffer.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (LEFT_MARGIN) built-in requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.

TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADMARGINS	WARNING	The left margin setting must be less than the right; both must be greater than zero.

EXAMPLES

1
```
SET (LEFT_MARGIN, my_buffer, 1)
```

This statement causes the left margin of the buffer represented by the variable *my_buffer* to be changed. The left margin of the buffer is set to 1. The right margin is unchanged.

2
```
SET (LEFT_MARGIN, CURRENT_BUFFER, 10)
```

This statement causes the left margin of the current buffer to be changed to 10. As above, the right margin is unchanged.

SET (LEFT_MARGIN_ACTION)

FORMAT

SET *(LEFT_MARGIN_ACTION, buffer1* [[{ , *buffer2* | , *learn_sequence* | , *program* | , *range* | , *string* }]] *)*

PARAMETERS

LEFT_MARGIN_ACTION

Refers to the action taken when the user presses a self-inserting key while the cursor is to the left of a line's left margin. A self-inserting key is one that is associated with a printable character.

buffer1

The buffer in which the left margin action routine is being set.

buffer2

A buffer containing the VAXTPU statements to be executed when the user presses a self-inserting key while the cursor is to the left of a buffer's left margin.

learn_sequence

A learn sequence that is to be replayed when the user presses a self-inserting key while the cursor is to the left of a buffer's left margin.

program

A program that is to be executed when the user presses a self-inserting key while the cursor is to the left of a buffer's left margin.

range

A range that contains VAXTPU statements that are to be executed when the user presses a self-inserting key while the cursor is to the left of a buffer's left margin.

string

A string that contains VAXTPU statements that are to be executed when the user presses a self-inserting key while the cursor is to the left of a buffer's left margin.

DESCRIPTION

The SET (LEFT_MARGIN_ACTION) built-in procedure allows you to specify an action to be taken when the user attempts to insert text to the left of the left margin of a line. If the third parameter is not specified, the left margin action routine is deleted. If no left margin action routine has been specified, the text is simply inserted at the current position before any necessary padding spaces, and the left margin of the line becomes the current position.

Newly created buffers do not receive a left margin action routine if a template buffer is not specified on the call to the CREATE_BUFFER built-in procedure. If a template buffer is specified, the left margin action routine of the template buffer is used.

The left margin action routine only affects text entered from the keyboard or a learn sequence. Inserting text into a buffer to the left of the left margin using the COPY_TEXT or MOVE_TEXT built-in procedure does not trigger the left margin action routine.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (LEFT_MARGIN_ACTION) built-in requires at least two parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_COMPILEFAIL	ERROR	Compilation aborted because of syntax errors.

EXAMPLES

1
```
SET (LEFT_MARGIN_ACTION, CURRENT_BUFFER, "push_to_left_margin")
```

This statement causes the procedure PUSH_TO_LEFT_MARGIN to be executed when the user attempts to type a character to the left of the left margin of the current line. A typical left margin action routine moves the editing point to the left margin and inserts an appropriate number of spaces starting at the left margin.

2
```
SET (LEFT_MARGIN_ACTION, CURRENT_BUFFER)
```

This statement deletes any left margin action routine that may be defined for the current buffer. When there is no user-defined left margin action routine, if the user types a character to the left of the current line's left margin, the text is inserted with spaces padding the text to the old left margin. The leftmost character on the line establishes the line's new left margin.

SET (LINE_NUMBER)

FORMAT

SET *(LINE_NUMBER,* { *ON* / *OFF* } *)*

PARAMETERS

LINE_NUMBER

Refers to the VAXTPU display of the procedure and line number at which an error occurred.

ON

Turns on display of the line number and procedure at which an error occurred.

OFF

Turns off display of the line number and procedure at which an error occurred.

DESCRIPTION

Line numbers are useful for programmers debugging VAXTPU programs, but they do not have much meaning to users who do not have the source code available to them.

After a compilation, the line numbers displayed for procedures are relative to the beginning of the procedure. For VAXTPU statements compiled outside a procedure, the line numbers displayed are relative to the beginning of the buffer, range, or string being compiled. If there are no procedure declarations before the executable statements, line numbering starts at the beginning of the buffer or range that is being compiled. For strings, the line number is always 1.

Line numbers may be changed when you use the SAVE built-in to write a section file. If you specify the parameter NO_PROCEDURE_NAMES, the line numbers displayed are relative to the beginning of the buffer or range that was compiled, not relative to the beginning of a procedure.

The default setting for LINE_NUMBER depends on whether a section file was loaded by VAXTPU. If a section file was loaded, the default is OFF. If a section file was not loaded, the default is ON.

Note that SET (LINE_NUMBER) is related to SET (TRACEBACK). SET (TRACEBACK, ON) turns on both traceback and line numbers. SET (LINE_NUMBER, OFF) turns off both traceback and line numbers. It is also possible to set traceback off and line numbers on.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (LINE_NUMBER) built-in requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	Only the keywords ON and OFF are allowed.

EXAMPLE

```
PROCEDURE line_number_example
    SET (LINE_NUMBER, ON);
    SET (LINE_NUMBER, BELL);
ENDPROCEDURE;
```

This procedure displays the line number at which the error occurred. Executing this procedure displays the following in the message buffer:

```
BELL is an invalid keyword
At line 4
```

SET (MAPPED_WHEN_MANAGED)

Controls whether a widget instance is mapped to the screen when it is managed.

FORMAT

SET *(MAPPED_WHEN_MANAGED,widget,* { *ON* / *OFF* } *)*

PARAMETERS

MAPPED_WHEN_MANAGED

A keyword indicating that SET is being used to control whether the specified widget instance should become visible when it is managed.

widget

The widget instance whose mapped status you want to set.

ON

A keyword directing VAXTPU to make the specified widget visible when it is managed. This is the default value.

OFF

A keyword directing VAXTPU not to make the specified widget visible when it is managed.

DESCRIPTION

For more information on managing widgets, see the *VMS DECwindows Guide to Application Programming* and the *VMS DECwindows Toolkit Routines Reference Manual*.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	Built-in does not return a value.
TPU$_REQUIRESDECW	ERROR	Requires the VAXTPU DECwindows screen updater.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_TOOMANY	ERROR	You specified too many parameters.

EXAMPLE

```
SET (MAPPED_WHEN_MANAGED, example_widget, OFF);
```

This statement directs VAXTPU to make the widget contained in *example_widget* invisible when the widget is managed.

SET (MARGINS)

FORMAT

SET *(MARGINS, buffer, integer1, integer2)*

PARAMETERS

MARGINS

A keyword indicating that SET is to determine the left and right margins of a buffer.

The default left margin is 1 and the default right margin is 80.

buffer

The buffer in which the margins are being set.

integer1

The column at which the left margin is set.

integer2

The column at which the right margin is set.

DESCRIPTION

The SET (MARGINS) built-in procedure allows you to change the left and right margins of a buffer. The default margins for a buffer are set to 1 for the left margin and 80 for the right margin when you use the CREATE_BUFFER built-in. The built-in procedure FILL uses these margin settings when it fills the text of a buffer.

This built-in procedure controls the buffer margin settings even if the terminal width or window width is set to something else.

The value of the left margin must be at least 1 and less than the right margin value. The value of the right margin must be less than the maximum record size for the buffer. You can use the built-in procedure GET_INFO (buffer, "record_size") to find out the maximum record size of a buffer.

If you want to use the margin settings of an existing buffer in a user-written procedure, the statements GET_INFO (buffer, "left_margin") and GET_INFO (buffer, "right_margin") return the values of the margin settings.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (MARGINS) built-in requires five parameters.
TPU$_TOOMANY	ERROR	You specified more than four parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

TPU$_BADMARGINS	WARNING	Left margin must be smaller than right; both must be greater than zero.

EXAMPLES

1
```
SET (MARGINS, my_buffer, 1, 132)
```

This statement causes the margins of the buffer represented by the variable *my_buffer* to be changed. The left margin of the buffer is set to 1 and the right margin is set to 132.

2
```
SET (MARGINS, CURRENT_BUFFER, 10, 70)
```

This statement causes the margins of the current buffer to be changed to left margin 10 and right margin 70.

SET (MAX_LINES)

FORMAT

SET *(MAX_LINES, buffer, integer)*

PARAMETERS

MAX_LINES

The maximum number of lines a buffer can contain.

buffer

The buffer for which you are setting the maximum number of lines.

integer

The maximum number of lines for the buffer. The valid values are 0, 2, or an integer greater than 2. The maximum value depends on the memory capacity of your system.

The default maximum number of lines is 0 (in other words, this feature is turned off).

DESCRIPTION

If you exceed the maximum number of lines for a buffer, VAXTPU deletes lines from the beginning of the buffer to make room for any lines that exceed the maximum.

Note that SET (MAX_LINES) does not consider the end-of-buffer text to be a record. For example, if you set the maximum number of lines to be 1000, the buffer can contain 1000 records plus the end-of-buffer text.

If you specify a value of 0 for *integer*, this feature is turned off and VAXTPU does not check for the maximum number of lines.

SIGNALED ERRORS

TPU$_MINVALUE	WARNING	Argument less than minimum allowed.
TPU$_MAXVALUE	WARNING	Argument greater than maximum allowed.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_TOOMANY	ERROR	SET (MAX_LINES) accepts only three parameters.
TPU$_TOOFEW	ERROR	SET (MAX_LINES) requires three parameters.

EXAMPLE

```
SET (MAX_LINES, message_buffer, 20)
```

This statement causes the maximum number of lines for the message buffer to be 20. Only the most recent lines of messages are kept.

SET (MENU_POSITION)

Sets menu positioning for one or more pop-up widgets.

FORMAT

{ **array1** / **NONE** } **:= SET** *(MENU_POSITION, mouse_down_button,* { *array2* / *NONE* / *widget* } *)*

PARAMETERS

MENU_POSITION

A keyword indicating that the SET built-in is being used to set the menu position of a pop-up widget or widgets.

mouse_down_button

This keyword (M1DOWN, M2DOWN, M3DOWN, M4DOWN, or M5DOWN) indicates the mouse button associated with the pop-up menus.

array2

An integer-indexed array of pop-up menu widgets to be set for automatic menu positioning.

NONE

This keyword requests that VAXTPU stop automatic positioning of pop-up menu widgets for the specified mouse button.

widget

The pop-up menu widget to be set for automatic menu positioning.

return values

array1

An integer-indexed array of all pop-up menu widgets that were set for automatic positioning for the specified mouse button prior to this built-in call.

NONE

A keyword indicating that no pop-up menu widgets were set for the specified mouse button prior to this built-in call.

DESCRIPTION

When the specified mouse button is pressed and the application manages a pop-up widget, VAXTPU positions the pop-up widget so that the last menu item chosen is at the mouse pointer. If no menu item has been chosen, VAXTPU positions the pop-up widget so that the mouse pointer is at the upper left corner of the widget.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_REQUIRESDECW	ERROR	Requires the VAXTPU DECwindows screen updater.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_TOOMANY	ERROR	You specified too many parameters.
TPU$_BADKEY	WARNING	You specified an invalid keyword.
TPU$_NEEDTOASSIGN	ERROR	Built-in must return a value.
TPU$_EXTRANEOUSARGS	ERROR	The array of widgets parameter had a non-widget element.
TPU$_REQARGSMISSING	ERROR	The array of widgets parameter was empty.

SET (MESSAGE_ACTION_LEVEL)

FORMAT

SET *(MESSAGE_ACTION_LEVEL,* { *integer* / *keyword* } *)*

PARAMETERS

MESSAGE_ACTION_LEVEL

A keyword indicating that SET is to determine the severity level at which VAXTPU sounds the terminal bell or highlights a message.

integer

A value between 0 and 3 specifying the severity level at which VAXTPU is to take the action you designate. The default value is 2. The severity levels and corresponding values, in ascending order of severity, are as follows:

1	Success
3	Informational
0	Warning
2	Error

VAXTPU performs the action you specify on all completion messages at the severity level you designate and on all messages of greater severity.

keyword

The keyword associated with a VAXTPU completion message. VAXTPU uses the keyword to determine the severity level of the associated completion message and performs the action you specify on all completion messages of that severity level or greater.

DESCRIPTION

To set the action that is taken when VAXTPU returns a completion status of the specified severity, use the SET (MESSAGE_ACTION_TYPE) built-in.

The action you specify using SET (MESSAGE_ACTION_TYPE) is taken for all completion messages of the specified severity or greater severity.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (MESSAGE_ACTION_LEVEL) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.
TPU$_ILLSEVERITY	WARNING	Illegal severity specified; VAXTPU used the severity "error."

EXAMPLES

```
1   SET (MESSAGE_ACTION_TYPE, REVERSE);
    SET (MESSAGE_ACTION_LEVEL, 3);
```

These statements direct VAXTPU to display informational, warning, and error messages in reverse video for 1/2 second, then in ordinary video.

```
2   SET (MESSAGE_ACTION_TYPE, BELL);
    SET (MESSAGE_ACTION_LEVEL, TPU$_SUCCESS);
```

These statements direct VAXTPU to ring the terminal's bell whenever a completion status occurs with a severity equal to or greater than the severity of TPU$_SUCCESS.

SET (MESSAGE_ACTION_TYPE)

FORMAT

SET *(MESSAGE_ACTION_TYPE,* { *NONE* / *BELL* / *REVERSE* } *)*

PARAMETERS

MESSAGE_ACTION_TYPE

A keyword indicating the action to be taken when VAXTPU generates a completion status of the severity you specify.

NONE

A keyword directing VAXTPU to take no action. This is the default.

BELL

A keyword directing VAXTPU to ring the terminal's bell when a completion status of the specified severity is returned.

REVERSE

A keyword directing VAXTPU to display the completion status in reverse video for 1/2 second, then display the status in ordinary video.

DESCRIPTION

To set the severity at which the action is taken, use the SET (MESSAGE_ACTION_LEVEL) built-in. The action you specify using SET (MESSAGE_ACTION_TYPE) is taken for all completion messages of the specified severity or greater severity.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (MESSAGE_ACTION_TYPE) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (MESSAGE_ACTION_TYPE, REVERSE);
SET (MESSAGE_ACTION_LEVEL, 3);
```

These statements direct VAXTPU to display informational, warning, and error messages in reverse video for 1/2 second, then in ordinary video.

SET (MESSAGE_FLAGS)

FORMAT

SET *(MESSAGE_FLAGS, integer)*

PARAMETERS

MESSAGE_FLAGS

The message flags in the $PUTMSG system service.

integer

The value specified for the $PUTMSG message codes. Table 7–11 lists the message codes.

DESCRIPTION

The following table shows the message codes for $PUTMSG:

Table 7–11 Message Codes for $PUTMSG System Service

Bit	Value	Meaning
0	1	Include text of message.
	0	Do not include text of message.
1	1	Include message identifier.
	0	Do not include message identifier.
2	1	Include severity level indicator.
	0	Do not include severity level indicator.
3	1	Include facility name.
	0	Do not include facility name.

If you do not set a value for the message flags, the default message flags for your process are used. Setting the message flags to 0 does not turn off the message text; it causes VAXTPU to use the default message flags for your process. In addition to setting the message flags from within VAXTPU, you can set them at the DCL level with the command SET MESSAGE. The DCL command SET MESSAGE is the only way you can turn off all message text. See the *VMS DCL Dictionary* for information on the DCL command SET MESSAGE.

Table 7–12 shows the predefined constants available for use with SET (MESSAGE_FLAGS).

Table 7–12 Message Flag Values

Bit	Constant	Meaning
0	TPU$K_MESSAGE_TEXT	Include text of message.
1	TPU$K_MESSAGE_ID	Include message identifier.
2	TPU$K_MESSAGE_SEVERITY	Include severity level indicator.
3	TPU$K_MESSAGE_FACILITY	Include facility name.

SIGNALED ERRORS

TPU$_FLAGTRUNC	WARNING	Message flag values must be less than or equal to 15.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_TOOFEW	ERROR	SET (MESSAGE_FLAGS) requires at least two parameters.
TPU$_TOOMANY	ERROR	SET (MESSAGE_FLAGS) accepts no more than two parameters.

EXAMPLES

1
```
SET (MESSAGE_FLAGS, 2)
```

This statement causes the message identifier to be the only item included in VAXTPU messages. The integer 2 sets bit 1.

2
```
SET (MESSAGE_FLAGS, 5)
```

This statement causes the message text and the severity level indicator to be included in VAXTPU messages. The integer 5 is a bit-encoded integer setting both bit 2 and bit 0 to 1.

3
```
SET (MESSAGE_FLAGS, TPU$K_MESSAGE_SEVERITY);
MESSAGE (TPU$_TOOFEW);
```

In this code fragment, the SET (MESSAGE_FLAGS) statement directs VAXTPU to include only the message severity level in messages identified by keywords or integers. Since TPU$_TOOFEW is an error-level message, the MESSAGE statement above causes VAXTPU to display *"%E"* in the message buffer. VAXTPU does not display the text associated with the keyword TPU$_TOOFEW because the statement does not contain an integer or constant directing VAXTPU to display the text. For more information on using constants to specify message format, see the description of the MESSAGE_TEXT built-in.

4
```
SET (MESSAGE_FLAGS, TPU$K_MESSAGE_ID + TPU$K_MESSAGE_TEXT);
MESSAGE (TPU$_TOOFEW);
```

In this code fragment, the integer parameter to SET (MESSAGE_FLAGS) is specified as two constants representing encoded bits. This message flag setting turns on the display of both message identifier and message text. Therefore, when the MESSAGE statement in this code fragment is compiled and executed, VAXTPU displays the words "%TOOFEW, Too few arguments" in the message buffer.

SET (MODIFIABLE)

FORMAT

SET *(MODIFIABLE, buffer,* { *ON* / *OFF* } *)*

PARAMETERS

MODIFIABLE
The ability to modify a buffer.

buffer
The buffer that will either be unmodifiable or able to be edited.

ON
Makes the buffer modifiable.

OFF
Makes the buffer unmodifiable, allowing only deletion of the buffer and setting of marks and ranges. Any attempt to change the buffer will result in a warning message.

DESCRIPTION

When a buffer is not modifiable, any attempt to insert, delete, or otherwise modify the contents of the buffer results in a warning message. This only affects the text within the buffer. The buffer can still be deleted, and marks and ranges can still be created or deleted in the text within the buffer.

Newly created buffers are modifiable by default if a template buffer was not used on the call to the CREATE_BUFFER procedure. The modifiability status is taken from the template buffer if one was specified.

You cannot make the messages buffer unmodifiable.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (MODIFIABLE) built-in requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_MSGBUFSET	ERROR	You cannot force the message buffer to be nonmodifiable.
TPU$_BADKEY	ERROR	Only the ON and OFF keywords are valid.

EXAMPLE

```
SET (MODIFIABLE, CURRENT_BUFFER, OFF)
```

This statement makes the current buffer unmodifiable. Any attempt to change the buffer fails with a warning message.

SET (MODIFIED)

Turns on or turns off the flag indicating that the specified buffer has been modified.

FORMAT

SET *(MODIFIED, buffer,* { *ON* / *OFF* } *)*

PARAMETERS

MODIFIED

A keyword directing VAXTPU to turn on or turn off the indicator designating a buffer as modified.

buffer

The buffer whose indicator you want to control.

ON

A keyword directing VAXTPU to mark a buffer as modified.

OFF

A keyword directing VAXTPU to mark a buffer as unmodified.

DESCRIPTION

Use SET (MODIFIED) with caution. When you turn off the flag indicating that the buffer is modified, it is possible to exit from an application layered on VAXTPU without writing out the contents of a modified buffer. Be sure your extension or layered application handles this possibility.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (MODIFIED) cannot return a value.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (MODIFIED) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (MODIFIED) built-in.

EXAMPLE

```
SET (MODIFIED, CURRENT_BUFFER, ON);
```

This statement marks the current buffer as modified.

SET (MOUSE)

FORMAT

[{ ON / OFF } :=] SET *(MOUSE,* { *ON / OFF* } *)*

PARAMETERS

MOUSE

Indicates that you are using SET to enable or disable VAXTPU's mouse support.

The default mouse setting depends on the terminal you are using. If the VAXTPU statement GET_INFO (SCREEN, "dec_crt2") returns true on your terminal, mouse support is turned on by default. Otherwise, mouse support is turned off by default.

ON

Causes VAXTPU to recognize mouse buttons when they are pressed, and allows you to bind programs or procedures to mouse buttons. Enables the LOCATE_MOUSE and POSITION (MOUSE) built-ins.

OFF

Disables VAXTPU mouse support. Pressing a mouse button when the mouse is set to OFF has no effect.

DESCRIPTION

Since VAXTPU mouse support disables the terminal emulator's cut and paste feature in non-DECwindows VAXTPU, you must turn off VAXTPU mouse support to use the non-VAXTPU cut and paste capability while VAXTPU is running.

The optional return value specifies whether VAXTPU mouse support was enabled or disabled before the current SET (MOUSE) statement was executed. This allows you to enable or disable mouse support and then reset the support to its previous setting without having to make a separate call.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	The keyword must be either ON or OFF.
TPU$_MOUSEINV	WARNING	You have tried to enable mouse support on an incompatible terminal.
TPU$_TOOFEW	ERROR	SET (MOUSE) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLE

```
SET (MOUSE, OFF)
```

This statement turns off mouse support.

SET (NO_WRITE)

FORMAT

SET *(NO_WRITE, buffer* [[{ , *ON* / , *OFF* }]]*)*

PARAMETERS

NO_WRITE

Specifies that VAXTPU should not create an output file from the contents of a buffer after execution of a QUIT or EXIT statement even if the contents of the buffer have been modified.

By default, a buffer is written out if it has been modified.

buffer

The buffer whose contents you do not want written out.

ON

Causes the buffer you name not to be written out.

OFF

Lets you change a buffer from the no-write state to the default state. By default, any modified buffers are written out after execution of a QUIT or EXIT statement.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (NO_WRITE) requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLES

1
```
SET (NO_WRITE, my_buffer)
```

This statement causes *my_buffer* not to be saved in a file after execution of a QUIT or EXIT statement.

2
```
SET (NO_WRITE, my_buffer, OFF)
```

This statement turns off the no-write state of *my_buffer*. The contents of the buffer are written out after execution of a QUIT or EXIT statement if the buffer has been modified.

SET (OUTPUT_FILE)

FORMAT

SET *(OUTPUT_FILE, buffer, string)*

PARAMETERS

OUTPUT_FILE

A keyword indicating that SET is to control creation of an output file for the contents of a buffer after execution of a QUIT or EXIT statement.

buffer

The buffer whose contents are written to the specified file.

string

The file specification for the file being written out.

The default output file is the input file name and the highest existing version number for that file plus 1.

DESCRIPTION

VAXTPU does not write out the contents of a buffer after execution of a QUIT or EXIT statement if the buffer has not been modified.

If a buffer is set to NO_WRITE, a file is not written out after execution of a QUIT or EXIT statement even though you specified a file specification for the contents of the buffer with the built-in procedure SET (OUTPUT_FILE).

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (OUTPUT_FILE) requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (OUTPUT_FILE, paste_buffer, "newfile.txt")
```

This statement causes the output file for *paste_buffer* to be NEWFILE.TXT.

SET (OVERSTRIKE)

FORMAT

SET *(OVERSTRIKE, buffer)*

PARAMETERS

OVERSTRIKE

A keyword specifying that SET is to control the mode of text entry. OVERSTRIKE means that the characters that you add to the buffer replace the characters in the buffer starting at the editing point and continuing for the length of the text that you enter.

The default mode of text entry is INSERT.

See also the description of the built-in procedure SET (INSERT). For information on how to control overstrike behavior in tabs, see SET (PAD_OVERSTRUCK_TABS).

buffer

The buffer whose mode of text entry you want to set.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (OVERSTRIKE) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (OVERSTRIKE, my_buffer)
```

This statement sets the mode for text entry in *my_buffer* to overstrike. Characters that you enter replace characters already in the buffer, starting at the editing point and continuing for the length of the text that you enter.

SET (PAD)

FORMAT

SET *(PAD, window,* { *ON* / *OFF* } *)*

PARAMETERS

PAD

A keyword indicating that SET is to control whether screen lines are padded with blanks. This keyword determines whether SET pads out the left and right ends of lines, beyond the text on the line. When video attributes are applied to a padded window, the window has an even or "boxed" appearance.

window

The window in which lines are padded.

ON

Causes VAXTPU to display blanks after the last character of a record so that the screen line extends to the right side of the window. If there are not enough lines in a buffer to fill an entire window, VAXTPU displays blank lines (according to the video setting of the window) from the end-of-buffer line to the end of the window.

OFF

Causes the display of lines on the screen to stop at the last character of a record. When video attributes are applied to the window, the window has a ragged appearance on the sides.

DESCRIPTION

By default, VAXTPU ends a line on the screen at the end of a record, without adding padding blanks. The default behavior of not padding the screen gives maximum editing performance. You can change the default with SET (PAD) for special visual effects. The records in the buffer are not padded; only the display lines have the padding.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (PAD) requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	The keyword must be ON or OFF.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.

EXAMPLE

```
SET (PAD, second_window, ON);
SET (VIDEO, second_window, REVERSE);
```

The first statement causes *second_window* to be blank padded. The second statement causes *second_window* to be displayed in reverse video. The window has an even right and left margin when displayed.

SET (PAD_OVERSTRUCK_TABS)

FORMAT

SET *(PAD_OVERSTRUCK_TABS,* { *ON* / *OFF* } *)*

PARAMETERS

PAD_OVERSTRUCK_TABS

How tabs are handled in overstrike mode.

ON

Causes the insertion of one or more characters on top of a tab in overstrike mode, as if the text insertion mode were INSERT instead of OVERSTRIKE for the width of the tab.

OFF

Causes overstruck tabs to be replaced by the first character that is inserted in the buffer in overstrike mode on top of a tab. This is the default setting.

DESCRIPTION

PAD_OVERSTRUCK_TABS controls how VAXTPU handles tabs in overstrike mode. When earlier versions of VAXTPU overstruck a tab, VAXTPU inserted spaces if necessary to preserve the cursor position within the tab, and then replaced the tab with the character that was being entered. This behavior is preserved when PAD_OVERSTRUCK_TABS is set OFF.

When PAD_OVERSTRUCK_TABS is set ON, VAXTPU inserts spaces as necessary to preserve the cursor position within the tab of the first character of the text, and then inserts the text. The tab is only replaced when it occupies a single column.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (PAD_OVERSTRUCK_TABS) built-in requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	WARNING	Only ON and OFF are allowed.

EXAMPLES

The following examples show what happens when PAD_OVERSTRUCK_TABS is set to OFF. In these examples, the character ">" represents the tab, the character "." represents one column of white space, and an underscore (_) represents the cursor.

Suppose a buffer contains the following text, with the cursor in the middle of white space created by a tab:

```
abc>._..def
```

Suppose the user inserts the character "*" while PAD_OVERSTRUCK_TABS is set to OFF. The white space to the left of the * is preserved. The tab character is removed, and the white space to the right of the * is not preserved. The text to the right of the collapsed white space moves leftward. The result is as follows:

```
abc..*def
```

Note that the cursor is on the "d" character. Suppose, given the same initial text, the user types the string "xyzzy" while PAD_OVERSTRUCK_TABS is set to OFF. The tab is removed. The text to the right of the tab moves leftward. The user's new string, "xyzzy", is written over the old text. The result is as follows:

```
abc..xyzzy
```

When PAD_OVERSTRUCK_TABS is set to ON, the text to the right of the tab does not move to the left when text is inserted within the tab. Instead of removing the tab, VAXTPU places the tab to the right of the inserted text if the inserted text is shorter than the length of the tab. The newly placed tab creates only enough white space to preserve the original column position of the text to the right of the tab.

The following examples show what happens when PAD_OVERSTRUCK_TABS is set to ON. In these examples, the character ">" represents the tab, the character "." represents one column of white space, and the underscore (_) represents the cursor.

Suppose a buffer contains the following text, with the cursor in the middle of white space created by a tab:

```
abc>._..def
```

Suppose the user inserts the character "*" while PAD_OVERSTRUCK_TABS is set to ON. The white space to the left of the * is preserved. The tab is inserted after the * character. The result is as follows:

```
abc..*>.def
```

Suppose, given the same initial text, the user inserts the string "xyzzy" while PAD_OVERSTRUCK_TABS is set to ON. To preserve the original position of the text to the right of the tab, VAXTPU fills the white space created by the tab with characters from the new string. When the white space is filled, VAXTPU writes the new characters over the old characters. Thus, the old text does not move left or right, but rather is overwritten by the new text. The result is as follows:

```
abc..xyzzyf
```

SET (PERMANENT)

FORMAT

SET *(PERMANENT, buffer)*

PARAMETERS

PERMANENT

Specifies that a buffer cannot be deleted. By default, buffers can be deleted; they are not permanent.

buffer

The buffer that is not to be deleted.

DESCRIPTION

Once you use SET (PERMANENT, buffer) to make a buffer permanent, you cannot reset the buffer so that it can be deleted.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (PERMANENT) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (PERMANENT, master_buffer)
```

This statement causes *master_buffer* to become a permanent buffer.

SET (POST_KEY_PROCEDURE)

FORMAT

SET *(POST_KEY_PROCEDURE, string1* [[{ , *buffer* / , *learn_sequence* / , *program* / , *range* / , *string2* }]] *)*

PARAMETERS

POST_KEY_PROCEDURE

The action taken after the code or learn sequence bound to a key is executed.

string1

A quoted string, or a variable name representing a string constant, that specifies the key map list for which this procedure is called.

buffer

The buffer containing VAXTPU statements specifying the action to be taken after the code or learn sequence bound to a key is executed. SET (POST_KEY_PROCEDURE) compiles the statements in the buffer and stores the resulting program in the specified key map list.

learn_sequence

The learn sequence specifying the action to be taken after the code or learn sequence bound to a key is executed. The contents of a variable of type learn do not require compilation. SET (POST_KEY_PROCEDURE) stores the learn sequence in the specified key map list.

program

The program specifying the action to be taken after the code or learn sequence bound to a key is executed. The contents of a variable of type program do not require compilation. SET (POST_KEY_PROCEDURE) stores the program in the specified key map list.

range

The range containing VAXTPU statements specifying the action to be taken after the code or learn sequence bound to a key is executed. SET (POST_KEY_PROCEDURE) compiles the statements in the range and stores the resulting program in the specified key map list.

string2

The string containing VAXTPU statements specifying the action to be taken after the code or learn sequence bound to a key is executed. SET (POST_KEY_PROCEDURE) compiles the statements in the string and stores the resulting program in the specified key map list.

DESCRIPTION

Postkey procedures allow an editor to perform some specified action before and after execution of code bound to a key. If you do not specify the third parameter, the postkey procedure for the specified key map list is deleted.

Pre- and postkey procedures interact with learn sequences in the following order:

1 When the user presses the key or key sequence to which the learn sequence is bound, VAXTPU executes the prekey procedure of that key if a prekey procedure has been set.

2 For each key in the learn sequence, VAXTPU executes procedures or programs in the following order:

   a. VAXTPU executes the prekey procedure of that key if a prekey procedure has been set.

   b. VAXTPU executes the code bound to the key itself.

   c. VAXTPU executes the postkey procedure of that key if a postkey procedure has been set.

3 When all keys in the learn sequence have been processed, VAXTPU executes the postkey procedure, if one has been set, for the key to which the entire learn sequence was bound.

The pre- and postkey procedures bound to a key map list can be found by using the following calls to the GET_INFO built-in procedure:

```
GET_INFO (key_map_list_name, "pre_key_procedure")
GET_INFO (key_map_list_name, "post_key_procedure")
```

By default, newly created key map lists do not have postkey procedures.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (POST_KEY_PROCEDURE) built-in requires at least two parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_COMPILEFAIL	ERROR	Compilation aborted because of syntax errors.
TPU$_NOKEYMAPLIST	WARNING	Attempt to access an undefined key map list.

EXAMPLE

```
SET (POST_KEY_PROCEDURE, "tpu$key_map_list",
        'MESSAGE ("Key " + GET_INFO (LAST_KEY, "name") + " Executed")');
```

This code displays a message after the code bound to a key is executed.

SET (PRE_KEY_PROCEDURE)

FORMAT

SET *(PRE_KEY_PROCEDURE, string1* [[{ , *buffer* | , *learn_sequence* | , *program* | , *range* | , *string2* }]] *)*

PARAMETERS

PRE_KEY_PROCEDURE

The action taken before the code or learn sequence bound to a key is executed.

string1

A quoted string, or a variable name representing a string constant, that specifies the key map list for which this procedure is called.

buffer

The buffer containing VAXTPU statements specifying the action to be taken before the code or learn sequence bound to a key is executed. SET (PRE_KEY_PROCEDURE) compiles the statements in the buffer and stores the resulting program in the specified key map list.

learn_sequence

The learn sequence specifying the action to be taken before the code or learn sequence bound to a key is executed. The contents of a variable of type learn do not require compilation. SET (PRE_KEY_PROCEDURE) stores the learn sequence in the specified key map list.

program

The program specifying the action to be taken before the code or learn sequence bound to a key is executed. The contents of a variable of type program do not require compilation. SET (PRE_KEY_PROCEDURE) stores the program in the specified key map list.

range

The range containing VAXTPU statements specifying the action to be taken before the code or learn sequence bound to a key is executed. SET (PRE_KEY_PROCEDURE) compiles the statements in the range and stores the resulting program in the specified key map list.

string2

The string containing VAXTPU statements specifying the action to be taken before the code or learn sequence bound to a key is executed. SET (PRE_KEY_PROCEDURE) compiles the statements in the string and stores the resulting program in the specified key map list.

DESCRIPTION

Prekey procedure allows an editor to perform some specified action before the execution of code bound to a key. If you do not specify the third parameter, the prekey procedure for the specified key map list is deleted.

Pre- and postkey procedures interact with learn sequences in the following order:

1 When the user presses the key or key sequence to which the learn sequence is bound, VAXTPU executes the prekey procedure of that key if a prekey procedure has been set.

2 For each key in the learn sequence, VAXTPU executes procedures or programs in the following order:

   a. VAXTPU executes the prekey procedure of that key if a prekey procedure has been set.

   b. VAXTPU executes the code bound to the key itself.

   c. VAXTPU executes the postkey procedure of that key if a postkey procedure has been set.

3 When all keys in the learn sequence have been processed, VAXTPU executes the postkey procedure, if one has been set, for the key to which the entire learn sequence was bound.

The prekey procedure or postkey procedure bound to a key map list can be found by using the following calls to the GET_INFO built-in procedure:

```
GET_INFO (key_map_list_name, "pre_key_procedure");
GET_INFO (key_map_list_name, "post_key_procedure");
```

By default, newly created key map lists do not have prekey procedures.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (PRE_KEY_PROCEDURE) built-in requires at least two parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_COMPILEFAIL	ERROR	Compilation aborted because of syntax errors.
TPU$_NOKEYMAPLIST	WARNING	Attempt to access an undefined key map list.

EXAMPLE

```
SET (PRE_KEY_PROCEDURE, "tpu$key_map_list",
        'MESSAGE ("Working...")');
```

This code displays a message before the code bound to a key is executed.

SET (PROMPT_AREA)

FORMAT

SET *(PROMPT_AREA, integer1, integer2,* { *NONE* | *BOLD* | *BLINK* | *REVERSE* | *UNDERLINE* } *)*

PARAMETERS

PROMPT_AREA

An area on the screen in which the prompts generated by the built-in procedure READ_LINE are displayed.

By default, there is no prompt area.

integer1

The screen line number at which the prompt area starts.

integer2

The number of screen lines in the prompt area.

NONE

Applies no video attributes to the characters in the prompt area.

BOLD

Causes the characters in the prompt area to be bolded.

BLINK

Causes the characters in the prompt area to blink.

REVERSE

Causes the characters in the prompt area to be displayed in reverse video.

UNDERLINE

Causes the characters in the prompt area to be underlined.

DESCRIPTION

If the prompt area overlaps a line of a window that is visible on the screen, the line is erased when the built-in procedure READ_LINE is executed. When the execution of READ_LINE is completed, the line is restored. If the prompt area does not overlap any windows, the prompt area continues to display the READ_LINE prompt and your input until new information is sent to the prompt area.

If you have a multiple-line prompt area and your terminal has hardware scrolling capabilities, the first prompt appears on the last line of the prompt area and as subsequent prompts are issued, the previous prompts scroll up to make room for new ones. If there are more prompts than there are prompt-area lines, the extra prompts are scrolled out of the window.

If your terminal does not have hardware scrolling capabilities, prompts are displayed starting at the first line in the prompt area. When the prompt area is filled, display starts again at the first line in the prompt area.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (PROMPT_AREA) requires four parameters.
TPU$_TOOMANY	ERROR	You specified more than four parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	The keyword must be NONE, BOLD, BLINK, REVERSE, or UNDERLINE.
TPU$_UNKKEYWORD	ERROR	You have specified an unknown keyword.
TPU$_BADFIRSTLINE	WARNING	Prompt area must not start off screen, or be less than one line long.
TPU$_BADPROMPTLEN	WARNING	Prompt area must not extend off screen.

EXAMPLE

```
SET (PROMPT_AREA, 24, 1, REVERSE)
```

This statement causes the prompt area to be screen line number 24. It is one line and is displayed in reverse video.

SET (RECORD_ATTRIBUTE)

Sets or alters any of three possible attributes for the specified record or records. The attributes you can set for a record are its left margin, its modifiability, and its visibility.

FORMATS

SET *(RECORD_ATTRIBUTE,* { *mark* / *range* / *buffer* }*,* { *DISPLAY_VALUE* / *LEFT_MARGIN* }*,*
{ *display_setting_integer* / *margin_setting_integer* }*)*

or

SET *(RECORD_ATTRIBUTE,* { *marker,* / *range,* / *buffer,* } *MODIFIABLE,* { *ON* / *OFF* }*)*

PARAMETERS

RECORD_ATTRIBUTE
A keyword indicating that the SET built-in is being used to specify or change a record attribute.

marker
The marker indicating the record whose attribute you want to set.

range
The range containing the records whose attribute you want to set. The record attribute is applied to all records in the range. Records that are partially within the range will be modified.

buffer
The buffer containing the records for which you want to set an attribute. The record attribute is applied to all records in the buffer.

DISPLAY_VALUE
A keyword indicating that you want to affect the visibility of the records. If you specify the DISPLAY_VALUE keyword as the third parameter, you must specify for the fourth parameter an integer providing a display setting.

LEFT_MARGIN
A keyword indicating that you want to specify the left margin for the specified records. If you specify the LEFT_MARGIN keyword as the third parameter, you must specify for the fourth parameter an integer providing a left margin value.

display_setting_integer
An integer value from –127 to +127. This is the display setting. To determine whether a record is to be visible or invisible in a given window, VAXTPU compares the record's display setting to the window's display

setting. (A window's display setting is specified with SET (DISPLAY_VALUE).) If the record's setting is greater than or equal to the window's setting, VAXTPU makes the record visible in that window; otherwise, VAXTPU makes the record invisible.

margin_setting_integer

An integer that is the column at which the left margin should be set. The value must be between 1 and the value of the right margin minus 1. (The maximum valid value for the right margin is 32767.)

MODIFIABLE

A keyword indicating that you want to determine whether the specified records are modifiable. If you specify the MODIFIABLE keyword as the third parameter, you must specify either ON or OFF as the fourth parameter.

ON

A keyword making records modifiable. Note, if a buffer is modifiable, you can use SET (RECORD_ATTRIBUTE) to make a record in the buffer unmodifiable (with keyword OFF). If a buffer is unmodifiable and you use SET (RECORD_ATTRIBUTE) to make a record in the buffer modifiable (with keyword ON), VAXTPU marks the record as modifiable but does not allow modifications to the record until the buffer is made modifiable.

OFF

A keyword making records unmodifiable.

DESCRIPTION

With each call to SET (RECORD_ATTRIBUTE), you can set only one attribute. For example, you cannot change visibility and modifiability using just one call. To set more than one record attribute, use multiple calls to SET (RECORD_ATTRIBUTE).

When you set an attribute for multiple records, each record gets the same value. For example, if you specify a range of records and a value for the left margin attribute, all records in the range receive the same left margin value.

You cannot change the left margin of an unmodifiable record. You can change the display value of a record at any time.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	You specified too many parameters.
TPU$_TOOFEW	ERROR	You specified too few parameters.
TPU$_INVPARAM	ERROR	The third parameter must be a keyword.
TPU$_ARGMISMATCH	ERROR	The second or fourth parameter has an incorrect type.
TPU$_BADKEY	WARNING	You have specified an invalid keyword.

TPU$_BADDISPVAL	WARNING	Display values must be between –127 and +127.
TPU$_BADMARGINS	WARNING	You have specified an illegal left margin value.

EXAMPLES

1
```
SET (MODIFIABLE, buf1, OFF);
r1:= CREATE_RANGE (BEGINNING_OF(buf1), END_OF(buf1), REVERSE);
SET (RECORD_ATTRIBUTE, r1, MODIFIABLE, OFF);
SET (RECORD_ATTRIBUTE, r1, MODIFIABLE, ON);
SET (MODIFIABLE, buf1, ON);
```

This code fragment uses statements that change buffer modifiability and record modifiability independently. Note that you can turn on the modifiability of a record or range of records even when the buffer's modifiability is turned off.

2
```
SET (RECORD_ATTRIBUTE, CURRENT_BUFFER, LEFT_MARGIN, 3);
```

This statement sets the left margin of all records in the current buffer to column 3.

3
```
SET (DISPLAY_VALUE, CURRENT_WINDOW, 0);
SET (RECORD_ATTRIBUTE, SELECT_RANGE, -1);
```

These statements make the records in the range *select_range* invisible in the current window.

4
```
SET (RECORD_ATTRIBUTE, MARK (FREE_CURSOR), MODIFIABLE, OFF);
```

This statement makes the current record unmodifiable.

SET (RESIZE_ACTION)

Specifies code to be executed when a resize event has occurred. Specifying a resize action routine overrides any previous resize action routines that have been defined.

FORMAT

SET *(RESIZE_ACTION* [, { *buffer* | *learn_sequence* | *program* | *range* | *string* | *NONE* }] *)*

PARAMETERS

RESIZE_ACTION

A keyword directing VAXTPU to set an attribute related to a resize action routine.

buffer

The buffer that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

learn_sequence

The learn sequence that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

program

The program that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

range

The range that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

string

The string that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

NONE

A keyword directing VAXTPU to delete the resize action routine. If you specify this keyword or do not specify the parameter at all, the application is not notified when a resize event occurs.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (RESIZE_ACTION) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (RESIZE_ACTION) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (RESIZE_ACTION) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (RESIZE_ACTION) built-in.

EXAMPLE

```
SET (RESIZE_ACTION, "eve$$resize_action");
```

This statement specifies the procedure EVE$$RESIZE_ACTION as the resize routine. To see this statement used in an initializing procedure, see the example in the description of the SET (SCREEN_LIMITS) built-in.

SET (REVERSE)

FORMAT

SET *(REVERSE, buffer)*

PARAMETERS

REVERSE

The direction of the buffer. REVERSE means to go toward the beginning of the buffer.

The default direction for a buffer is forward.

buffer

The buffer whose direction you want to set.

DESCRIPTION

Interfaces use this feature to keep track of direction for searching or movement.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (REVERSE) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (REVERSE, my_buffer)
```

This statement causes the direction of the buffer to be toward the beginning of the buffer.

SET (RIGHT_MARGIN)

FORMAT

SET *(RIGHT_MARGIN, buffer, integer)*

PARAMETERS

RIGHT_MARGIN

The right margin of a buffer.

buffer

The buffer in which the right margin is being set.

integer

The column at which the right margin is set.

DESCRIPTION

The SET (RIGHT_MARGIN) built-in procedure allows you to change only the right margin of a buffer.

Newly created buffers receive a right margin of 80 if a template buffer is not specified on the call to the CREATE_BUFFER built-in procedure. If a template buffer is specified, the right margin of the template buffer is used.

Use SET (RIGHT_MARGIN) to override the default right margin.

The buffer margin settings are independent of the terminal width or window width settings.

The built-in procedure FILL uses these margin settings when it fills the text of a buffer.

The SET (RIGHT_MARGIN) built-in procedure controls the buffer margin setting even if the terminal width or window width is set to something else.

The value of the right margin must be less than the maximum record size for the buffer, and greater than the left margin value. You can use the built-in procedure GET_INFO (buffer, "record_size") to find out the maximum record size of a buffer.

If you want to use the margin settings of an existing buffer in a user-written procedure, the statements GET_INFO (buffer, "left_margin") and GET_INFO (buffer, "right_margin") return the values of the margin settings in the specified buffer.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (RIGHT_MARGIN) built-in requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.

TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADMARGINS	WARNING	Right must be greater than left; both must be greater than zero.

EXAMPLES

1
```
SET (RIGHT_MARGIN, my_buffer, 132)
```

This statement causes the right margin of the buffer represented by the variable *my_buffer* to be changed. The right margin of the buffer is set to 132. The left margin is unchanged.

2
```
SET (RIGHT_MARGIN, CURRENT_BUFFER, 70)
```

This statement causes the right margin of the current buffer to be changed to 70. As above, the left margin is unchanged.

SET (RIGHT_MARGIN_ACTION)

FORMAT

SET *(RIGHT_MARGIN_ACTION, buffer1* [, { *buffer2* | *learn_sequence* | *program* | *range* | *string* }] *)*

PARAMETERS

RIGHT_MARGIN_ACTION

Refers to the action taken when the user presses a self-inserting key while the cursor is to the right of a buffer's right margin. A self-inserting key is one that is associated with a printable character.

buffer1

The buffer in which the right margin action routine is being set.

buffer2

A buffer containing the VAXTPU statements to be executed when the user presses a self-inserting key while the cursor is to the right of a buffer's right margin.

learn_sequence

A learn sequence that is to be replayed when the user presses a self-inserting key while the cursor is to the right of a buffer's right margin.

program

A program that is to be executed when the user presses a self-inserting key while the cursor is to the right of a buffer's right margin.

range

A range that contains VAXTPU statements that are to be executed when the user presses a self-inserting key while the cursor is to the right of a buffer's right margin.

string

A string that contains VAXTPU statements that are to be executed when the user presses a self-inserting key while the cursor is to the right of a buffer's right margin.

DESCRIPTION

The SET (RIGHT_MARGIN_ACTION) built-in procedure allows you to specify an action to be taken when the user attempts to insert text to the right of the right margin of a line. If the third parameter is not specified, the right margin action routine is deleted. If no right margin action routine has been specified, the text is simply inserted at the current position after any necessary padding spaces.

Newly created buffers do not receive a right margin action routine if a template buffer is not specified on the call to the CREATE_BUFFER built-in procedure. If a template buffer is specified, the right margin action routine of the template buffer is used.

The right margin action routine only affects text entered from the keyboard or a learn sequence. Inserting text into a buffer to the right of the right margin using the COPY_TEXT or MOVE_TEXT built-in procedures does not trigger the right margin action routine.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (RIGHT_MARGIN_ ACTION) built-in requires at least two parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_COMPILEFAIL	ERROR	Compilation aborted because of syntax errors.

EXAMPLES

1
```
SET (RIGHT_MARGIN_ACTION, CURRENT_BUFFER, "fill_current_line")
```

This statement causes the procedure FILL_CURRENT_LINE to be executed when the user attempts to type a character to the right of the right margin of the current line. A typical right margin action routine invokes the FILL built-in to fill the current line and force text to the right of the right margin to a new line.

2
```
SET (RIGHT_MARGIN_ACTION, CURRENT_BUFFER)
```

This statement deletes any right margin action routine that may be defined for the current buffer. If the user attempts to type a character to the right of the right margin of the current line, the text is inserted with spaces padding the text from the end of the line.

SET (SCREEN_LIMITS)

Specifies the minimum and maximum allowable sizes for the VAXTPU screen during resize operations. VAXTPU passes these limits to the DECwindows window manager, which is free to use or ignore the limits.

FORMAT

SET *(SCREEN_LIMITS, array)*

PARAMETERS

SCREEN_LIMITS

A keyword directing VAXTPU to pass hints to the DECwindows window manager about screen size.

array

An integer-indexed array using four elements to specify hints for the minimum and maximum screen width and length. The array indices and their corresponding elements are as follows:

1 The minimum screen width, in columns. This value must be at least 0 and less than or equal to the maximum screen width. The default value is 0.

2 The minimum screen length, in lines. This value must be at least 0 and less than or equal to the maximum screen length. The default value is 0.

3 The maximum screen width, in columns. This value must be greater than or equal to the minimum screen width and less than or equal to 255. The default value is 255.

4 The maximum screen length, in lines. This value must be greater than or equal to the minimum screen length and less than or equal to 255. The default value is 255.

SIGNALED ERRORS

TPU$_BADVALUE	WARNING	An integer parameter was specified with a value outside the valid range.
TPU$_MAXVALUE	WARNING	You specified a value higher than the maximum allowable value.
TPU$_MINVALUE	WARNING	You specified a value lower than the minimum allowable value.
TPU$_EXTRANEOUSARGS	ERROR	One or more extraneous arguments have been specified for a DECwindows built-in.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.

TPU$_NORETURNVALUE	ERROR	SET (SCREEN_LIMITS) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (SCREEN_LIMITS) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (SCREEN_LIMITS) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (SCREEN_LIMITS) built-in.
TPU$_REQARGSMISSING	ERROR	One or more required arguments are missing.

EXAMPLE

```
PROCEDURE eve$$decwindows_init     ! Module Initialization

LOCAL   temp_array;

eve$x_decwindows_active := GET_INFO (SCREEN, "decwindows");

!  .
!  .
!  .

IF NOT eve$x_decwindows_active
THEN
    RETURN (FALSE)
ENDIF;

! The following statements set the package up to handle resize actions.

temp_array := CREATE_ARRAY (4);
temp_array {1} := 20;   ! Minimum width.
temp_array {2} := 6;    ! Minimum height.
temp_array {3} := 250;  ! Maximum width.
temp_array {4} := 100;  ! Maximum height.

SET (SCREEN_LIMITS, temp_array);
SET (RESIZE_ACTION, "eve$$resize_action");
SET (ENABLE_RESIZE, ON);

!  .
!  .
!  .

ENDPROCEDURE;
```

These statements show one possible way that a layered application can use the SET (SCREEN_LIMITS) built-in. The statements are a portion of the EVE procedure EVE$$DECWINDOWS_INIT. You can find the original version in SYS$EXAMPLES:EVE$DECWINDOWS.TPU.

The procedure EVE$$DECWINDOWS_INIT is the module initialization procedure for the package EVE$DECWINDOWS.

SET (SCREEN_UPDATE)

Turns on or turns off support for screen updating. For more information on screen updating, see Section 6.2.

FORMAT

[{ **ON** / **OFF** } **:=] SET** *(SCREEN_UPDATE,* { *ON* / *OFF* } *)*

PARAMETERS

SCREEN_UPDATE

A keyword directing VAXTPU to set an attribute of screen updating.

ON

A keyword indicating that screen updating is enabled.

OFF

A keyword indicating that screen updating is disabled.

return value

A variable containing the keyword value ON or OFF. The keyword specifies whether VAXTPU screen updating support was enabled or disabled before the current SET (SCREEN_UPDATE) statement was executed. Using the returned variable, you can enable or disable screen updating and then reset the support to its previous setting without having to make a separate call to fetch the previous setting.

DESCRIPTION

When you set SCREEN_UPDATE on, the screen manager is immediately called to update the screen. The extent of the update depends on the built-ins that have been used since the last screen update. The update may range from a complete screen refresh to an updating of the existing text on the screen.

For more information on screen updating, see Section 6.2.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	The keyword must be ON or OFF.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_TOOFEW	ERROR	SET (SCREEN_UPDATE) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_UNKKEYWORD	ERROR	You have specified an unknown keyword.

EXAMPLE

```
SET (SCREEN_UPDATE, OFF)
```

This statement causes screen updating to be turned off. When you design an editing interface, you may want to use this statement to prevent some intermediate processing steps from appearing on the screen.

SET (SCROLL_BAR)

Enables a horizontal or vertical scroll bar for the specified window.

FORMAT

[{ integer / widget } :=] SET (SCROLL_BAR, window, { HORIZONTAL / VERTICAL }, { ON / OFF })

PARAMETERS

SCROLL_BAR
A keyword directing VAXTPU to enable or disable a scroll bar in a VAXTPU window.

window
The window in which the scroll bar does or does not appear.

HORIZONTAL
A keyword directing VAXTPU to enable or disable a horizontal scroll bar.

VERTICAL
A keyword directing VAXTPU to enable or disable a vertical scroll bar.

ON
A keyword indicating that the scroll bar is to be visible in the specified window.

OFF
A keyword indicating that the scroll bar is not to be visible in the specified window.

return value

integer	The value 0 if an error prevents VAXTPU from associating a widget with the window.
widget	The widget instance implementing the vertical or horizontal scroll bar associated with a window.

DESCRIPTION

Scroll bars represent the location of the editing point in the buffer. By dragging the scroll bar's slider, the user can reposition the editing point in the buffer mapped to the window. Scroll bars are unique among VAXTPU widgets in the following respects:

- Each scroll bar widget is associated with a specific VAXTPU window.
- Instead of handling scroll widgets at the application level, you can direct VAXTPU to handle resizing and repositioning of the scroll bar slider. VAXTPU always handles sizing and positioning of the scroll bar itself.

Note that windows having fewer than four lines of text cannot display a vertical scroll bar. Similarly, a window less than four columns wide cannot display a horizontal scroll bar.

SET (SCROLL_BAR) returns the scroll bar widget, or 0 if an error prevents VAXTPU from associating a widget with the window.

By default, VAXTPU creates its windows without any scroll bars; using SET (SCROLL_BAR) with the keyword ON overrides the default. To make a scroll bar invisible after it has been placed in a window (for example, to allow the user of a layered application to turn off scroll bars), use SET (SCROLL_BAR) with the keyword OFF.

When the size of a VAXTPU window changes, VAXTPU automatically adjusts the scroll bar to fit the new window size. If a window becomes too small to support a scroll bar, VAXTPU turns off the scroll bar. However, if the window subsequently becomes larger, VAXTPU automatically turns the scroll bar back on.

The height of a vertical scroll bar represents the total number of lines in the buffer mapped to the window.

The width of a horizontal scroll bar represents the greater of the following:

- The width of the widest line in the set of lines visible in the window. "Width" means the distance from the first character on the line to the last character, regardless of whether all characters on the line are visible.
- In a case where none of the lines in the set of lines visible in the window has text extending all the way to the rightmost window column, the width of the widest line from the first character on the line to the rightmost window column.

Note that the horizontal scroll bar represents only the lines that are visible in the window, not all the lines in the buffer mapped to the window.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_REQUIRESDECW	ERROR	You can use the SET (SCROLL_BAR) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (SCROLL_BAR) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (SCROLL_BAR) built-in.

EXAMPLE

```
vertical_bar := SET (SCROLL_BAR, CURRENT_WINDOW, VERTICAL, ON);
```

This statement turns on a vertical scroll bar in the current window.

For sample code using the SET (SCROLL_BAR) built-in, see Example B–7.

SET (SCROLL_BAR_AUTO_THUMB)

Enables or disables automatic adjustment of the scroll bar slider.

FORMAT

SET *(SCROLL_BAR_AUTO_THUMB, window,* { *HORIZONTAL* / *VERTICAL* }*,*
{ *ON* / *OFF* }*)*

PARAMETERS

SCROLL_BAR_AUTO_THUMB

A keyword directing VAXTPU to enable or disable automatic adjustment of the scroll bar slider in a VAXTPU window.

window

The window whose scroll bar slider you want VAXTPU to adjust.

HORIZONTAL

A keyword directing VAXTPU to set the slider on a horizontal scroll bar.

VERTICAL

A keyword directing VAXTPU to set the slider on a vertical scroll bar.

ON

A keyword directing VAXTPU to enable automatic adjustment of the scroll bar slider.

OFF

A keyword directing VAXTPU to disable automatic adjustment of the scroll bar slider.

DESCRIPTION

By default, SET (SCROLL_BAR_AUTO_THUMB) is set to ON and VAXTPU automatically manages a window's scroll bar slider in the following ways:

- Adjusts the size of the slider as the user adds, deletes, or moves text, so that the slider size represents the amount of visible text in relation to the total amount of text
- Adjusts the size of the slider whenever the size of the window and the size of the scroll bar change, so that the slider size remains proportional to the scroll bar size
- Adjusts the position of the slider as the user adds, deletes, or moves text, so that the slider shows whether the current buffer or line contains text not visible on the screen and, if so, where the invisible text is in relation to the visible text

When the scroll bar slider is adjusted automatically, the width of the slider in a horizontal scroll bar represents the width of the window. For example, the size of the slider changes when the window width is changed from 80 to 132 columns or the reverse. The position of the slider changes when the window is shifted left or right. The height of the slider in a vertical scroll bar represents the height of the window.

If you do not want VAXTPU to adjust the scroll bar slider automatically or if you want to change the size or position of the slider, specify the OFF keyword. For more information about calculating the size and position of the slider, see the description of the SET (SCROLL_BAR) built-in.

Note that you cannot disable VAXTPU's automatic adjustment of the scroll bar itself. VAXTPU always adjusts the scroll bar to the size of the window.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (SCROLL_BAR_AUTO_THUMB) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the SET (SCROLL_BAR_AUTO_THUMB) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (SCROLL_BAR_AUTO_THUMB) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (SCROLL_BAR_AUTO_THUMB) built-in.

EXAMPLE

```
vertical_bar := SET (SCROLL_BAR_AUTO_THUMB, CURRENT_WINDOW, VERTICAL, ON);
```

This statement turns on automatic adjustment of the vertical scroll bar's slider in the current window.

For sample code using the SET (SCROLL_BAR_AUTO_THUMB) built-in, see Example B–7.

SET (SCROLLING)

FORMAT

SET *(SCROLLING, window,* { *ON* / *OFF* }*, integer1, integer2, integer3)*

PARAMETERS

SCROLLING

This keyword refers to the upward or downward movement of existing lines in a window to make room for new lines at the bottom or top of the window. When a window is scrolled, the cursor position remains in the same column, but the screen line that the cursor is on may change.

window

The window in which the scrolling limits are being set.

ON

Causes scrolling of the text in a window to be turned on. This is the default value for the third parameter if the terminal supports scrolling.

OFF

Causes scrolling of the text in a window to be turned off. The screen is completely repainted each time a scroll would otherwise take place. This is the default value for the third parameter if the terminal does not support scrolling.

integer1

The offset from the top screen line of a window. The offset identifies the top limit of an area in which the cursor can move as it tracks the editing point. If the cursor is forced to move above this screen line to track the editing point, lines in the window move downward so that the cursor stays within the limits of the scroll margins. If you reach the beginning of the buffer, the text is no longer scrolled.

The value you specify for this parameter must be greater than or equal to zero and less than or equal to the number of lines in the window.

integer2

The offset from the bottom screen line of a window. The offset identifies the bottom limit of an area in which the cursor can move as it tracks the editing point. If the cursor is forced to move below this screen line to track the editing point, lines in the window move upward so that the cursor stays within the limits of the scroll margins. If you reach the end of the buffer, the text is no longer scrolled.

The value you specify for this parameter must be greater than or equal to zero and less than or equal to the number of lines in the window.

integer3

The number indicating how many lines from the top or the bottom scroll margin the cursor should be positioned after a window is scrolled. For example, if the bottom scroll margin is screen line 14 and *integer3* has a value of 0, the cursor is positioned on screen line 14 after text is scrolled upward. However, if *integer3* has a value of 3, the cursor is positioned on screen line 11.

The value you specify for this parameter must be greater than or equal to zero and less than or equal to the number of lines in the window.

You cannot specify a value that would position the cursor outside the window. That is, *integer1* + *integer3* or *integer2* + *integer3* must be less than the height of the window. For example, if the window is 10 lines long and *integer1* is set at 3, you cannot specify a value of 7 or more for *integer3*. Such a specification would place the cursor outside the window.

Note that if you use the SET (SCROLLING) built-in from within EVE by way of the TPU command, EVE may override the value you specify for this parameter.

DESCRIPTION

This built-in procedure is used to modify the scrolling action of a window.

If the terminal on which you are running VAXTPU supports scrolling, you can use the SET (SCROLLING) built-in to turn scrolling on or off. If the terminal does not support scrolling, scrolling will always be off. If scrolling is off, the window is repainted every time a scroll would otherwise occur.

The SET (SCROLLING) built-in also defines scroll margins using *integer1* and *integer2*. If the cursor is moved above the top scroll margin or below the bottom scroll margin using CURSOR_VERTICAL, MOVE_HORIZONTAL, MOVE_VERTICAL, POSITION, or a text manipulation built-in, then SET (SCROLLING) moves the cursor by the number of lines specified in *integer3*.

You must provide values for *integer1* and *integer2* that leave at least one line in the window unaffected by either scroll margin. That is, *integer1* + *integer2* must be less than the height of the window. For example, if you have a window that is ten lines tall, you cannot specify a value of 5 for the top scroll margin and a value of 5 for the bottom scroll margin. Such a specification leaves no area of the window that is not within a scroll margin.

You can move the cursor above or below a scroll margin under certain circumstances. If CROSS_WINDOW_BOUNDS is set to off, CURSOR_VERTICAL does not cause scrolling when the cursor reaches a scroll margin. If you are moving backward through the file and the top line of the buffer is already visible on the screen, the top scroll margin is ignored. If you are moving forward through the file and the bottom line of the buffer is already visible on the screen, the bottom scroll margin is ignored.

If using the ADJUST_WINDOW built-in makes the window so much smaller that the scroll margins overlap, VAXTPU automatically reduces the scroll margins proportionally to fit the new window. If you use ADJUST_WINDOW to make a window larger, VAXTPU does not adjust the scroll margins.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (SCROLLING) requires at least six parameters.
TPU$_TOOMANY	ERROR	You specified more than six parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters has the wrong type.
TPU$_UNKKEYWORD	ERROR	You have specified an unknown keyword.
TPU$_BADKEY	ERROR	Keyword must be either ON or OFF.
TPU$_BADMARGINS	ERROR	You have specified values for the top margin, bottom margin, and cursor movement that exceed the dimensions of the window.
TPU$_BADVALUE	ERROR	Integer values must be from 0 to 255.

EXAMPLES

1
```
SET (SCROLLING, new_window, ON, 0, 0, 2)
```

This statement turns on scrolling in the window *new_window*. The statement sets the top and bottom scroll margins to 0. This means that the cursor can be moved all the way to the top or bottom of the window before new text is scrolled into the window. Finally, the statement causes VAXTPU to place the cursor two lines down from the top or up from the bottom of the window when scrolling is completed.

2
```
SET (SCROLLING, new_window, ON, 0, 0, 20)
```

This statement demonstrates how to set scrolling if you want VAXTPU to present an entire window of new text each time a scroll occurs. If the variable *new_window* is 21 lines long, this statement causes VAXTPU to scroll all the text in the window off the top or bottom of the screen when you move the cursor to the top or bottom of the screen. This statement scrolls 20 new lines of text into the window.

Note that this statement does not produce a new window of text if you issue the statement from within EVE using the TPU command and move the cursor using the up arrow key or the down arrow key.

SET (SELF_INSERT)

FORMAT

SET *(SELF_INSERT, string,* { *ON* / *OFF* } *)*

PARAMETERS

SELF_INSERT

A keyword specifying whether a character is inserted into the buffer when the user presses a key with the following characteristics:

- Associated with a printable character
- Not bound to a procedure or program

string

A string specifying the key map list in which the behavior of undefined keys associated with printing characters is to be set.

ON

Causes the printable characters to be inserted when no procedures are bound to them, while the specified key map list is active. This is the default.

OFF

Causes the UNDEFINED_KEY procedure to be called when these characters are entered. If an undefined key procedure has not been specified, VAXTPU merely displays a warning message when the user presses an undefined, printable key. You can specify an undefined key procedure using the SET (UNDEFINED_KEY) built-in.

DESCRIPTION

SET (SELF_INSERT) lets you control what happens when the user presses an undefined key associated with a printable character. If SELF_INSERT is set ON and the user presses an undefined key associated with a printable character, the character is inserted into the current buffer at the current cursor position. If SELF_INSERT is turned off, printable characters whose keys are not defined in any key maps in the key map list bound to the current buffer are considered undefined. These undefined keys cause either the message "key has no definition" to be displayed, or some user-defined action to occur.

The default result for pressing an undefined key associated with a printable character procedure is that the character is inserted. The default condition for SET (SELF_INSERT) is ON. The default behavior, if SET (SELF_INSERT) is OFF, is to call the UNDEFINED_KEY procedure. See the description of the built-in procedure SET (UNDEFINED_KEY).

For more information on how to define what happens when SET (SELF_INSERT) is turned off, see the description of the built-in procedure SET (UNDEFINED_KEY) in this chapter.

SIGNALED ERRORS

TPU$_NOKEYMAPLIST	WARNING	You attempted to access an undefined key map list.
TPU$_TOOFEW	ERROR	SET (SELF_INSERT) requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
PROCEDURE toggle_self_insert

   LOCAL current_key_map_list;

   current_key_map_list := GET_INFO (CURRENT_BUFFER, "key_map_list");

   IF GET_INFO (current_key_map_list, "self_insert")
   THEN
       SET (SELF_INSERT, current_key_map_list, OFF)
   ELSE
       SET (SELF_INSERT, current_key_map_list, ON)
   ENDIF;
ENDPROCEDURE;
```

This procedure toggles the ON and OFF setting of SELF_INSERT for the key map list bound to the current buffer.

SET (SHIFT_KEY)

FORMAT

SET *(SHIFT_KEY, keyword [, string])*

PARAMETERS

SHIFT_KEY

This keyword refers to VAXTPU's shift key (by default PF1), not the key marked SHIFT on the keyboard.

keyword

A VAXTPU key name for a key.

string

A string that is a key map list name. This optional argument specifies the key map list in which the shift key is used. If the key map list is not specified, the key map list associated with the current buffer is used.

DESCRIPTION

The VAXTPU shift key is similar to the GOLD key in the EDT editor. This shift key allows you to assign two commands to one key: one is used when the key is pressed by itself, and the other is used when the key is pressed after the defined shift key.

Only one VAXTPU shift key can be active at a time. The VAXTPU shift key can be any key other than the following:

- The SHIFT key
- The ESCAPE key
- The SCROLL key on the VT100 keyboard
- The F1, F2, F3, F4, and F5 keys on the VT300 or VT200 keyboard
- The Compose Character key on the VT300 or VT200 keyboard

By default, PF1 is the VAXTPU shift key.

You cannot make VAXTPU execute a procedure or learn sequence bound to the shift key. However, designating a defined key as the shift key does not undefine the key; it merely disables the definition so long as the key is designated as the shift key. If you define another key as the shift key, VAXTPU reenables the first key's definition.

If you want to use PF1 for another purpose, use SET (SHIFT_KEY) to define a key other than PF1 as VAXTPU's shift key.

If you use SET (SHIFT_KEY) to define a GOLD key in EVE, EVE does not undefine the GOLD key correctly. When you use the EVE command SET NOGOLD or SET NOSHIFT, EVE returns the error message "There is no user GOLD key currently set." Although this message appears to say that the GOLD key has successfully been undefined, what it really means is that EVE does not recognize that a GOLD key was ever defined.

To redefine a GOLD key in these circumstances, you can use either of the following approaches:

- Use the EVE command SET GOLD KEY or SET SHIFT KEY.
- Undefine the GOLD key using the VAXTPU statement SET (SHIFT_KEY, KEY_NAME (PF1, SHIFT_KEY)). Then set the GOLD key using the SET GOLD KEY or SET SHIFT KEY command.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (SHIFT_KEY) requires at least two parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.
TPU$_NOKEYMAPLIST	WARNING	You specified an undefined key map list.

EXAMPLES

1
```
SET (SHIFT_KEY, PF4, "tpu$key_map_list")
```

This statement causes the keypad key PF4 to be defined as the shift key for the editor. The definition is stored in the default key map list, TPU$KEY_MAP_LIST. PF4 operates as the shift key only in buffers to which TPU$KEY_MAP_LIST is bound.

2
```
SET (SHIFT_KEY, KEY_NAME (PF1, SHIFT_KEY))
```

This statement disables the shift key by making the shift key itself a shifted key. Note that you can substitute the key name of whatever key is the SHIFT key. This technique works regardless of what key is defined as the SHIFT key. You might want to use such a statement if you are creating an editor that does not support user-defined shift key sequences.

SET (SPECIAL_ERROR_SYMBOL)

FORMAT

SET *(SPECIAL_ERROR_SYMBOL, string)*

PARAMETERS

SPECIAL_ERROR_SYMBOL

A keyword specifying that you want to use SET to designate a global variable to be set to 0 when a case-style error handler does not return from a CTRL/C or other error.

string

The name of the global variable that you want VAXTPU to set to 0.

DESCRIPTION

Once you designate the variable that is to be the special error symbol, VAXTPU sets the variable to 0 if any of the following events occurs:

- VAXTPU executes the TPU$_CONTROLC selector in a case-style error handler and does not encounter a RETURN statement
- VAXTPU executes the OTHERWISE clause in a case-style error handler and does not encounter a RETURN statement
- VAXTPU generates an error that is not handled by any clause in a case-style error handler

You can only use SET (SPECIAL_ERROR_SYMBOL) once in a program. This built-in is usually used during initialization. You must declare or create the variable before you use it in the SET statement. VAXTPU does not clear the variable in response to non-case-style error handlers.

The variable specified by SET (SPECIAL_ERROR_SYMBOL) can be used to determine whether VAXTPU has exited from current procedures and returned to the main loop to wait for a new keystroke.

SIGNALED ERRORS

TPU$_ERRSYMACTIVE	ERROR	A special error symbol has already been declared.
TPU$_TOOFEW	ERROR	SET (SPECIAL_ERROR_SYMBOL) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLE

```
SET (SPECIAL_ERROR_SYMBOL "back_to_main")
```

This statement designates the global variable *back_to_main* as the variable to be cleared if a procedure or program with a case-style error handler fails to handle a CTRL/C error or other error.

SET (STATUS_LINE)

FORMAT

SET *(STATUS_LINE, window,* { *NONE* | *BOLD* | *BLINK* | *REVERSE* | *SPECIAL_GRAPHICS* | *UNDERLINE* } *, string)*

PARAMETERS

STATUS_LINE
The last line in a window. You can use the status line to display regular text or you can use it to display status information about the window.

window
The window whose status line you want to modify.

NONE
Applies no video attributes to the characters on the status line.

BOLD
Causes the characters on the status line to be bolded.

BLINK
Causes the characters on the status line to blink.

REVERSE
Causes the characters on the status line to be displayed in reverse video.

SPECIAL_GRAPHICS
Causes the characters on the status line to display graphic characters, such as a solid line. These characters are from the DEC Special Graphics Set (also known as the VT100 Line Drawing Character Set). For more information on the special graphics that are available, see the appropriate programming manual for your video terminal.

UNDERLINE
Causes the characters on the status line to be underlined.

string
A quoted string, a variable name representing a string constant, or an expression that evaluates to a string, that is the text to be displayed on the status line. To remove a status line, use a null string ("") for this parameter.

DESCRIPTION

To have a status line, a window must be at least two lines high. You can establish a status line for a window when you create a window. CREATE_WINDOW requires you to specify whether the status line is ON (used for status information) or OFF (used as a regular text line). When you specify ON, the default status line is displayed in reverse video.

The algorithm for determining whether a window is tall enough to be given a status line depends on whether the window is visible or invisible.

If the window to which you want to add a status line is visible, VAXTPU checks the length of the visible portion of the window. A visible window can have an invisible portion if the window is partially occluded by another window. The visible portion of the visible window must have at least one text line; that is, at least one line not occupied by a scroll bar.

If the window is invisible, VAXTPU checks the full length of the window. The window must have at least one text line.

If the window that you use as a parameter for SET (STATUS_LINE) already has a status line, either because you specified ON for the status line parameter in the built-in procedure CREATE_WINDOW, or because you used a previous SET (STATUS_LINE) for the window, the video attribute that you specify is added to the video attribute of the existing status line unless you specify NONE. NONE overrides the other video keywords and specifies that there are to be no video attributes for the status line. The string you specify as the last parameter replaces the text of an existing status line. Adding a status line to a window that already has a status line does not cause an error.

If there is no status line for a window, the built-in procedure SET (STATUS_LINE) establishes a status line on the last visible screen line of the window. The status line has the video attribute and the text you specify. Adding a status line reduces the number of screen lines available for text by one line.

To remove a status line, use a null string ("") as the last parameter. The status line is removed even if the window is not two lines high at that time.

The default setting for the status line (ON or OFF) is determined by the built-in procedure CREATE_WINDOW.

If a window has a status line, by default the status line contains the name of the buffer associated with the window and the name of the file associated with the buffer, if there is one.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (STATUS_LINE) requires four parameters.
TPU$_TOOMANY	ERROR	You specified more than four parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	The keyword must be NONE, BOLD, BLINK, REVERSE, UNDERLINE, or SPECIAL_GRAPHICS.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.

TPU$_STATOOLONG	INFO	The status line is truncated to the screen width.
TPU$_BADWINDLEN	ERROR	The window must be at least two lines long.

EXAMPLES

1
```
SET (STATUS_LINE, my_window, REVERSE, "MAIN BUFFER, newfile.txt");
```

This statement displays the status line in *my_window* in reverse video with the buffer specified as MAIN BUFFER and the file specified as NEWFILE.TXT.

2
```
SET (STATUS_LINE, my_window, NONE, "");
```

This statement removes the status line in *my_window* by setting the final parameter to a null string.

3
```
line_text := "qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq" +
     "qqqqqqqqqqqqqqqqqqqq";
line_window := CREATE_WINDOW (1, 20, OFF);
MAP (line_window, current_buffer);
SET (STATUS_LINE, line_window, SPECIAL_GRAPHICS, line_text);
```

This code fragment creates a window with a status line displayed in special graphics rendition. Since the glyph (member of the DEC Multinational Character Set occupying one column width) having the same value as the character "q" is a full-width line, the status line appears as a solid line across the screen.

SET (SUCCESS)

FORMAT

SET *(SUCCESS,* { *ON* / *OFF* } *)*

PARAMETERS

SUCCESS

Controls whether VAXTPU writes success messages to the message buffer.

ON

Causes the success messages to be written.

OFF

Suppresses the display of success messages.

DESCRIPTION

By default, VAXTPU writes success messages to the message buffer. If you want to suppress the display of these messages, you can use this built-in procedure.

See Appendix D for a table of the VAXTPU messages and their severity levels.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (SUCCESS) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (SUCCESS, OFF)
```

This statement turns off the display of success messages.

SET (SYSTEM)

FORMAT

SET *(SYSTEM, buffer)*

PARAMETERS

SYSTEM

The status of a buffer. SYSTEM means that it is a system buffer rather than a user buffer.

By default, newly created buffers are user buffers.

buffer

The buffer that is being set as a system buffer.

DESCRIPTION

Once you make a buffer a system buffer, you cannot reset the buffer to be a user buffer.

The SET (SYSTEM) built-in procedure allows programmers who are building an editing interface to distinguish their system buffers from buffers that the user creates. VAXTPU does not handle system buffers differently from user buffers. Any distinction between the two kinds of buffers must be implemented by the application programmer.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (SYSTEM) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.

EXAMPLE

```
SET (SYSTEM, message_buffer)
```

This statement makes the message buffer a system buffer.

SET (TAB_STOPS)

FORMAT

SET *(TAB_STOPS, buffer,* { *integer* / *string* } *)*

PARAMETERS

TAB_STOPS

A keyword indicating that SET is to control placement of tab stops in a buffer.

buffer

The buffer in which the tab stops are being set.

integer

An integer specifying the interval between tab stops, measured in column widths. The minimum value for the integer is 1. The maximum value is 65,535.

string

A string of numbers that specifies the tab stops. The string represents column numbers at which the tab stops are placed. The minimum value for a tab stop is 1. The maximum value is 65,535. The maximum number of tab stops that you can include in the string is 100. The quoted string must list tab stops in ascending order, separating values with a single space: ("3 6 9 12.")

DESCRIPTION

When a buffer is created, the tabs are set at every eight columns, unless, when the buffer is created, a template buffer with different tab settings is specified.

The SET (TAB_STOPS) built-in enables you to set the tab stops at positions you specify or to establish equal intervals other than the default eight.

Tab stops are not saved when you write a file. When you create a buffer, the tabs are set to the default, unless, when you create the buffer, you specify a template buffer with different tab settings.

SET (TAB_STOPS) does not affect the hardware tab settings of your terminal. On any terminals or printers that have tab settings different from those you specify with this built-in, the file does not appear the same as it does when viewed using VAXTPU. In addition, if you invoke VAXTPU with the /NODISPLAY qualifier, any values you enter for SET (TAB_STOPS) are ignored, and a SHOW (BUFFER) command will return tabs every 0 columns.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (TAB_STOPS) requires at least three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.
TPU$_ARGMISMATCH	ERROR	The third parameter must be a string or an integer.
TPU$_INVTABSPEC	WARNING	You specified a bad third argument.

EXAMPLES

1
```
SET (TAB_STOPS, CURRENT_BUFFER, 4);
```

This statement causes the tab stops in the current buffer to be set at intervals of 4 columns.

2
```
SET (TAB_STOPS, CURRENT_BUFFER, "4 8 12 16");
```

This statement causes the tab stops in the current buffer to be set at 4, 8, 12, and 16 columns.

SET (TEXT)

FORMAT

SET *(TEXT,* { *widget, string* | *window,* { *BLANK_TABS* | *GRAPHIC_TABS* | *NO_TRANSLATE* } } *)*

PARAMETERS

TEXT

A keyword indicating that SET is to control the way text is displayed in a window or to determine the text that is to appear in a widget.

widget

The widget instance whose text you want to set. SET (TEXT, widget, string) is equivalent to the XUI Toolkit routine *S TEXT SET STRING*.

You can only use *widget* as the second parameter if you are using DECwindows VAXTPU.

string

The text you want to assign to the simple text widget.

window

The window in which the mode of display is being set.

BLANK_TABS

Displays tabs as blank spaces. This is the default keyword.

GRAPHIC_TABS

Displays tabs as special graphic characters so that the width of each tab is visible.

NO_TRANSLATE

Sends every keystroke from the keyboard to the terminal without any translation. In this mode, the terminal settings, not VAXTPU, determine the effect of characters typed from the keyboard.

Digital recommends that you use this mode for sending directives to the terminal but not for editing. VAXTPU does not manage margins or window shifts while NO_TRANSLATE mode is enabled. Furthermore, VAXTPU does not necessarily update lines of text in the order in which they appear while NO_TRANSLATE mode is enabled.

To send escape sequences from within a VAXTPU procedure, you can use SET (TEXT) with the NO_TRANSLATE keyword followed by statements using the MESSAGE and UPDATE built-ins. See the example in this built-in description for more information on this technique.

For more information on the effect of using various characters and sequences in NO_TRANSLATE mode, see your terminal manual.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	SET (TEXT) cannot return a value.
TPU$_REQUIRESDECW	ERROR	You have specified *widget* as the second parameter to SET (TEXT) while using non-DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (TEXT) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (TEXT) built-in.
TPU$_WIDMISMATCH	ERROR	The specified widget is not of class SText.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.

EXAMPLES

1
```
SET (TEXT, user_text_widget, "No default string available.");
```

Assuming that the variable *user_text_widget* has been assigned a text widget instance, this statement causes the widget to display the text *No default string available.*

2
```
wildcard_dialog_box := GET_INFO (WIDGET, "widget_id",
                                 eve$x_wildcard_find_dialog,
                                 "WILDCARD_FIND_DIALOG.WILDCARD_FIND_TEXT");
status := SET (TEXT, wildcard_dialog_box, eve$x_target);
```

These statements show one possible way that a layered application can use the SET (TEXT) widget. The variable *eve$x_target* stores the string (if one exists) that the user specified as the wildcard search string the last time the user invoked the wildcard find dialog box. The SET (TEXT) statement directs EVE's wildcard find dialog box widget to display the string assigned to *eve$x_target*.

3
```
SET (TEXT, CURRENT_WINDOW, GRAPHIC_TABS)
```

This statement causes the text in the main window to be displayed with special characters indicating tab characters.

```
4   ! If your terminal has a printer hooked up to the printer port,
    ! the following procedure allows you to perform a PRINT SCREEN
    ! function.

    PROCEDURE user_print

    ! Set window to NO_TRANSLATE to allow the escape sequence
    ! to pass to the printer. Note that this procedure does not send
    ! a form feed.

       SET (TEXT, message_window, NO_TRANSLATE);
       MESSAGE (ASCII (27) + "[i");
       UPDATE (message_window);

    ! Put back the window the way it was.

       SET (TEXT, message_window, BLANK_TABS);
       ERASE (message_buffer);
    ENDPROCEDURE;
```

This procedure uses the NO_TRANSLATE keyword. Notice that the window is set to this state temporarily, and that the default setting for the window is reset as soon as the function for which NO_TRANSLATE is used is finished executing.

SET (TIMER)

FORMAT

SET *(TIMER,* { *ON* / *OFF* } *[[, string]])*

PARAMETERS

TIMER

Controls attributes of messages displayed in the prompt area.

ON

Causes the message that you specify to be written to the prompt area and displayed at 1-second intervals. By default, the timed message is turned on.

OFF

Turns off the display of timed messages in the prompt area.

string

A quoted string, a variable name representing a string constant, or an expression that evaluates to a string, that is displayed in the prompt area. The maximum length of the message is 15 characters. If you specify a string longer than 15 characters, VAXTPU truncates the string but does not signal an error. The message is displayed in the last 15 character positions of the prompt area. If ON is specified and a string was never specified for the last argument, the timer puts out the message "working". If ON is specified and a string was specified previously, the saved string is used as the default.

DESCRIPTION

When SET (TIMER) is set to ON, the timer puts out messages at 1-second intervals while you are executing procedures or editing actions that are bound to a key. The message is written out to the prompt area and then erased to clear the prompt area for the next message.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (TIMER) requires at least two parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	The keyword must be ON or OFF.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.

EXAMPLE

```
SET (TIMER, ON, "Executing")
```

This statement causes the message "Executing" to be written to the prompt area at 1-second intervals while you are executing a VAXTPU procedure.

SET (TRACEBACK)

FORMAT

SET *(TRACEBACK,* { *ON* / *OFF* } *)*

PARAMETERS

TRACEBACK

Whether VAXTPU displays the sequence of procedures called after an error occurs.

ON

Causes VAXTPU to display the procedure calling sequence after an error occurs.

OFF

Prevents VAXTPU from displaying the procedure calling sequence after an error occurs.

DESCRIPTION

Traceback information provides the context in which an error occurs. Turning on the traceback setting can be helpful to a programmer debugging a VAXTPU program. The traceback setting is usually turned off during normal editing, because end users of editors do not usually use the traceback information.

The default setting for TRACEBACK depends on whether a section file was loaded by VAXTPU. If a section file was loaded, the default is OFF. If a section file was not loaded, the default is ON.

Note that SET (TRACEBACK) is related to SET (LINE_NUMBER). SET (TRACEBACK, ON) turns on both traceback and line numbers because both are needed for debugging. SET (LINE_NUMBER, OFF) turns off both traceback and line numbers because one feature is not useful without the other.

Allowable settings are as follows:

- Both off
- Both on
- Traceback off
- Line number on

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	The SET (TRACEBACK) built-in requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.

TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	WARNING	Only ON and OFF are allowed.

EXAMPLES

1
```
SET (TRACEBACK, OFF)
```

This statement prevents VAXTPU from displaying the procedure calling sequence after an error occurs.

2
```
PROCEDURE traceback_example
   SET (TRACEBACK, ON);
   SET (TRACEBACK, BELL);
   RETURN 5;
ENDPROCEDURE;

PROCEDURE call_example
    traceback_example;
ENDPROCEDURE;
```

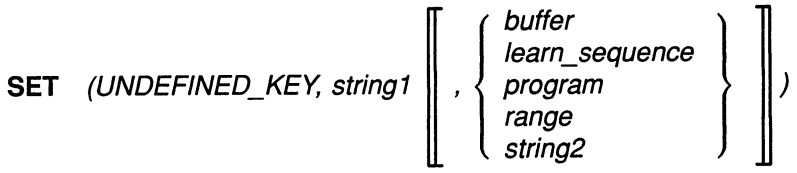
This procedure results in a traceback display when the procedure is executed and traceback is enabled.

Invoking the procedure CALL_EXAMPLE results in the following traceback:

```
BELL is an invalid keyword
Occurred in builtin SET
At line 2
Called from builtin EXECUTE
Called from line 22 of procedure EVE_TPU
Called from line 1
Called from builtin EXECUTE
Called from line 96 of procedure EVE$PROCESS_COMMAND
Called from line 3 of procedure EVE$PARSER_DISPATCH
Called from line 97 of procedure EVE$$EXIT_COMMAND_WINDOW
Called from line 2
```

SET (UNDEFINED_KEY)

FORMAT



PARAMETERS

UNDEFINED_KEY
A keyword specifying that SET is to determine the action taken when an undefined key is input.

string1
A string specifying the key map list for which this procedure is called.

buffer
The buffer containing VAXTPU statements specifying the action to be taken if the user presses an undefined key. SET (UNDEFINED_KEY) compiles the statements in the buffer and stores the resulting program in the specified key map list.

learn_sequence
The learn sequence specifying the action to be taken if the user presses an undefined key. The contents of a variable of type learn do not require compilation. SET (UNDEFINED_KEY) stores the learn sequence in the specified key map list.

program
The program specifying the action to be taken if the user presses an undefined key. The contents of a variable of type program do not require compilation. SET (UNDEFINED_KEY) stores the program in the specified key map list.

range
The range containing VAXTPU statements specifying the action to be taken if the user presses an undefined key. SET (UNDEFINED_KEY) compiles the statements in the range and stores the resulting program in the specified key map list.

string2
The string containing VAXTPU statements specifying the action to be taken if the user presses an undefined key. SET (UNDEFINED_KEY) compiles the statements in the string and stores the resulting program in the specified key map list.

DESCRIPTION

SET (UNDEFINED_KEY) determines the action taken when an undefined key is pressed.

If the third parameter is not specified, VAXTPU displays the message "key has no definition" when the user presses an undefined key.

SIGNALED ERRORS

TPU$_NOKEYMAPLIST	WARNING	You attempted to access an undefined key map list.
TPU$_TOOFEW	ERROR	SET (UNDEFINED_KEY) requires at least two parameters.
TPU$_TOOMANY	ERROR	SET (UNDEFINED_KEY) accepts no more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_ARGMISMATCH	ERROR	The second parameter must be a string.

EXAMPLE

```
IF GET_INFO ("tpu$key_map_list", "undefined_key") <> 0
THEN
    SET (UNDEFINED_KEY, "tpu$key_map_list");
ENDIF;
```

This code causes the default undefined key message to be displayed when an undefined key is entered.

SET (VIDEO)

FORMAT

SET *(VIDEO, window,* { *NONE* | *BOLD* | *BLINK* | *REVERSE* | *UNDERLINE* } *)*

PARAMETERS

VIDEO
The video attributes of a window.

window
The window in which a video attribute is being set.

NONE
Applies no video attributes to the characters in the window. This is the default.

BOLD
Causes the characters in the window to be bolded.

BLINK
Causes the characters in the window to blink.

REVERSE
Causes the characters in the window to be displayed in reverse video.

UNDERLINE
Causes the characters in the window to be underlined.

DESCRIPTION

Video attributes for a window are cumulative. The window assumes the video attribute of each video keyword that you use with SET (VIDEO) during an editing session. If you want to change the video attribute of a window, and you do not want the cumulative effect of previous attributes, use SET (VIDEO, window, NONE) before specifying the new attribute. SET (VIDEO, window, NONE) turns off all video attributes for a window.

The video attribute is applied during the next screen update. The screen manager repaints the window to apply the video attributes, even if the cumulative effect of your changes has been to leave the video attributes the same.

Note that the built-in procedure SET (VIDEO) does not affect the status line of a window. You can specify a video attribute for a status line either with CREATE_WINDOW or with the built-in procedure SET (STATUS_LINE). When the window and the status line have different video attributes, the status line can be used to separate multiple windows on the screen, or to highlight status information.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (VIDEO) requires three parameters.
TPU$_TOOMANY	ERROR	SET (VIDEO) accepts no more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADKEY	ERROR	You specified an invalid keyword.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.

EXAMPLE

```
SET (VIDEO, CURRENT_WINDOW, REVERSE);
SET (VIDEO, CURRENT_WINDOW, UNDERLINE);
```

These statements cause the current window to be displayed in reverse video and with underlining.

SET (WIDGET)

Allows you to assign values to various resources of a widget.

FORMAT

SET *(WIDGET, widget,*
{ widget_args [, widget_args...] })

PARAMETERS

WIDGET

A keyword directing VAXTPU to set an attribute of a widget.

widget

The widget instance whose values you want to set.

widget_args

One or more pairs of resource names and resource values. You can specify a pair in an array or as a pair of separate parameters. If you use an array, you index the array with a string that is the name of the resource you want to set. Note that resource names are case sensitive. The corresponding array element contains the value you want to assign to that resource. The array can contain any number of elements. If you use a pair of separate parameters, use the following format:

resource_name_string, resource_value

Arrays and string/value pairs may be interspersed. Each array index and its corresponding element value, or each string and its corresponding value, must be valid widget arguments for the class of widget whose resources you are setting.

DESCRIPTION

This built-in is functionally equivalent to the X Toolkit routine SET VALUES.

If you specify the name of a resource that the widget does not support, VAXTPU signals the error TPU$_ARGMISMATCH.

For more information about specifying resources, see Chapter 4.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_ARGMISMATCH	ERROR	You specified a value whose data type is not supported.
TPU$_NONAMES	WARNING	You specified an invalid widget resource name.
TPU$_NORETURNVALUE	ERROR	SET (WIDGET) cannot return a value.

TPU$_REQUIRESDECW	ERROR	You can use the SET (WIDGET) built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (WIDGET) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (WIDGET) built-in.
TPU$_WIDMISMATCH	ERROR	You have specified a widget whose class is not supported.

EXAMPLE

```
scroll_bar_widget := SET (SCROLL_BAR, CURRENT_WINDOW, VERTICAL, ON);
SET (WIDGET, scroll_bar_widget, eve$dwt$c_nvalue, 100);
```

These statements set the *Nvalue* resource of the current window's scroll bar widget to 100. This causes the scroll bar slider to be displayed as far toward the bottom of the scroll bar widget as possible.

For an example of a procedure using the SET (WIDGET) built-in, see Example B–8.

SET (WIDGET_CALL_DATA)

Allows you to create a template telling VAXTPU how to interpret the information in the fields of a widget's callback data structure.

FORMAT

SET *(WIDGET_CALL_DATA, widget, reason_code, request_string, keyword [, request_string, keyword...])*

PARAMETERS

WIDGET_CALL_DATA

A keyword indicating that the SET built-in is being used to control how VAXTPU interprets information in a widget's callback data structure.

widget

The specific widget instance for which you want to determine how the callback data are interpreted.

reason_code

The identifier for the reason code with which the callback data structure is associated. For example, if you are using SET (WIDGET_CALL_DATA) to set the format of the callback structure associated with the *Help Requested* reason code of the File Selection widget and if your program defines the VAX reason code bindings as constants, you could refer to the *Help Requested* reason code by using the constant DWT$C_CRHELP_REQUESTED.

request_string

One of the six valid strings describing the data type of a given field in a callback data structure. The valid strings are as follows:

"char"	"compound_string"
"int"	"short"
"void"	"widget"

keyword

One of the four valid keywords indicating the VAXTPU data type to which VAXTPU should convert the data in a given field of a callback data structure. The valid keywords are as follows:

INTEGER	STRING
UNSPECIFIED	WIDGET

Use the request_string parameter with the keyword parameter to inform VAXTPU, for each field of the structure, what data type the field had originally and what VAXTPU data type corresponds to the original data type. The valid keywords corresponding to each request string are as follows:

Request string	Associated keyword(s)
"widget"	WIDGET or UNSPECIFIED
"short"	INTEGER or UNSPECIFIED
"int"	INTEGER or UNSPECIFIED
"compound_string"	STRING or UNSPECIFIED
"char"	STRING or UNSPECIFIED
"void"	UNSPECIFIED

DESCRIPTION

You use SET (WIDGET_CALL_DATA) to tell VAXTPU what data type to assign to each field in a callback data structure. You must specify the widget and the callback reason whose data structure you want VAXTPU to process. During a callback generated by the specified widget for the specified reason, VAXTPU interprets the data in the callback structure according to the description you create.

In an application layered on VAXTPU, you can obtain the interpreted callback data by using the built-in GET_INFO (WIDGET, "callback_parameters").

You can create a different template for each of the reason codes associated with a given widget. To do so, make a separate call to the SET (WIDGET_CALL_DATA) built-in for each reason code. If you specify the same widget and reason code in more than one call, VAXTPU uses the most recently specified format.

In all callback data structures defined by the DECwindows Toolkit, the first field is the reason code field and the second field is the event field. For more information on the fields in each widget's callback structures, see the *VMS DECwindows Toolkit Routines Reference Manual*. If your application creates or uses a new kind of widget, the widget's callback structure must follow this convention.

Do not specify any request string or keyword for the reason field. In almost all cases, you specify the event field with the request string "void" and the keyword UNSPECIFIED. Specify all subsequent fields, if the callback structure has such fields, up to and including the last field you want to specify. Note that the VAX longword data type corresponds to the "int" request string and the INTEGER data type in VAXTPU.

Although you can skip trailing fields, you cannot skip intermediate fields even if they are unimportant to your application. To direct VAXTPU to ignore the information in a given field, use the request string "void" and the keyword UNSPECIFIED when specifying that field.

If you specify an invalid request string, VAXTPU signals TPU$_ILLREQUEST. If you specify an invalid keyword, VAXTPU signals TPU$_BADKEY. If you use valid parameters but assign the wrong data type to a field and if VAXTPU detects the error, VAXTPU assigns the data type UNSPECIFIED to that field during processing of a callback.

An application should use this built-in only if it needs access to callback information other than the reason code. For more information on how SET (WIDGET_CALL_DATA) affects GET_INFO (WIDGET, "callback_parameters"), see the online HELP topic GET_INFO(WIDGET).

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_NORETURNVALUE	ERROR	Built-in does not return a value.
TPU$_REQUIRESDECW	ERROR	You can use this built-in only if you are using DECwindows VAXTPU.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (WIDGET-CALL_DATA) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (WIDGET-CALL_DATA) built-in.

EXAMPLE

```
CONSTANT DWT$C_CRSINGLE := 20;

SET (WIDGET_CALL_DATA, initial_list_box, DWT$C_CRSINGLE,
     "void",             UNSPECIFIED,  ! event
     "compound_string", STRING,        ! item
     "int",              INTEGER,      ! item length
     "int",              INTEGER);     ! item number
```

This code fragment begins by defining the constant DWT$C_CRSINGLE to be the integer value 20, which is the integer associated with the reason "user selected a single item." Note that the file SYS$LIBRARY:DECW$DWTSTRUCT.H contains constants defined for reason code. If you layer an application, the values you assign to the reason code constants must match the values in this file. The next statement tells VAXTPU how to interpret the fields of the callback data structure associated with a List Box widget assigned to the variable "initial_list_box". The statement directs VAXTPU to ignore the data in the "event" field and to treat the data in the item field as type STRING, in the "item length" field as type INTEGER, and the "item number" field as type INTEGER.

SET (WIDGET_CALLBACK)

Specifies the VAXTPU program or learn sequence to be called by VAXTPU when a widget callback occurs for the widget instance.

FORMAT

SET *(WIDGET_CALLBACK, widget,* { *buffer* / *learn_sequence* / *program* / *range* / *string* } *, closure)*

PARAMETERS

WIDGET_CALLBACK

A keyword directing VAXTPU to set the application-level widget callback.

widget

The widget instance whose callback you want to set.

buffer

The buffer that contains the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

learn_sequence

The learn sequence that specifies the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

program

The program that specifies the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

range

The range that contains the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

string

The string that contains the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

closure

A string or integer. VAXTPU passes the value to the application when the widget performs a callback to VAXTPU. Note that DECwindows documentation refers to closures as *tags*. For more information about using closures, see Chapter 4.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by the SET (WIDGET_CALLBACK) built-in.
TPU$_BADDELETE	ERROR	You are attempting to modify an integer, a keyword, or a string constant.
TPU$_TOOFEW	ERROR	Too few arguments passed to the SET (WIDGET_CALLBACK) built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the SET (WIDGET_CALLBACK) built-in.
TPU$_COMPILEFAIL	WARNING	Program compilation has been terminated because of a syntax error.
TPU$_REQUIRESDECW	ERROR	You can use SET (WIDGET_ CALLBACK) only if you are using DECwindows VAXTPU.

EXAMPLE

```
SET (WIDGET_CALLBACK, scroll_bar_widget, "eve$scroll_dispatch", 'h');
```

This statement designates the EVE procedure EVE$SCROLL_DISPATCH as the callback routine for the widget *scroll_bar_widget* and assigns to the callback the closure value *'h'*.

For a procedure using this statement to map windows see Example B–7.

SET (WIDTH)

Sets the width of a window or the VAXTPU screen.

FORMAT

SET *(WIDTH,* { *window* | *ALL* | *SCREEN* }*, integer)*

PARAMETERS

WIDTH

A keyword indicating that the horizontal dimension is being set.

window

The window for which you want to set or change the width.

ALL

A keyword indicating that VAXTPU should set the screen and all windows, visible and invisible, to the specified width.

SCREEN

A keyword indicating that VAXTPU should set the screen to the specified width without altering the size of any VAXTPU windows. Note, however, that by default EVE resizes the windows to match the width of the screen. Note, too, that you cannot set the screen to be narrower than the widest VAXTPU window.

integer

The width of the window in columns. You can specify any integer between 1 and 255. In non-DECwindows VAXTPU, a value of 80 causes VAXTPU to repaint the screen and depict the text in normal-width font, if the text is not already so depicted. A value of 132 causes VAXTPU to repaint the screen and depict the text in narrow font, if the text is not already so depicted. Other values do not affect the font. By default, the width of a window is the same as the physical width of the terminal when the window is created.

DESCRIPTION

When you call SET (WIDTH), VAXTPU determines the width of the widest visible window. If this width has changed, the effect of SET (WIDTH) depends on your terminal.

If you are using VAXTPU with a VWS or DECwindows terminal emulator, the terminal emulator is resized to match the width of the widest visible window. You can specify any width between 1 column and 255 columns.

If you are using VAXTPU on a VT300-series, VT200-series, or VT100-series terminal, setting the width of a window only causes a change if the widest visible window is 80 or 132 columns wide. When the new width is one of these numbers, VAXTPU causes the terminal to switch from 80-column mode to 132-column mode, or the reverse.

If you are using DECwindows VAXTPU (that is, not in a DECterm window), changing the width of the screen does not affect the font of the characters displayed. (There are no 80-column or 132-column modes.)

If the width of the widest visible window has changed, VAXTPU redisplays all windows.

By default, the width of a window is the same as the number of columns on the screen of the terminal on which you are running VAXTPU. If you exceed the value set for the width of the window when entering text, VAXTPU displays a diamond symbol in the rightmost column of the screen to indicate that there is text beyond the diamond symbol that is not visible on the screen. You cannot force VAXTPU to use multiple lines to display a line that is longer than the width of a window.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SET (WIDTH) requires three parameters.
TPU$_TOOMANY	ERROR	You specified more than three parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_BADVALUE	ERROR	Arguments are out of minimum or maximum bounds.

EXAMPLES

1
```
SET (WIDTH, main_window, 132);
```

This statement sets the width of the main window to 132 columns and changes the font from standard to narrow.

2
```
SET (WIDTH, ALL, 40);
```

This statement sets the width of the screen and all windows, visible and invisible, to 40 columns. The statement does not affect the font.

SHIFT

For a buffer whose lines are too long to be displayed all at once, moves the window so the unseen parts of the lines can be viewed. SHIFT can move the window right to display text past the right edge of the window, or left (for a previously shifted window). SHIFT optionally returns an integer specifying the number of columns in the buffer lying to the left of the left edge of the shifted window.

FORMAT

[integer2 :=] SHIFT *(window, integer1)*

PARAMETERS

window

The window that is shifted.

integer1

The signed integer that specifies how many columns to shift the window. A positive integer causes the window to shift to the right so that you can see text that was previously beyond the right edge of the window.

A negative integer causes the window to shift to the left so that you can see text that was previously beyond the left edge of the window. If the first character in the line of text is already in column 1, then using a negative integer has no effect.

If you specify 0 as the value, no shift takes place. Furthermore, 0 as the value does *not* cause the window to be repainted.

By default, windows are not shifted.

return value

An integer representing the amount by which the window has been shifted to the right.

DESCRIPTION

Use the built-in procedure SHIFT when one or more lines of text in a buffer are too long to fit in the window (indicated by the diamond symbol in the rightmost column). By shifting the window from left to right, you can view text that was beyond the right edge of the window.

Because SHIFT commands are cumulative during an editing session, this built-in procedure optionally returns a value in *integer2*. This positive integer represents the absolute shift value.

The shift applies to any buffer associated with the window that you specify. For example, if you shift a window and then map another buffer to that window, you see the text of the newly mapped buffer in the shifted position. You must specify another shift to return the window to its unshifted position.

If you specify an integer value of 0, the window is not left-shifted. Furthermore, when you attempt to left-shift, the window is not repainted. Otherwise, SHIFT causes the entire window to be repainted. If you execute the built-in procedure SHIFT within a procedure, the screen is not updated to reflect the shift until the procedure has finished executing and control has returned to the screen manager. If you want the screen to reflect changes before the entire program is executed, you can force the immediate update of a window by adding an UPDATE statement to the procedure.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SHIFT requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLES

1
```
SHIFT (user_window, +5)
```

This statement shifts the window *user_window* five columns to the right.

2
```
SHIFT (CURRENT_WINDOW, -5)
```

This statement shifts the current window five columns to the left. (If the window was not previously shifted, this statement has no effect.)

3
```
SHIFT (CURRENT_WINDOW, -SHIFT (CURRENT_WINDOW, 0))
```

This statement always returns the current window to an unshifted state.

SHOW

Displays information about VAXTPU data types and the current settings of attributes that can be applied to certain data types. See also the description of the built-in procedure GET_INFO.

FORMAT

SHOW { *BUFFER[S]* | *KEY_MAP_LIST[S]* | *KEY_MAP[S]* | *KEYWORDS* | *PROCEDURES* | *SCREEN* | *SUMMARY* | *VARIABLES* | *WINDOW[S]* | *buffer* | *string* | *window* }

PARAMETERS

BUFFER[S]

Displays information about all buffers available to the editor. BUFFER is a synonym for BUFFERS.

KEY_MAP_LIST[S]

Displays the names of all defined key map lists, their key maps, and the number of keys defined in each key map. KEY_MAP_LIST is a synonym for KEY_MAP_LISTS.

KEY_MAP[S]

Displays the names of all defined key maps. KEY_MAP is a synonym for KEY_MAPS.

KEYWORDS

Displays the contents of the internal keyword table.

PROCEDURES

Displays the names of all defined procedures.

SCREEN

Displays information about the terminal.

SUMMARY

Displays statistics about VAXTPU, including the current version number.

VARIABLES

Displays the names of all defined variables.

WINDOW[S]

Displays information about all windows available to the editor. WINDOW is a synonym for WINDOWS.

buffer
Shows information about the buffer variable you specify.

string
Shows information about the string variable you specify.

window
Shows information about the window variable you specify.

DESCRIPTION

VAXTPU looks for the variable *show_buffer* and checks to see if it refers to a buffer. VAXTPU also looks for the variable *info_window* and checks to see if it refers to a window. If these two items exist when you call the built-in procedure SHOW, VAXTPU writes information to *show_buffer* and displays the information on the screen in the window called *info_window*.

You, or the interface you are using, must create the buffer variable *show_buffer* when you initialize the interface to ensure that the built-in procedure SHOW works as expected.

If you create a window called *info_window*, VAXTPU associates *show_buffer* with *info_window* and maps this window to the screen when there is information to be displayed. You can optionally create a different window in which to display the information from *show_buffer*. In this case, you must associate *show_buffer* with the window that you create and map the window to the screen when there is information to be displayed.

Because this built-in procedure maps INFO_WINDOW to the screen, any interfaces layered on VAXTPU should provide a mechanism for unmapping INFO_WINDOW and returning the user to the editing position that was current before the built-in procedure SHOW was invoked.

VAXTPU always deletes the current text in the show buffer before inserting the new information.

SIGNALED ERRORS

TPU$_NOSHOWBUF	WARNING	The requested information cannot be stored because the buffer variable *show_buffer* does not exist.
TPU$_TOOMANY	ERROR	SHOW accepts only one parameter.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

EXAMPLES

1 `SHOW (PROCEDURES)`

This statement displays on the screen a list of all the VAXTPU built-in procedures and the user-written procedures that are available to your editing interface.

2 SHOW (KEY_MAP_LISTS)

This statement displays the names of all defined key map lists, their key maps, and the number of keys defined in each key map. When you use the default interface, EVE, the VAXTPU command SHOW (KEY_MAP_LISTS) displays information similar to the following:

```
Defined key map lists:
    TPU$KEY_MAP_LIST contains the following key maps:
        EVE$USER_KEYS                   (0 keys defined)
        EVE$VT200_KEYS                  (14 keys defined)
        EVE$STANDARD_KEYS               (29 keys defined)
Total of 1 key map list defined
```

SLEEP

Causes VAXTPU to pause for the amount of time you specify or until input is available.

FORMAT

SLEEP ({ *integer* | *string* })

PARAMETERS

integer

The number of seconds to sleep.

string

An absolute or a delta time string indicating how long to sleep. See the documentation on the system service $BINTIM for the format of this string.

DESCRIPTION

This built-in suspends VAXTPU for the specified amount of time. This built-in is useful if you wish to display something for only a short period of time. SLEEP ends immediately when input becomes available from the terminal.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	SLEEP requires one argument.
TPU$_TOOMANY	ERROR	SLEEP accepts only one argument.
TPU$_ARGMISMATCH	ERROR	The argument to SLEEP must be an integer or string.
TPU$_INVTIME	ERROR	The argument to SLEEP was an invalid sleep time.

EXAMPLES

1
```
SLEEP (2);
```

This statement suspends VAXTPU for two seconds.

2
```
SLEEP ("0 0:0:1.50");
```

This statement suspends VAXTPU for one and one-half seconds.

```
PROCEDURE user_emphasize_message (user_message)
   LOCAL here,
         start_mark,
         the_range;

   here := MARK (NONE);

   POSITION (END_OF (message_buffer));
   COPY_TEXT (user_message);
   MOVE_HORIZONTAL (-CURRENT_OFFSET);
   start_mark := MARK (NONE);
   MOVE_VERTICAL (1);
   MOVE_HORIZONTAL (-1);

   the_range := CREATE_RANGE (start_mark, MARK (NONE), REVERSE);
   UPDATE (message_window);
   SLEEP ("0 00:00:00.33");
   the_range := 0;
   UPDATE (message_window);

   POSITION (here);
ENDPROCEDURE;
```

This procedure takes a string and puts it into the message buffer. The procedure displays the string in reverse video rendition for a third of a second. After a third of a second, the reverse video rendition is canceled and the string is displayed in ordinary rendition.

SPAN

Returns a pattern that matches a string of characters, each of which appears in the buffer, range, or string used as its parameter. SPAN matches as many characters as possible.

FORMAT

pattern := SPAN ({ *buffer* / *range* / *string* } [, { *FORWARD* / *REVERSE* }])

PARAMETERS

buffer

An expression that evaluates to a buffer. SPAN matches only those characters that appear in the buffer.

range

An expression that evaluates to a range. SPAN matches only those characters that appear in the range.

string

An expression that evaluates to a string. SPAN matches only those characters that appear in the string.

FORWARD

A keyword directing VAXTPU to match characters in the forward direction. This is the default.

REVERSE

A keyword directing VAXTPU to match characters as follows: first, match characters in the forward direction until VAXTPU finds a character that is not a member of the set of characters in the specified buffer, range, or string. Next, return to the first character matched and start matching characters in the reverse direction until VAXTPU finds a character that is not in the specified buffer, range, or string.

You can specify REVERSE only if you are using SPAN in the first element of a pattern being used in a reverse search. In all other contexts, specifying REVERSE has no effect.

The behavior enabled by REVERSE allows an alternate form of reverse search. By default, a reverse search stops as soon as a successful match occurs, even if there might have been a longer successful match in the reverse direction. By specifying REVERSE, you direct VAXTPU not to stop matching in either direction until it has matched as many characters as possible.

return value

A pattern that matches a sequence of characters, each of which appears in the buffer, range, or string used in the parameter to SPAN.

DESCRIPTION

SPAN matches one or more characters, each of which must appear in the string, buffer, or range passed as its parameter. SPAN matches as many characters as possible, stopping only if it finds a character not present in its parameter or if it reaches the end of a line. If SPAN is part of a larger pattern, SPAN does not match a character if doing so prevents the rest of the pattern from matching.

SPAN does not cross line boundaries. To match a string of characters that may cross one or more line boundaries, use SPANL.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	SPAN must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	SPAN requires at least one argument.
TPU$_TOOMANY	ERROR	SPAN accepts no more than one argument.
TPU$_ARGMISMATCH	ERROR	Argument passed to SPAN is of the wrong type.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of SPAN.

EXAMPLES

1
```
pat1 := SPAN ("0123456789")
```

This assignment statement creates a pattern that matches any sequence of numbers.

2
```
pat1 := span ("abcdefghijklmnopqrstuvwxyz") + "s";
```

This assignment statement creates a pattern that matches any word of two or more letters ending in the letter *s*. Given the word *dogs*, the SPAN part of the pattern matches *dog*. It does not match the *s* as well as this would prevent the rest of the pattern from matching.

3
```
PROCEDURE user_remove_xyz
   LOCAL pat1,
         xyz_line;

   pat1 := LINE_BEGIN + SPAN ("xyz") + LINE_END;

   LOOP
      xyz_line := SEARCH_QUIETLY (pat1, FORWARD);
      EXITIF xyz_line = 0;
      POSITION (xyz_line);
      ERASE_LINE;
   ENDLOOP;
ENDPROCEDURE;
```

This procedure removes all lines that contain only the letters *x*, *y*, and *z*.

SPANL

Returns a pattern that matches a string of characters and line breaks, each of which appears in the buffer, range, or string used as its parameter. The pattern matches as many characters and line breaks as possible.

FORMAT

pattern := SPANL ({ *buffer* / *range* / *string* } [, { *FORWARD* / *REVERSE* }])

PARAMETER

buffer

An expression that evaluates to a buffer. SPANL matches only those characters that appear in the buffer.

range

An expression that evaluates to a range. SPANL matches only those characters that appear in the range.

string

An expression that evaluates to a string. SPANL matches only those characters that appear in the string.

FORWARD

A keyword directing VAXTPU to match characters in the forward direction. This is the default.

REVERSE

A keyword directing VAXTPU to match characters as follows: first, match characters in the forward direction until VAXTPU finds a character that is not a member of the set of characters in the specified buffer, range, or string. Next, return to the first character matched and start matching characters in the reverse direction until VAXTPU finds a character that is not in the specified buffer, range, or string.

You can specify REVERSE only if you are using SPANL in the first element of a pattern being used in a reverse search. In all other contexts, specifying REVERSE has no effect.

The behavior enabled by REVERSE allows an alternate form of reverse search. By default, a reverse search stops as soon as a successful match occurs, even if there might have been a longer successful match in the reverse direction. By specifying REVERSE, you direct VAXTPU not to stop matching in either direction until it has matched as many characters as possible.

return value

A pattern matching a sequence of characters and line breaks.

DESCRIPTION

SPANL is similar to SPAN in that it matches one or more characters, each of which must appear in the string, buffer, or range used as a parameter. However, unlike SPAN, SPANL does not stop matching when it reaches the end of a line. It successfully matches the end of the line and continues trying to match characters on the next line. If SPANL is part of a larger pattern, it does not match a character or line boundary if doing so prevents the rest of the pattern from matching.

Normally, SPANL must match at least one character. However, if the argument to SPANL contains no characters, then SPANL matches one or more line breaks.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	SPANL must appear in the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	SPANL requires at least one argument.
TPU$_TOOMANY	ERROR	SPANL accepts no more than one arguments.
TPU$_ARGMISMATCH	ERROR	Argument passed to SPANL is of the wrong type.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of SPANL.

EXAMPLES

1
```
pat1 := SPANL (" ")
```

This assignment statement stores a pattern in *pat1* that matches the longest sequence of blank characters starting at the editing point and continuing until the search encounters a nonmatching character or the end of the buffer, range, or string.

2
```
pat2 := SPANL ("0123456789")
```

This assignment statement stores in *pat2* a pattern that matches the longest sequence of digits starting at the editing point and continuing until the search encounters a nonmatching character or the beginning or end of the buffer, range, or string.

3
```
pat3 := SPANL ("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

This assignment statement stores in *pat3* a pattern that matches the longest sequence of the alphabetic characters listed in the parameter. If you use this pattern with the built-in procedure SEARCH, the search starts at the current character position and continues to an end-of-search condition. If you specify an EXACT search, the characters must be uppercase for a successful match.

4
```
PROCEDURE user_remove_numbers
   LOCAL pat1,
         number_region;

   pat1 := SPANL ("0123456789");

   POSITION (BEGINNING_OF (CURRENT_BUFFER));

   LOOP
      number_region := SEARCH_QUIETLY (pat1, FORWARD);
      EXITIF number_region = 0;
      ERASE (number_region);
      POSITION (number_region);
   ENDLOOP;
   POSITION (BEGINNING_OF (CURRENT_BUFFER));
ENDPROCEDURE;
```

This procedure removes all parts of a document that contain only numbers.

5
```
PROCEDURE user_remove_blank_lines
   LOCAL pat1,
         blank_lines;

   pat1 := LINE_END + (SPANL ("") @ blank_lines)
           + LINE_BEGIN;

   POSITION (BEGINNING_OF (CURRENT_BUFFER));

   LOOP
      blank_lines := 0;
      SEARCH_QUIETLY (pat1, FORWARD);
      EXITIF blank_lines = 0;
      ERASE (blank_lines);
      POSITION (blank_lines);
ENDLOOP;
POSITION (BEGINNING_OF (CURRENT_BUFFER));
ENDPROCEDURE;
```

This procedure removes all empty lines from the current buffer. A line that contains only spaces or tabs is not empty.

6
```
PROCEDURE user_find_mark_twain

   LOCAL pat1,
         mark_twain;

   pat1 := "Mark" + (SPANL (" " + ASCII(9)) | SPANL (""))
           + "Twain";

   mark_twain := SEARCH_QUIETLY (pat1, FORWARD, NOEXACT);
   IF mark_twain = 0
   THEN
       MESSAGE ("String not found");
   ELSE
       POSITION (mark_twain);
   ENDIF;
ENDPROCEDURE;
```

This procedure positions you to the next occurrence of the text *Mark Twain*, where *Mark* and *Twain* may be separated by any number of spaces, tabs, or line breaks.

SPAWN

Creates a subprocess running the command line interpreter.

FORMAT

SPAWN *[(string, [[ON | OFF]])]*

PARAMETERS

string

The command string that you want to be executed in the context of the subprocess that is created with SPAWN.

ON

A keyword indicating that control is to be returned to VAXTPU after the command has been executed. This is the default unless the value specified for the first parameter is the null string.

OFF

A keyword indicating that the user is to be prompted for additional operating system commands after the specified command has been executed. If the value specified for the first parameter is the null string, the default value for the second parameter is OFF.

DESCRIPTION

SPAWN suspends your VAXTPU process and spawns a VMS subprocess. This built-in procedure is especially useful for running programs and utilities that take control of the screen (these programs cannot be run in a subprocess created with the built-in procedure CREATE_PROCESS). See Chapter 2 for a list of restrictions for subprocesses.

If you are using DCL, you can return to your VAXTPU session after finishing in a subprocess by using either the DCL command ATTACH or the DCL command LOGOUT. If you use the DCL command ATTACH, the subprocess is available for future use. If you use the DCL command LOGOUT, the subprocess is deleted. When you return to the VAXTPU session, the screen is repainted.

If you specify a DCL command as the parameter for SPAWN, the command is executed after the subprocess is created. When the command completes, the subprocess terminates, and control is returned to the VAXTPU process. If you want to remain in DCL, add the keyword OFF as the second parameter.

SPAWN was designed to allow you to leave a VAXTPU session, do other work in a VMS subprocess, and return to the VAXTPU session that you interrupted. Subprocesses created with SPAWN give you direct access to the command line interpreter. These subprocesses are different from the subprocesses created with the built-in procedure CREATE_PROCESS. CREATE_PROCESS creates a subprocess within a VAXTPU session, and all of the output from the subprocess goes into a buffer.

SPAWN is not a valid built-in in DECwindows VAXTPU. However, if you are running non-DECwindows VAXTPU in a DECwindows terminal emulator, SPAWN works as described in this section.

Note that SPAWN fails if you are running in an account with the CAPTIVE flag set in the authorization file.

See the description of the built-in procedure ATTACH in this section for information on moving control from one subprocess to another. See the *VMS DCL Dictionary* for more information on the characteristics of a spawned subprocess.

If the current buffer is mapped to a visible window, the SPAWN built-in causes the screen manager to synchronize the editing point, which is a buffer location, with the cursor position, which is a window location. This may result in the insertion of padding spaces or lines into the buffer if the cursor position is before the beginning of a line, in the middle of a tab, beyond the end of a line, or after the last line in the buffer.

SIGNALED ERRORS

TPU$_TOOMANY	ERROR	Too many arguments passed to the SPAWN built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the SPAWN built-in.
TPU$_REQUIRESTERM	ERROR	SPAWN is not a valid built-in in DECwindows VAXTPU.
TPU$_UNKKEYWORD	ERROR	An unknown keyword has been used as an argument. Only ON or OFF is allowed.
TPU$_BADKEY	ERROR	An unknown keyword has been used as an argument. Only ON or OFF is allowed.
TPU$_CAPTIVE	WARNING	Unable to create a subprocess in a captive account.
TPU$_CREATEFAIL	WARNING	Unable to activate the subprocess.

EXAMPLES

1
```
SPAWN
```

This spawns a VMS subprocess and suspends VAXTPU process. After completing work in the subprocess, you can return to your VAXTPU session by using the DCL command ATTACH or the DCL command LOGOUT.

2
```
SPAWN ("DIRECTORY")
```

This spawns a VMS subprocess and executes the DCL command DIRECTORY. When the command completes, you are returned to your VAXTPU session.

```
3   SPAWN ("SHOW LOGICAL SYS$LOGIN", OFF)
```

This spawns a VMS subprocess and puts your VAXTPU process on hold. The DCL command is executed in the subprocess to show the translation of the logical name SYS$LOGIN, and you are left at the DCL prompt. After completing work in the subprocess, you can return to your VAXTPU session by using the DCL command ATTACH or the DCL command LOGOUT.

SPLIT_LINE

Breaks the current line before the editing point and creates two lines.

FORMAT

SPLIT_LINE

PARAMETERS

None.

DESCRIPTION

SPLIT_LINE breaks the current line into two lines. The relative screen position of the line you are splitting may change as a result of this procedure. The first line contains any characters to the left of the editing point. The second line contains the rest of the characters. The new line that is created is inserted directly after the former current line.

When you use SPLIT_LINE, the editing point remains on the same character, but that character is now the first character on the newly created line.

If the editing point is not the first character in the line being split, the left margin of the old line is not changed. The new line, which contains the editing point and the characters to the right of the editing point, takes the buffer's left margin as its own left margin.

If the editing point is the first character of a line, SPLIT_LINE creates a blank line where the original line was. The left margin of this blank line is the buffer's left margin. SPLIT_LINE moves the original line, including the editing point, to the line below the blank line. If the original line had a left margin different from the buffer's current left margin, SPLIT_LINE preserves the original line's left margin when it moves the line down.

If the editing point is on a blank line, SPLIT_LINE creates a new blank line below the existing line. The editing point moves to the new blank line. The new blank line receives the buffer's left margin value. If the original blank line had a left margin different from the buffer's current left margin, the original blank line retains its margin.

Using SPLIT_LINE may cause VAXTPU to insert padding spaces or blank lines in the buffer. SPLIT_LINE causes the screen manager to place the editing point at the cursor position if the current buffer is mapped to a visible window. (For more information on the distinction between the cursor position and the editing point, see Chapter 6.) If the cursor is not located on a character (that is, if the cursor is before the beginning of a line, beyond the end of a line, in the middle of a tab, or below the end of the buffer), VAXTPU inserts padding spaces or blank lines into the buffer to fill the space between the cursor position and the nearest text.

SIGNALED ERRORS

TPU$_NOCURRENTBUF	WARNING	You are not positioned in a buffer.
TPU$_NOCACHE	ERROR	There is not enough memory to allocate a new cache.
TPU$_NOTMODIFIABLE	WARNING	You cannot modify an unmodifiable buffer.
TPU$_TOOMANY	ERROR	SPLIT_LINE takes no arguments.

EXAMPLES

1
```
SPLIT_LINE
```

This statement breaks the current line at the editing point and creates a new line.

2
```
PROCEDURE user_split_line

   LOCAL old_position,
         new_position;
   SPLIT_LINE;
   IF (CURRENT_ROW = 1) AND (CURRENT_COLUMN = 1)
   THEN
       old_position := MARK (NONE);
       SCROLL (CURRENT_WINDOW, -1);
       new_position := MARK (NONE);
       !Make sure we scrolled before doing CURSOR_VERTICAL
         IF new_position <> old_position
         THEN
             CURSOR_VERTICAL (1);
         ENDIF;
   ENDIF;
ENDPROCEDURE;
```

This procedure splits a line at the editing point. If the editing point is row 1, column 1, the procedure causes the screen to scroll.

STR

Returns a string equivalent for an integer, a keyword, a string, or the contents of a range or buffer.

FORMAT

```
string3 := STR ( { integer1 [ ,integer2 ] } )
                 { keyword                }
```

FORMAT

```
string3 := STR

( { { buffer } [ [, string2 ] [[ , ON ]] ] } )
  { { range  }                [[ , OFF ]]
  { string1                                }
```

PARAMETERS

integer1

The integer you want converted to a string.

integer2

The radix (base) you want VAXTPU to use when converting the first integer parameter to a string. The default radix is 10. The other allowable values are 8 and 16.

keyword

The keyword whose string representation you want.

buffer

The buffer whose contents you want returned as a string.

range

The range whose contents you want returned as a string.

string1

Any string. STR now accepts a parameter of type string, so you need not check the type of the parameter you supply to the built-in.

string2

A string specifying how you want line ends represented. The default is the null string. You can only use *string2* if you specify a range or buffer as the first parameter. If you want to specify the keyword ON or OFF but do not want to specify *string2*, you must use a comma before the keyword as a placeholder, as follows:

```
new_string := STR (old_buffer, , ON);
```

ON

A keyword directing VAXTPU to insert spaces preserving the white space created by the left margin of each record in the specified buffer or range. Specifically, if you specify a buffer or range with a left margin greater than

1, the keyword ON directs VAXTPU to insert a corresponding number of spaces after the line ends in the resulting string. For example, if the left margin of the specified lines is 10 and you use the keyword ON, VAXTPU inserts 9 spaces after each line end in the resulting string. VAXTPU does not insert any spaces after line beginnings of lines that do not contain characters. If the first line of a buffer or range starts at the left margin, VAXTPU inserts spaces before the text in the first line.

Note that you can only use this keyword if you specify a buffer or range as a parameter.

OFF

A keyword directing VAXTPU to ignore the left margin setting of the records in the specified buffer or range. This is the default. For example, if the left margin of the specified lines is 10 and you use the keyword OFF, VAXTPU does not insert any spaces after the line ends in the resulting string.

Note that you can only use this keyword if you specify a buffer or range as a parameter.

return value

string3	The string equivalent of the parameter you specify.

DESCRIPTION

If you use the first format shown above, STR returns a string representation of an integer or a keyword. You can then use the variable containing the returned string in operations that require string data types. For another method of generating a string representation of an integer, see the description of the built-in procedure FAO.

If you use the second format shown above, STR returns a string equivalent for any string or for the contents of a range or buffer.

SIGNALED ERRORS

TPU$_TRUNCATE	WARNING	You specified a buffer or range so large that converting it would exceed the maximum length for a string. VAXTPU has truncated characters from the returned string.
TPU$_NEEDTOASSIGN	ERROR	STR must appear on the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	STR requires at least one argument.
TPU$_TOOMANY	ERROR	STR accepts only two arguments.
TPU$_INVPARAM	ERROR	The argument to STR must be an integer, buffer, string, or range.
TPU$_BADVALUE	ERROR	You specified a value other than 8, 10, or 16 for the radix parameter.

EXAMPLES

1
```
return_string := STR (SELECT_RANGE, "<CRLF>", ON);
```

This statement creates a string using the text in the select range. Line breaks are marked with the string *CRLF*. The white space created by the margin is preserved by inserting spaces after the line breaks.

2
```
still_a_string := STR ("confetti");
```

This statement assigns the string *confetti* to the variable *still_a_string*.

3
```
new_numbers := STR (123)
```

This assignment statement stores the string "123" in the variable *new_numbers*.

4
```
the_string := STR (32, 16)
```

This assignment statement assigns the string "00000020" to the variable *the_string*.

5
```
the_string := STR (32, 10)
```

This assignment statement assigns the string "32" to the variable *the_string*.

6
```
PROCEDURE user_display_position

   v1 := GET_INFO (second_window, "current_column");
   MESSAGE ("Column: " + STR (v1));

   v2 := GET_INFO (second_window, "current_row");
   MESSAGE ("Row: " + STR (v2));

ENDPROCEDURE;
```

This procedure uses the built-in procedure STR to convert the integer variables *v1* and *v2* to strings so that your row and column position can be displayed in the message area.

7
```
this_string := STR (this_range, "EOL")
```

This statement forms a string using the text in the range "this_range." In the string, each end-of-line is represented by the letters EOL. For example, suppose the text in "this_range" is as follows:

```
Sufficient unto the day
are the cares thereof
```

Given this text in "this_range", "this_string" contains the following:

```
Sufficient unto the dayEOLare the cares thereof
```

If "this_range" extends to the character after the "f" in "thereof", "this_string" contains the following:

```
Sufficient unto the dayEOLare the cares thereofEOL
```

SUBSTR

Returns a string that represents a substring of a buffer, range, or string.

FORMAT

string2 := SUBSTR ({ *buffer* / *range* / *string* }, *integer1* [, *integer2*])

PARAMETERS

buffer
The buffer that contains the substring.

range
The range that contains the substring.

string
The string that contains the substring.

integer1
The character position at which the substring starts. The first character position is 1.

integer2
The number of characters to include in the substring. If you do not specify this parameter, VAXTPU sets the returned string's end point to the end of the first parameter.

return value

A string representing a substring of a string or range.

DESCRIPTION

If you specify a larger number of characters for *integer2* than are present in the substring, only the characters present are returned in *string2*. No error is signaled.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	SUBSTR must appear on the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	SUBSTR requires three arguments.
TPU$_TOOMANY	ERROR	SUBSTR accepts only three arguments.
TPU$_INVPARAM	ERROR	One of the arguments to SUBSTR is of the wrong type.
TPU$_ARGMISMATCH	ERROR	One of the arguments to SUBSTR is of the wrong type.

TPU$_TRUNCATE	WARNING	You specified a buffer or range so large that returning the requested substring would exceed the maximum length for a string. VAXTPU has truncated characters from the returned string.

EXAMPLES

1
```
file_type := SUBSTR ("login.com", 7, 3)
```

This assignment statement returns the string "com" in the variable *file_type*. The substring starts at the seventh character position ("c") and contains three characters ("com"). If you use a larger number for *integer2*, for example, *7*, the variable *file_type* still contains "com" and no error is signaled.

2
```
! Capitalize the first letter in a string.
!
PROCEDURE user_initial_cap (my_string)

LOCAL
    first_part_of_string,
    rest_of_string,
    first_letter,
    cur_loc;

cur_loc := 1;
first_part_of_string := "";
rest_of_string := "";

LOOP
    first_letter := SUBSTR (my_string, cur_loc, 1);
    EXITIF first_letter = "";
    EXITIF (first_letter >= "a") AND (first_letter <= "z");
    EXITIF (first_letter >= "A") AND (first_letter <= "Z");
    cur_loc := cur_loc + 1;
ENDLOOP;

CHANGE_CASE (first_letter, UPPER);
first_part_of_string := SUBSTR (my_string, 1, cur_loc - 1);
rest_of_string := SUBSTR (my_string, cur_loc + 1,
                          LENGTH (my_string) - cur_loc);

my_string := first_part_of_string + first_letter
             + rest_of_string;

ENDPROCEDURE;
```

This procedure capitalizes the first character in a string. It does not affect any other characters in the string. It makes use of the fact that SUBSTR returns a null string if the second parameter points past the end of the string.

```
3   first_ten_characters := SUBSTR (CURRENT_BUFFER, 1, 10);
    buffer_range := CREATE_RANGE (BUFFER_BEGIN, BUFFER_END, NONE);
    same_ten_characters := SUBSTR (r1, 1, 10);
    !
    ! Leaving the last parameter off means "go to the end".
    !
    a_string := "abcdefghijk";
    IF SUBSTR (a_string, 5, length (a_string)) <> SUBSTR (a_string, 5)
    THEN
        MESSAGE ('This message will never be displayed.');
    ELSE
        MESSAGE ('This message is always displayed.');
    ENDIF;
```

These two calls to SUBSTR return the same value.

TRANSLATE

Substitutes one set of specified characters for another set. TRANSLATE returns a value for the translated range or buffer or for the string representation of the translated text. TRANSLATE is based on the Run-Time Library (RTL) routine STR$TRANSLATE. For complete information on STR$TRANSLATE, see the *VMS RTL String Manipulation (STR$) Manual.*

FORMAT

$$\left\{\begin{array}{l}\textbf{buffer1}\\ \textbf{range1}\\ \textbf{string1}\end{array}\right\} := \textbf{TRANSLATE}\ \left(\left\{\begin{array}{l}\textit{buffer2}\\ \textit{range2}\\ \textit{string2}\end{array}\right\}, \textit{string3}, \textit{string4}\ \left[\!\left[, \left\{\begin{array}{l}\textit{IN\_PLACE}\\ \textit{NOT\_IN\_PLACE}\end{array}\right\}\right]\!\right]\right)$$

PARAMETERS

buffer2

A buffer in which one or more characters are to be replaced. Note that you cannot use the keyword NOT_IN_PLACE if you specify a buffer for the first parameter.

range2

A range in which one or more characters are to be replaced. Note that you cannot use the keyword NOT_IN_PLACE if you specify a range for the first parameter.

string2

A string in which one or more characters are to be replaced. If a return value is specified, the substitution is performed in the returned string. If you specify IN_PLACE for the third parameter, TRANSLATE makes the specified change to the string specified in the first parameter. Note that if *string2* is a constant, IN_PLACE has no effect.

string3

The string of replacement characters.

string4

The literal characters within the text specified by parameter1 that are to be replaced.

IN_PLACE

A keyword directing VAXTPU to make the indicated change in the buffer, range, or string specified. This is the default.

NOT_IN_PLACE

A keyword directing VAXTPU to leave the specified string unchanged and return a string that is the result of the specified translation. You cannot use NOT_IN_PLACE if the first parameter is specified as a range or buffer. To use NOT_IN_PLACE, you must specify a return value for TRANSLATE.

return values

buffer1
A variable of type buffer pointing to the buffer containing the modified text, if you specify a buffer for the first parameter. The variable "returned_buffer" points to the same buffer pointed to by the buffer variable specified as the first parameter.

range1
A range containing the modified text, if you specify a range for first parameter. The returned range spans the same text as the range specified as a parameter, but they are two separate ranges. If you subsequently change or delete one of the ranges, this has no effect on the other range.

string1
A string containing the modified text, when you specify a string for the first parameter. TRANSLATE can return a string even if you specify IN_PLACE.

DESCRIPTION

The TRANSLATE built-in searches the text specified by the first parameter for the characters contained in the third parameter. When VAXTPU finds the sequence specified by *string3*, VAXTPU substitutes the first character in *string2* for the first character in *string3*, and so forth.

If the translate string, *string2*, is shorter than the match string, *string3*, and the number of matched character positions is greater than the number of character positions in the translate string, the translation character is a space.

The IN_PLACE and NOT_IN_PLACE keywords specify whether the source is to be changed. IN_PLACE means that the source is modified, while NOT_IN_PLACE indicates that the source is not changed.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	TRANSLATE requires three arguments.
TPU$_TOOMANY	ERROR	TRANSLATE accepts no more than three arguments.
TPU$_ARGMISMATCH	ERROR	One of your arguments to TRANSLATE is of the wrong data type.
TPU$_INVPARAM	ERROR	One of your arguments to TRANSLATE is of the wrong data type.
TPU$_NOTMODIFIABLE	WARNING	You cannot translate text in an unmodifiable buffer.
TPU$_CONTROLC	ERROR	You pressed CTRL/C during the execution of TRANSLATE.

EXAMPLES

1
```
TRANSLATE (second_buffer, "I","i")
```

This statement replaces any lowercase "i" in *second_buffer* with an uppercase "I".

2
```
! Procedure to translate characters to decipher scrambled text.
! Characters are shifted 13 places for encryption.

PROCEDURE user_trans_text ( text_to_translate )

TRANSLATE (text_to_translate,
   "NOPQRSTUVWXYZABCDEFGHIJKLMnopqrstuvwxyzabcdefghijklm",
   "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz");

ENDPROCEDURE;
```

This procedure translates the text you specify as "text_to_translate" according to the following pattern: any "A" is converted to an "N"; any "B" is converted to an "O"; and so on.

3
```
PROCEDURE user_strip_eighth

   LOCAL i,          ! Loop counter
         seven,      ! ASCII (0) through ASCII (127)
         eight;      ! ASCII (128) through ASCII (255)

! Build translate strings

   seven := "";
   eight := "";
   i := 0;
   LOOP
      seven := seven + ASCII (i);
      eight := eight + ASCII (i + 128);
      i := i + 1;
      EXITIF i = 128;
   ENDLOOP;

   TRANSLATE (CURRENT_BUFFER, seven, eight);

ENDPROCEDURE;
```

This procedure strips the eighth bit from all characters in the current buffer. A procedure like this is useful for reading files from systems like TOPS–20 on which the eighth bit is set without using the DEC Multinational Character Set.

4

The following statements show how the asterisk (*) character can replace the character r during an interactive session. Suppose the following text is written in a buffer and that the variable "the_range" spans this text:

```
This darned wind is a darned nuisance, darn it!
```

The following statement assigns to "the_string" the characters in "the_range":

```
the_string := STR (the_range)
```

The following statement assigns to *translated_string* the text that results when an asterisk is substituted for each "r":

```
translated_string := TRANSLATE (the_string, "*", "r", NOT_IN_PLACE)
```

The variable "translated_string" then contains the following text:

```
This da*ned wind is a da*ned nuisance, da*n it!
```

Note that if the text contained other r's, they would also be replaced by asterisks.

UNANCHOR

Specifies that the next pattern element may match anywhere after the previous pattern element.

FORMAT

UNANCHOR

PARAMETERS

None.

DESCRIPTION

Normally, when a pattern contains several concatenated or linked pattern elements, the pattern matches only when the text that matches one particular pattern element immediately follows the text that matches the previous pattern element. If UNANCHOR appears between two pattern elements, the text that matches the second pattern element may appear anywhere after the text that matches the first pattern element.

Although UNANCHOR behaves much like a built-in, it is actually a keyword.

For more information on patterns or pattern searching, see Chapter 2.

SIGNALED ERRORS

UNANCHOR is a keyword and has no completion codes.

EXAMPLES

1
```
pat1 := "a" + UNANCHOR + "123"
```

This assignment statement creates a pattern that matches any text beginning with the letter *a* and ending with the digits *123*. Any amount of text may appear between the *a* and the *123*.

2
```
pat1 := UNANCHOR + "a123";
```

This assignment statement creates a pattern that matches from the search start position (the current position if searching the current buffer) through to and including the first occurrence of the string *a123*.

```
3  PROCEDURE user_remove_paren_text (paren_buffer)
      LOCAL pat1,
            paren_text,
            searched_text;
      pat1 := "(" + UNANCHOR + ")";
      searched_text := paren_buffer;
      LOOP
         paren_text := SEARCH_QUIETLY (pat1, FORWARD, EXACT,
                                       searched_text);
         EXITIF paren_text = 0;
         ERASE (paren_text);
         searched_text := CREATE_RANGE (END_OF (paren_text),
                                        END_OF (paren_buffer), NONE);
    ENDLOOP;
   ENDPROCEDURE;
```

This procedure removes all parenthesized text from a buffer. The text may span several lines. It does not handle multiple levels of parentheses.

UNDEFINE_KEY

Removes the current binding from the key that you specify.

FORMAT

UNDEFINE_KEY *(keyword* [, { *key-map-list-name* | *key-map-name* }] *)*

PARAMETERS

keyword

The name of a key or key combination that VAXTPU allows you to define. See Table 2–1 for a list of the valid VAXTPU key names.

key-map-list-name

Specifies a key map list in which the key is defined. The first definition of the key in the key maps that make up the key map list is deleted. If neither a key map nor a key map list is specified, the key map list bound to the current buffer is used.

key-map-name

Specifies a key map in which the key is defined. The first definition of the key in the key map is deleted. If neither a key map nor a key map list is specified, the key map list bound to the current buffer is used.

DESCRIPTION

After you use UNDEFINE_KEY, the key you specify is no longer defined. VAXTPU does not save any previous definitions that you may have associated with the key. However, any definitions of the specified key in key maps or key map lists other than the ones you specified are not removed.

VAXTPU writes a message to the message buffer telling you that the key is undefined if you try to use it after you have undefined it.

SIGNALED ERRORS

TPU$_NODEFINITION	WARNING	There is no definition for this key.
TPU$_NOTDEFINABLE	WARNING	First argument is not a valid reference to a key.
TPU$_NOKEYMAP	WARNING	Second argument is not a defined key map.
TPU$_NOKEYMAPLIST	WARNING	Second argument is not a defined key map list.
TPU$_KEYMAPNTFND	WARNING	The key map listed in the second argument is not found.
TPU$_EMPTYKMLIST	WARNING	The key map list specified in the second argument contains no key maps.

TPU$_TOOFEW	ERROR	Too few arguments passed to the UNDEFINE_KEY built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the UNDEFINE_KEY built-in.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the UNDEFINE_KEY built-in.

EXAMPLES

1
```
UNDEFINE_KEY (CTRL_Z_KEY)
```

This statement removes the association between the key combination CTRL/Z and the code that it previously executed.

2
```
! Parameters:
!
!     Name          Function                    Input or Output?
!     ----          --------                    ----------------
!     which_key     Keyword for key to clear      input

PROCEDURE user_clear_key (which_key)
   IF (LOOKUP_KEY (which_key, PROGRAM) <> 0)
   THEN
       UNDEFINE_KEY (which_key);
   ELSE
       MESSAGE ("Key not defined");
   ENDIF;
ENDPROCEDURE;
```

This procedure undefines a key. A procedure like this can be used by keypad initialization procedures.

3
```
PROCEDURE delete_all_definitions
   LOCAL key;

   LOOP
      key := GET_INFO (DEFINED_KEY, "first", "tpu$key_map");
      EXITIF key = 0;
      UNDEFINE_KEY (key, "tpu$key_map");
   ENDLOOP;
ENDPROCEDURE;
```

This procedure deletes all of the key definitions in the key map TPU$KEY_MAP.

UNMANAGE_WIDGET

Makes the specified widget and all of its children invisible.

For more information about managing widgets, see the *VMS DECwindows Toolkit Routines Reference Manual.*

FORMAT

UNMANAGE_WIDGET *(widget [, widget...])*

PARAMETERS

widget
The widget instance to be unmanaged.

DESCRIPTION

If you want to unmanage several widgets that are children of the same parent, but you do not want to unmanage the parent, include all the children in a single call to UNMANAGE_WIDGET. Unmanaging several widgets at once is more efficient than unmanaging one widget at a time.

The UNMANAGE_WIDGET built-in is equivalent to the X Toolkit UNMANAGE CHILD and UNMANAGE CHILDREN routines.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	You specified a parameter of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to the UNMANAGE_WIDGET built-in.
TPU$_NORETURNVALUE	ERROR	UNMANAGE_WIDGET cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the UNMANAGE_WIDGET built-in only if you are using DECwindows VAXTPU.
TPU$_WIDMISMATCH	ERROR	You have specified a widget whose class is not supported.

EXAMPLE

```
PROCEDURE eve$$replace_clean_up

ON_ERROR
    [TPU$_CONTROLC]:
        eve$learn_abort;
        abort;
    [OTHERWISE]:
        eve$$replace_error_handler;
ENDON_ERROR;

IF NOT eve$$x_replace_array {eve$$k_replace_asking}

THEN        ! If all occurrences were replaced, the editing
            ! point is positioned to the last saved mark.
```

```
    POSITION (eve$$x_replace_array {eve$$k_replace_saved_mark});
ENDIF;

! Restore the buffer's original direction and mode.

SET (eve$$x_replace_array {eve$$k_replace_saved_direction},
     eve$$x_replace_array {eve$$k_replace_this_buffer});
SET (eve$$x_replace_array {eve$$k_replace_saved_mode},
     eve$$x_replace_array {eve$$k_replace_this_buffer});

SET (SCREEN_UPDATE, ON);
eve$message (EVE$_REPLCOUNT, 0,
             eve$$x_replace_array {eve$$k_replace_occurrences});

 IF (eve$$x_state_array {eve$$k_command_line_flag} = eve$k_invoked_by_menu)
    AND (eve$$x_state_array {eve$$k_dialog_box})
 THEN
    IF eve$x_decwindows_active
    THEN
        IF GET_INFO (eve$x_replace_each_dialog, "type") = WIDGET
        THEN

            UNMANAGE_WIDGET (eve$x_replace_each_dialog);    ! This statement
                                                            ! unmanages the
                                                            ! replace dialog
                                                            ! box.
        ENDIF;
    ENDIF;
 ENDIF;

ENDPROCEDURE;
```

This procedure shows one possible way that a layered application can use the UNMANAGE_WIDGET built-in. The procedure is a modified version of the EVE procedure EVE$$REPLACE_CLEAN_UP. You can find the original version in SYS$EXAMPLES:EVE$EDIT.TPU.

The procedure performs screen cleanup operations after the user has used the EVE command REPLACE. It restores the direction and mode to which the buffer was set before the replace operation began, then tests whether the replace dialog box is present and, if so, makes it invisible.

UNMAP

Disassociates a window from its buffer and removes the window or widget from the screen.

FORMAT

UNMAP ({ *window* | *widget* })

PARAMETERS

window

The window you want to remove from the screen.

widget

The widget instance you want to make invisible.

DESCRIPTION

If you unmap the current window, VAXTPU tries to move the cursor position to the window that was most recently the current window. The window in which VAXTPU positions the cursor becomes the current window, and the buffer that is associated with this window becomes the current buffer.

The screen area of the window you unmap is either erased or returned to any windows that were occluded by the window you unmapped. VAXTPU returns lines to adjacent windows if the size of the windows requires the lines that were used for the window you unmap. The size of a window is determined by the values you specified for the built-in procedure CREATE_WINDOW when you created the window, or by the values you specified for the built-in procedure ADJUST_WINDOW if you changed the size of the window. If adjacent windows do not require the lines that were used by the window you unmap, the lines that the window occupied on the screen remain blank.

The window that you unmap is not deleted from the list of available windows. You can cause the window to appear on the screen again with MAP. UNMAP does not have any effect on the buffer that was associated with the window being unmapped.

Note that unmapping a widget does not delete the widget. Future MAP operations will make the widget visible again.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	UNMAP requires one parameter.
TPU$_TOOMANY	ERROR	UNMAP accepts only one parameter.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.

TPU$_WINDNOTMAPPED	WARNING	Window is not mapped to a buffer.

EXAMPLES

1
```
UNMAP (main_window)
```

This statement removes the main window from the screen and disassociates from the main window the buffer that was mapped to it.

2
```
PROCEDURE user_one_window_to_two

   LOCAL wind_length,
         wind_half,
         first_line,
        last_line;

   cur_wind := CURRENT_WINDOW;

! If it exists
   IF (cur_wind <> 0)
   THEN
       first_line := GET_INFO (cur_wind, "visible_top");
       last_line := GET_INFO (cur_wind, "visible_bottom");
       wind_buf := GET_INFO (cur_wind, "buffer");
       UNMAP (cur_wind);
   ELSE
! If there is no current window then create an empty buffer

       first_line := 1;
       last_line := GET_INFO (SCREEN, "visible_length");
       wind_buf := CREATE_BUFFER ("Empty Buffer");
    ENDIF;

   wind_length := (last_line - first_line) + 1;
   wind_half := wind_length/2;
   new_window_1 := CREATE_WINDOW (first_line, wind_half, OFF);

   SET (VIDEO, new_window_1, UNDERLINE);
   new_window_2 := CREATE_WINDOW (wind_half+1,
                   last_line-wind_half, OFF);

! Associate the same buffer with both windows
! and map the windows to the screen

   MAP (new_window_1, wind_buf);
   MAP (new_window_2, wind_buf);
ENDPROCEDURE;
```

This procedure unmaps the current window and puts two new windows in its place. (Note that if the window that you are replacing has a status line, this line is not included in the screen area used by the two new windows. This is because GET_INFO (window, "visible_bottom") does not take the status line into account.)

3
```
UNMAP (example_widget);
```

This statement causes the widget assigned to the variable *example_widget* to become invisible.

UPDATE

Causes the screen manager to make a window reflect the current internal state of the buffer that is associated with the window. One important task that UPDATE performs is to move the cursor to the editing point if the cursor and the editing point are not synchronized when the UPDATE built-in is executed.

FORMAT

UPDATE ({ *ALL* / *window* })

PARAMETERS

ALL

A keyword directing VAXTPU to make all visible windows reflect the current state of the buffers mapped to them.

window

The window that you want updated. The window must be mapped to the screen for the update to occur.

DESCRIPTION

The screen manager updates windows after each keystroke. However, if a key has a procedure bound to it, VAXTPU may execute many statements when that key is pressed. By default, UPDATE does not reflect the result of any statement in a procedure bound to a key until all the statements in the procedure have been executed. As a result, the screen may not reflect the current state of the buffer during execution of a procedure bound to a key. If you want the screen to reflect changes before the entire procedure is executed, you can force an immediate update by adding an UPDATE statement to the procedure.

UPDATE (window) affects a single window that is visible on the screen. If the buffer associated with the window you use as a parameter is associated with other windows that are mapped to the screen, all of these windows may be updated.

UPDATE (ALL) updates all visible windows. The difference between the UPDATE (ALL) built-in and the REFRESH built-in is that UPDATE (ALL) makes whatever changes are necessary on a window-by-window basis. REFRESH clears the screen and repaints everything from scratch, as well as reinitializing scrolling regions and other terminal-dependent settings.

For more information on the results of the REFRESH built-in, see the description of REFRESH in this chapter. For more information on how the VAXTPU screen manager uses the UPDATE built-in in various circumstances, see Chapter 6.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	UPDATE requires one parameter.
TPU$_TOOMANY	ERROR	You specified more than one parameter.
TPU$_INVPCARAM	ERROR	The specified parameter has the wrong type.
TPU$_BADKEY	ERROR	The keyword must be ALL.
TPU$_UNKKEYWORD	ERROR	You specified an unknown keyword.
TPU$_WINDNOTMAPPED	WARNING	You cannot update a window that is not on the screen.

EXAMPLES

1
```
UPDATE (new_window)
```

This statement causes the screen manager to make *new_window* reflect the current internal state of the buffer associated with *new_window*.

2
```
PROCEDURE user_show_first_line

   LOCAL old_position,         ! Marker of position before scroll
         new_position;         ! Marker of position after scroll

   UPDATE (CURRENT_WINDOW);
   IF (GET_INFO (CURRENT_WINDOW, "current_row") =
      GET_INFO (CURRENT_WINDOW, "visible_top"))
      AND
         (CURRENT_COLUMN = 1)
   THEN
       old_position := MARK (NONE);
       SCROLL (CURRENT_WINDOW, -1);
       new_position := MARK (NONE);

! Make sure we scrolled before doing the CURSOR_VERTICAL

      IF new_position <> old_position
      THEN
          CURSOR_VERTICAL (1);
      ENDIF;
   ENDIF;
ENDPROCEDURE;
```

This procedure updates the screen to display the new line of text that you are inserting before the top line of the window. (When you insert text in front of the top of a window, the included text is not visible on the screen unless you use a procedure such as this one to ensure that the text is displayed.)

WRITE_CLIPBOARD

Writes string format data to the clipboard.

FORMAT

WRITE_CLIPBOARD *(clipboard_label,* { *buffer* | *range* | *string* } *)*

PARAMETERS

clipboard_label

The label for multiple entries in the clipboard. Since the clipboard does not currently support multiple labels, use any string, including the null string, to specify this parameter.

buffer

The buffer containing text to be written to the clipboard. VAXTPU represents line breaks by a line-feed character (ASCII (10)). If you specify a buffer, VAXTPU converts the buffer to a string, replacing line breaks with line feeds, and replacing the white space before the left margin with padding blanks.

The buffer must contain at least one character or line break. If it does not, VAXTPU signals TPU$_CLIPBOARDZERO.

range

The range containing text to be written to the clipboard. VAXTPU represents line breaks by a line-feed character (ASCII (10)). If you specify a range, VAXTPU converts the range to a string, replacing line breaks with line feeds, and replacing the white space before the left margin with padding blanks.

The range must contain at least one character or line break. If it does not, VAXTPU signals TPU$_CLIPBOARDZERO.

string

The string containing text to be written to the clipboard. The string must contain at least one character. If it does not, VAXTPU signals TPU$_CLIPBOARDZERO.

DESCRIPTION

The *clipboard_label* parameter provides support for multiple entries on the clipboard; at present, however, the clipboard does not support multiple entries.

SIGNALED ERRORS

TPU$_CLIPBOARDLOCKED	WARNING	The clipboard is locked by another process.
TPU$_CLIPBOARDZERO	WARNING	The data to be written to the clipboard have zero length.
TPU$_TRUNCATE	WARNING	VAXTPU has truncated characters from the data written because you specified a buffer or range containing more than 65,535 characters.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	WRITE_CLIPBOARD cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the WRITE_CLIPBOARD built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the WRITE_CLIPBOARD built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the WRITE_CLIPBOARD built-in.

EXAMPLES

1
```
WRITE_CLIPBOARD ("", this_range);
```

This statement writes the contents of the range *this_range* to the clipboard.

2
```
PROCEDURE eve$$cut_copy (delete_range)

LOCAL   remove_range,           ! Local copy of the currently
                                ! selected range.
        done_message;           ! Success message.

ON_ERROR
    [TPU$_CLIPBOARDLOCKED]:
        eve$message (EVE$_CLIPBDWRITLOCK);
        eve$learn_abort;
        RETURN (FALSE);
    [OTHERWISE]:
        eve$learn_abort;
ENDON_ERROR;

remove_range := eve$selection (TRUE);
IF remove_range <> 0
THEN
    WRITE_CLIPBOARD ("", remove_range);  ! This statement writes a copy
                                         ! of the selected range to the
                                         ! clipboard.
```

```
    IF delete_range
    THEN
        done_message := EVE$_REMCOMPL;
        ERASE (remove_range);

    ELSE
        done_message := EVE$_COPYCOMPL;
    ENDIF;
    remove_range := 0;
    eve$message (done_message);
    RETURN (TRUE);
ENDIF;

eve$learn_abort;
RETURN (FALSE);

ENDPROCEDURE;
```

This procedure shows one possible way that a layered application can use the WRITE_CLIPBOARD built-in. This procedure is a copy of the EVE procedure EVE$$CUT_COPY. You can find this procedure in SYS$EXAMPLES:EVE$DECWINDOWS.TPU.

The procedure checks whether a selection is active and, if so, writes the contents of the selected range to the clipboard. If the user has directed EVE to cut the selected text, the procedure erases the selected range.

WRITE_FILE

Writes data to the file that you specify. WRITE_FILE optionally returns a string that is the file specification of the file created.

FORMAT

[string2 :=] WRITE_FILE ({ *buffer* / *range* } *[,string1]*)

PARAMETERS

buffer

The buffer whose contents you want to write to a file.

range

The range whose contents you want to write to a file.

If you use WRITE_FILE on a range that does not start at the left margin of a line, VAXTPU does the following:

- Determines the left margin of the line in which the range starts
- Writes the range to the output file starting at the same left margin as the margin of the line where the range starts

For example, if you write a range that starts in column 30 of a line whose left margin is 10, WRITE_FILE writes the range in the output file starting at column 10.

string1

A string specifying the file to which the contents of the buffer are to be written. If you do not specify a full file specification, VAXTPU determines the output file specification using the current device and directory as defaults.

This parameter is optional. If you omit it, VAXTPU uses the associated output file name for the buffer. If there is no associated file name, VAXTPU prompts you for one. If you do not give a file name at the prompt, VAXTPU does not write to a file. In that case, the optional *string2* that is returned is a null string.

return value

A string representing the file specification of the file created.

DESCRIPTION

If you specify a result, WRITE_FILE returns a string that is the file specification of the file to which the data was written.

VAXTPU uses a flag to mark a buffer as modified or not modified. When you write data from a buffer to an external file, VAXTPU clears the modified flag for that buffer. If you do not make any further modifications to that buffer, VAXTPU does not consider the buffer as being modified and does not write out the file by default when you exit. If an error occurs while VAXTPU is writing a file, VAXTPU does not clear the modified flag.

When the contents of a buffer are written to a file, the associated journal file (if any) is closed and deleted and a new journal file is created. The new file contains the name of the file to which the buffer was written.

Note that deleting the file that has been written out invalidates the buffer change journal.

See Appendix F for a list of the file types that VAXTPU supports.

SIGNALED ERRORS

TPU$_CONTROLC	ERROR	The execution of the write operation terminated because you pressed CTRL/C.
TPU$_TOOFEW	ERROR	WRITE_FILE requires at least one parameter.
TPU$_TOOMANY	ERROR	WRITE_FILE accepts no more than two parameters.
TPU$_ARGMISMATCH	ERROR	One of the parameters to WRITE_FILE is of the wrong type.
TPU$_INVPARAM	ERROR	One of the parameters to WRITE_FILE is of the wrong type.

The following completion codes can be signaled by VAXTPU's file I/O routine. You can provide your own file I/O routine by using the VAXTPU callable interface. If you do so, WRITE_FILE's completion status depends upon what status you signaled in your file I/O routine.

TPU$_OPENOUT	ERROR	WRITE_FILE could not create the output file.
TPU$_NOFILEACCESS	ERROR	WRITE_FILE could not connect to the newly created output file.
TPU$_WRITEERR	ERROR	WRITE_FILE could not write the text to the file because it encountered a file system error during the operation.
TPU$_CLOSEOUT	ERROR	WRITE_FILE encountered a file system error closing the file.

EXAMPLES

1
```
WRITE_FILE (paste_buffer, "myfile.txt")
```

This statement writes the contents of the paste buffer to the file named MYFILE.TXT.

2
```
my_file := WRITE_FILE (select_range, "myfile.txt")
```

This assignment statement puts the file name to which the *select_range* is written in the string *my_file*.

```
3  PROCEDURE user_write_file
      WRITE_FILE (extra_buf);
      DELETE (extra_window);
      DELETE (extra_buf);
   ! Return the lines from extra_window to the main window
      ADJUST_WINDOW (main_window, -11, 0);
   ENDPROCEDURE;
```

This procedure writes the contents of a buffer called *extra_buf* to a file (because you do not specify a file name, the associated file for the buffer is used). The procedure then removes the extra window and buffer from your editing context.

WRITE_GLOBAL_SELECT

Sends requested information about a global selection from the VAXTPU layered application to the application that issued the information request.

FORMAT

WRITE_GLOBAL_SELECT ({ *array* | *buffer* | *range* | *string* | *integer* | *NONE* })

PARAMETERS

array

An array that passes information about a global selection whose contents describe information that is not of a data type supported by VAXTPU. For example, the array could pass information about a pixmap, an icon, or a span.

VAXTPU does not use or alter the information in the array; the application layered on VAXTPU is responsible for determining how the information is used, if at all. Since the array is used to pass information to and from other DECwindows applications, all applications that send or receive information whose data type is not supported by VAXTPU must agree on how the information is to be sent and used.

The application sending the information is responsible for creating the array and giving it the proper structure. The array's structure is as follows:

- The element *array {0}* contains a string naming the data type of the information being passed. For example, if the information being passed is a span, the element contains the string *"SPAN"*.
- The element *array {1}* contains either the integer 8, indicating that the information is passed as a series of bytes, or the integer 32, indicating that the information is passed as a series of longwords.
- If *array {1}* contains the value 8, the element *array {2}* contains a string and there are no array elements after *array {2}*. The string does not name anything, but rather is a series of bytes. As mentioned, the meaning and use of the information is agreed upon by convention among the DECwindows applications.
- If *array {1}* contains the value 32, the remaining elements of the array contain integer data. In this case, the array can have any number of elements after *array {2}*. These elements must be numbered sequentially, starting at *array {3}*. All the elements contain integers. Each integer represents a longword of data. To determine how many longwords are being passed, an application can determine the length of the array and subtract 2 to allow for elements *array {0}* and *array {1}*.

buffer
The buffer containing the information to be sent to the requesting application as the response to the global selection information request. If you specify a buffer, VAXTPU converts the buffer to a string, converts line breaks to line feeds, and inserts padding blanks before text to fill any unoccupied space before the left margin.

range
The range containing the information to be sent to the requesting application as the response to the global selection information request. If you specify a range, VAXTPU converts the buffer to a string, converts line breaks to line feeds, and inserts padding blanks before and after text to fill any unoccupied space before the left margin.

string
The string containing the information to be sent to the requesting application as the response to the global selection information request. VAXTPU sends the information in string format.

integer
An integer whose value is to be sent to the requesting application as the response to the global selection information request. VAXTPU sends the information in integer format.

NONE
A keyword indicating that no information about the global selection is available.

DESCRIPTION

WRITE_GLOBAL_SELECT is valid only inside a routine that responds to requests for information about a global selection.

The parameter specifies the data to supply to the requesting application. If you specify NONE, VAXTPU informs the requesting application that no information is available. Note, however, that for any case in which a routine omits a WRITE_GLOBAL_SELECT statement, by default VAXTPU informs the requesting application that no information is available.

Call WRITE_GLOBAL_SELECT no more than once during the execution of a global selection read routine. VAXTPU signals TPU$_INVBUILTIN if you attempt to call this routine more than once.

SIGNALED ERRORS

TPU$_BUILTININV	WARNING	WRITE_GLOBAL_SELECT has been used more than once in the same routine.
TPU$_TRUNCATE	WARNING	VAXTPU has truncated characters from the data written because you specified a buffer or range containing more than 65,535 characters.

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	WRITE_GLOBAL_SELECT cannot return a value.
TPU$_REQUIRESDECW	ERROR	You can use the WRITE_GLOBAL_SELECT built-in only if you are using DECwindows VAXTPU.
TPU$_TOOFEW	ERROR	Too few arguments passed to the WRITE_GLOBAL_SELECT built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the WRITE_GLOBAL_SELECT built-in.

EXAMPLE

```
WRITE_GLOBAL_SELECT (this_range);
```

This statement sends the contents of the range *this_range* to the requesting application.

For an example of a procedure using the WRITE_GLOBAL_SELECT built-in, see Example B–11.

A Sample VAXTPU Procedures

The following VAXTPU procedures are included as samples of how to use VAXTPU to perform certain tasks. These procedures merely show one way of using VAXTPU; there may be other, more efficient ways to perform the same task. Make changes to these procedures to accommodate your style of editing.

For these procedures to compile and execute correctly, you must make sure that there are no conflicts between these sample procedures and your interface. The following types of procedures are contained in this appendix:

1. Line-mode editor
2. Translation of control characters
3. Restoring terminal width before exiting from VAXTPU
4. DCL command procedure to run VAXTPU from a subprocess

A.1 Line-Mode Editor

The following example shows a portion of an editing interface that uses line mode rather than screen displays for editing tasks. This mode of editing can be used for batch jobs, or for running VAXTPU on terminals that do not support screen-oriented editing.

```
! Portion of a line mode editor for VAXTPU
! Invoked from DCL with: EDIT/TPU/NODISPLAY/NOSECTION/COM=linemode.tpu file
!
input_file := GET_INFO (COMMAND_LINE, "file_name");  ! Set up main
main_buffer := CREATE_BUFFER ("MAIN", input_file);   ! buffer from input
POSITION (BEGINNING_OF (main_buffer));               ! file
!
LOOP                    ! Continuously loop until QUIT
    cmd := READ_LINE ("*");
    IF cmd = ""
    THEN
        cmd_char := "N";
    ELSE
        cmd_char := SUBSTR (cmd, 1, 1);  CHANGE_CASE (cmd_char, UPPER);
    ENDIF;

    CASE cmd_char FROM "I" TO "T"       ! Only accepting I,L,N,Q,T
```

```
!Top of buffer command
        ["T"]:
                POSITION (BEGINNING_OF (CURRENT_BUFFER));
                MESSAGE (CURRENT_LINE);
!Next line command
         ["N"]:
                MOVE_HORIZONTAL (-CURRENT_OFFSET);
                MOVE_VERTICAL (1);
                MESSAGE (CURRENT_LINE);
!Insert text command
         ["I"]:
                SPLIT_LINE;
                COPY_TEXT (SUBSTR (cmd, 2, 999));
                MESSAGE (CURRENT_LINE);
!List from here to end of file command
         ["L"]:
                m1 := MARK (NONE);
                LOOP
                MESSAGE (CURRENT_LINE);
                MOVE_VERTICAL (1);
                EXITIF MARK (NONE) = END_OF (CURRENT_BUFFER);
                ENDLOOP;
                POSITION (m1);
!QUIT
         ["Q"]:
                QUIT;
                 [INRANGE,OUTRANGE]:
                MESSAGE ("Unrecognized command - enter I,L,N,Q or T");
    ENDCASE;
ENDLOOP;
```

A.2 Translation of Control Characters

The following procedures are examples of how to display control characters in a meaningful way. This is accomplished by translating the buffer to a different visual format and mapping this new form to a window. On the VT300 series and VT200 series of terminals, control characters are shown as reverse question marks. On the VT100 series of terminals, they are shown as rectangles.

```
! This procedure performs the substitution of meaningful characters
! for the escape or control characters.
!
PROCEDURE translate_controls (char_range)

    LOCAL
        replace_text;
!
! If the translation array is not yet set up, then do it now.  The elements
! that we do not initialize will contain the value TPUK_UNSPECIFIED.  They are
! characters that TPU will display meaningfully.
!
    IF translate_array = TPU$K_UNSPECIFIED
    THEN
```

```
        translate_array := CREATE_ARRAY (32, 0);
        translate_array {1} := '<SOH>';
        translate_array {2} := '<STX>';
        translate_array {3} := '<ETX>';
        translate_array {4} := '<EOT>';
        translate_array {5} := '<ENQ>';
        translate_array {6} := '<ACK>';
        translate_array {7} := '<BEL>';
        translate_array {8} := '<BS>';
        translate_array {14} := '<SO>';
        translate_array {15} := '<SI>';
        translate_array {16} := '<DLE>';
        translate_array {17} := '<DC1>';
        translate_array {18} := '<DC2>';
        translate_array {19} := '<DC3>';
        translate_array {20} := '<DC4>';
        translate_array {21} := '<NAK>';
        translate_array {22} := '<SYN>';
        translate_array {23} := '<ETB>';
        translate_array {24} := '<CAN>';
        translate_array {25} := '<EM>';
        translate_array {26} := '<SUB>';
        translate_array {27} := '<ESC>';
        translate_array {28} := '<FS>';
        translate_array {29} := '<GS>';
        translate_array {30} := '<RS>';
        translate_array {31} := '<US>';
    ENDIF;
!
! The range *must* be a single character long
!
    IF LENGTH (char_range) <> 1
    THEN
        RETURN 0;
    ENDIF;
!
! Find the character
!
    replace_text := translate_array {ASCII (STR (char_range))};
!
! If we got back a value of TPU$K_UNSPECIFIED, TPU will display the character
! meaningfully
!
    IF replace_text = TPU$K_UNSPECIFIED
    THEN
        RETURN 0;
    ENDIF;
!
! Erase the range and insert the new text
!
    ERASE (char_range);
    COPY_TEXT (replace_text);

    RETURN 1;

ENDPROCEDURE;

!
! This procedure controls the outer loop search for the special
! control characters that we want to view.
!
PROCEDURE view_controls (source_buffer)
```

```
    CONSTANT
        ctrl_char_str :=
                ASCII  (0) + ASCII  (1) + ASCII  (2) + ASCII  (3) +
                ASCII  (4) + ASCII  (5) + ASCII  (6) + ASCII  (7) +
                ASCII  (8) + ASCII  (9) + ASCII (10) + ASCII (11) +
                ASCII (12) + ASCII (13) + ASCII (14) + ASCII (15) +
                ASCII (16) + ASCII (17) + ASCII (18) + ASCII (19) +
                ASCII (20) + ASCII (21) + ASCII (22) + ASCII (23) +
                ASCII (24) + ASCII (25) + ASCII (26) + ASCII (27) +
                ASCII (28) + ASCII (29) + ASCII (30) + ASCII (31);
    LOCAL
        ctrl_char_pattern,
        ctrl_char_range;
!
! Create the translation buffer and window, if necessary
!
    IF translate_buffer = TPU$K_UNSPECIFIED
    THEN
        translate_buffer := CREATE_BUFFER ("translation");
        SET (NO_WRITE, translate_buffer);
    ENDIF;
    IF translate_window = TPU$K_UNSPECIFIED
    THEN
        translate_window := CREATE_WINDOW (1, 10, ON);
    ENDIF;
!
! Make a copy of the buffer we are translating
!
    ERASE (translate_buffer);
    POSITION (translate_buffer);
    COPY_TEXT (source_buffer);
!
! Search for any control characters and translate them.  If a control character
! is not found, SEARCH_QUIETLY will return a 0.
!
    ctrl_char_pattern := ANY (ctrl_char_str);
    POSITION (BEGINNING_OF (translate_buffer));
    LOOP
        ctrl_char_range := SEARCH_QUIETLY (ctrl_char_pattern, FORWARD);
        EXITIF ctrl_char_range = 0;
        POSITION (ctrl_char_range);
        !
        ! If we did not translate the character, move past it
        !
        IF NOT translate_controls (ctrl_char_range)
        THEN
            MOVE_HORIZONTAL (1);
        ENDIF;
    ENDLOOP;
!
! Now display what we have done
!
    POSITION (BEGINNING_OF (translate_buffer));
    MAP (translate_window, translate_buffer);
ENDPROCEDURE;
```

A.3 Restoring Terminal Width Before Exiting from VAXTPU

The following procedure compares the current width of the screen with the original width. If the current width differs from the original width, the procedure restores each window to its original width. The screen is refreshed so that information is visible on the screen after you exit from VAXTPU. When all of the window widths are the same, the physical screen width is changed.

```
PROCEDURE user_restore_screen

LOCAL
    original_screen_width,
    temp_w;

original_screen_width := GET_INFO (SCREEN, "original_width");

IF original_screen_width <> GET_INFO (SCREEN, "width")
THEN
    temp_w := get_info(windows,"first");

    LOOP
        EXITIF temp_w = 0;

        SET (WIDTH, temp_w, original_screen_width);

        temp_w := GET_INFO (WINDOWS, "next");
    ENDLOOP;

    REFRESH;
ENDIF;

ENDPROCEDURE;

! Define the key combination CTRL/E to do an exit which
! restores the screen to its original width, repaints
! the screen, and then exits.

DEFINE_KEY ("user_restore_screen;EXIT", CTRL_E_KEY);
```

A.4 Running VAXTPU from a Subprocess

The following DCL command procedure shows one way of running VAXTPU from a subprocess. It also shows how to move to or from the subprocess.

```
!
!DCL command procedure to run VAXTPU from subprocess
!
!Put $ e = "@keptedit"
!in your login.com. This spawns the editor the first time
!and attaches to it after that.  I have defined a key to be
!"attach" so it always goes back to the parent.
```

```
$ tt = f$getdvi("sys$command","devnam") - "_" - "_" - ":"
$ edit_name = "Edit_" + tt
$ priv_list = f$setprv("NOWORLD, NOGROUP")
$ pid = 0
$10$:
$ proc = f$getjpi(f$pid(pid), "PRCNAM")
$ if proc .eqs. edit_name then goto attach
$ if pid .ne. 0 then goto 10$
$spawn:
$ priv_list = f$setprv(priv_list)
$ write sys$error "[Spawning a new Kept Editor]"
$ define/nolog sys$input sys$command:
$ t1 = f$edit(p1 + " " + p2 + " " + p3 + " " + p4 + " "
  + p5 + " " + p6 + " " + p7 + " " +  p8,"COLLAPSE")
$ spawn/process="''edit_name'" /nolog edit/tpu 't1'
$ write sys$error "[Attached to DCL in directory ''f$env("DEFAULT")']"
$ exit
$attach:
$ priv_list = f$setprv(priv_list)
$ write sys$error "[Attaching to Kept Editor]"
$ define/nolog sys$input sys$command:
$ attach "''edit_name'"
$ write sys$error "[Attached to DCL in directory ''f$env("DEFAULT")']"
$ exit
```

B Sample DECwindows VAXTPU Procedures

B.1 Using DECwindows VAXTPU Built-ins

You can use the DECwindows VAXTPU built-in procedures in many ways. However, you may find it useful to look at sample procedures showing how other programmers have used some of the DECwindows VAXTPU built-ins. Therefore, this appendix presents a number of procedures using DECwindows built-ins.

The following example procedures are contained in this appendix:

1 Displaying a Dialog Box

2 Creating a "Mouse Pad"

3 Implementing an EDT-Style APPEND Command

4 Testing and Returning a Select Range

5 Resizing Windows

6 Unmapping Saved Windows

7 Mapping Saved Windows

8 Handling Callbacks from a Scroll Bar Widget

9 Implementing the COPY SELECTION Operation

10 Reactivating a Select Range

11 Implementing the DECwindows COPY SELECTION Operation from EVE to Another Application

Most of the procedures are drawn from the code implementing the Extensible VAX Editor (EVE). Some have been modified to make them easier to understand.

You can see all the code used to implement EVE by looking at the files in the directory pointed to by the logical name SYS$EXAMPLES. To see a directory of the files available, type the following command from the DCL command line:

```
$  DIR SYS$EXAMPLES:EVE$*.*
```

These files contain procedures using many of the VAXTPU built-ins.

B.2 Displaying a Dialog Box

Example B–1 illustrates one of the ways a layered application can use the CONVERT built-in. This procedure is a modified version of the EVE procedure *eve$$mb2_dispatch*. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

B.2 Displaying a Dialog Box

The procedure displays EVE's selection pop-up menu on the screen if the procedure is called while a select range or found range is active.

This example uses the following global variables and procedures:

- EVE$CALLBACK_DISPATCH — The procedure that EVE uses to dispatch all widget callbacks.
- EVE$X_FOUND_RANGE — A global variable that holds the range for the last text found. If there is not currently a found range, it is set to zero.
- EVE$X_SELECT_POPUP — A global variable that holds the pop-up menu widget used when a selection is present.
- EVE$X_SELECT_POPUP_HEIGHT — A global variable that holds the height of the selection pop-up menu.
- EVE$X_SELECT_POPUP_WIDTH — A global variable that holds the width of the selection pop-up menu.
- EVE$X_SELECT_POSITION — A global variable that holds the start marker for the select range. If there is not currently a selection, it is set to zero.

Example B–1 EVE Procedure That Displays a Selection Dialog Box

```
PROCEDURE eve$$mb2_dispatch

local   status,
        the_window,
        temp_array,
        the_widget,
        x_1,
        X2,
        widget_hierarchy,
        y_1,
        y_2;


❶ IF (LOCATE_MOUSE (the_window, x_1, y_1) <> 0)
  THEN
❷     CONVERT (the_window, CHARACTERS, x_1, y_1,
             DECW_ROOT_WINDOW, COORDINATES,
             X2, y_2);


      IF (eve$x_select_position <> 0) OR          ! A selection exists
         (eve$x_found_range <> 0)                 ! A found range exists

      THEN
          IF GET_INFO (eve$x_select_popup, "type") <> WIDGET

          THEN
❸             widget_hierarchy := SET (DRM_HIERARCHY, "EVE$WIDGETS");
```

(continued on next page)

Example B–1 (Cont.) EVE Procedure That Displays a Selection Dialog Box

```
            eve$x_select_popup := CREATE_WIDGET ("SELECT_POPUP",
                                                 widget_hierarchy,
                                                 SCREEN,
                                                 "eve$callback_dispatch");

        ENDIF;

            ! Get width and height of this pop-up menu if needed

❹       temp_array := CREATE_ARRAY;
            temp_array {eve$dwt$c_width} := 0;
            temp_array {eve$dwt$c_height} := 0;
            status := GET_INFO (eve$x_select_popup, "WIDGET_INFO", temp_array);
            eve$x_select_popup_width := temp_array {eve$dwt$c_width};
            eve$x_select_popup_height := temp_array {eve$dwt$c_height};

        ! Calculate position for upper left corner of
        ! dialog box and set the appropriate resources of the widget

        temp_array := CREATE_ARRAY;
        temp_array {eve$dwt$c_nx} := X2 - (eve$x_select_popup_width/2);
        IF temp_array {eve$dwt$c_nx} < 1
        THEN
            temp_array {eve$dwt$c_nx} := 1;
        ENDIF;
        temp_array {eve$dwt$c_ny} := y_2 - (eve$x_select_popup_height/2);
        IF temp_array {eve$dwt$c_ny} < 1
        THEN
            temp_array {eve$dwt$c_ny} := 1;
        ENDIF;

❺       SET (WIDGET, eve$x_select_popup, temp_array);
❻       MANAGE_WIDGET (eve$x_select_popup);
    ENDIF;
ENDIF;
RETURN (TRUE);

ENDPROCEDURE;
```

❶ The return value from the LOCATE_MOUSE built-in procedure indicates whether the pointer cursor is in the window. LOCATE_MOUSE also returns the row, column and window where the pointer cursor is located. The coordinates returned refer to a system whose origin is in the upper left corner of the VAXTU window.

❷ This clause converts the pointer cursor location from a system whose origin is at the upper left corner of the VAXTPU window to a system whose origin is at the upper left corner of the DECwindows root window. For more information about the difference between VAXTPU windows and DECwindows windows, see Chapter 4.

❸ SET (DRM_HIERARCHY, file_spec) allows you to tell VAXTPU which XUI Resource Manager hierarchy to use. An XUI Resource Manager hierarchy is a set of widgets implementing a user interface. For example, EVE's menu bar and menu widgets compose an XUI Resource Manager hierarchy.

EVE uses the XUI Resource Manager hierarchy stored in the file EVE$WIDGETS.UID. If you are extending EVE, you need not set the hierarchy again.

VAXTPU allows you to use multiple XUI Resource Manager hierarchies. If you want to use a second hierarchy (defined in a file other than EVE$WIDGETS.UID), use the SET (DRM_HIERARCHY) statement before using the CREATE_WIDGET statement.

❹ GET_INFO (widget, "widget_info", array) allows you to fetch information about a widget. The index of each element of the array must be a string naming the resource whose value you want to fetch. For more information about what resources a given widget supports, see the *VMS DECwindows Toolkit Routines Reference Manual.*

❺ SET (WIDGET, widget, array) allows you to set a widget's resource values. The index of each element of the array must be a string naming the resource whose values you want to set. For more information about what resources a given widget supports, see the *VMS DECwindows Toolkit Routines Reference Manual.*

❻ MANAGE_WIDGET realizes the widget and makes it visible on the screen.

B.3 Creating a "Mouse Pad"

Example B–2 shows how to use the variant of CREATE_WIDGET that calls the XUI Toolkit low-level creation routine. The module in Example B–2 creates a screen representation of a keypad. Instead of pressing a keypad key, a user can click on the widget representing the key.

Example B–2 Procedure That Creates a "Mouse Pad"

```
! SAMPLE.TPU

!++
!                                Table of Contents
!
!                                   SAMPLE.TPU
!
!        Procedure name          Description
!        --------------          -----------
!
!        sample_sample_module_ident   Ident.
!        sample_sample_module_init    Initializes the module.
!        eve_mouse_pad                Implements the user command DISPLAY MOUSE PAD.
!        sample_key_def               Creates a mouse pad "key" push button.
!        sample_key_dispatch          Handles push button widget callbacks.
!        sample_row_to_pix            Converts a row number to pixels.
!        sample_col_to_pix            Converts a column number to pixels.
!        sample_key_height            Converts y dimension from rows to pixels.
!        sample_key_width             Converts x dimension from columns to pixels.
!--
```

(continued on next page)

Example B–2 (Cont.) Procedure That Creates a "Mouse Pad"

```
! This module layers a "mouse pad" on top of VAXTPU.  The mouse pad
! is implemented by creating a dialog box widget that is the parent of a group
! of push button widgets depicting keypad keys.  The resulting
! "mouse pad" is a screen representation of a keypad.  The user can
! click on a push button to execute the same function that would be
! executed by pressing the corresponding keypad key.  The module uses
! the key map list mapped to the current buffer to determine what
! code to execute when the user clicks on a given push button.  To
! use a different key map, substitute a string naming the desired
! key map for the null string assigned to "sample_k_keymap".
! This module can be used with the EVE section file
! or with a non-EVE section file.
!
! This module uses the variant of CREATE_WIDGET that calls the XUI
! Toolkit low-level creation routine.


PROCEDURE sample_sample_module_ident                ! This procedure returns
RETURN "V01-001";                                   ! the Ident.
ENDPROCEDURE;


PROCEDURE sample_sample_module_init          ! Module initialization.
ENDPROCEDURE;

! VAXTPU Declarations for XUI Toolkit constants

      ! Use these constants as arguments to the DEFINE_WIDGET built-in.
      ! The strings are the symbols that evaluate to the
      ! widget class records for the DECwindows widgets.

CONSTANT
    sample_k_labelwidgetclass := "labelwidgetclassrec",
    sample_k_dialogwidgetclass := "dialogwidgetclassrec",
    sample_k_pushbuttonwidgetclass := "pushbuttonwidgetclassrec";


      ! Use these constants, which are XUI Toolkit
      ! resource name strings, as callback reasons, resource values, or
      ! arguments to the CREATE_WIDGET built-in.

CONSTANT
    sample_k_cstyle := "style",
    sample_k_modeless := 2,
    sample_k_nunits := "units",
    sample_k_pixelunits := 1,
    sample_k_ntitle := "title",
    sample_k_nx := "x",
    sample_k_ny := "y",
    sample_k_nheight := "height",
    sample_k_nwidth := "width",
    sample_k_nlabel := "label",
    sample_kt_nactivate_callback := "activateCallback",
    sample_kt_nborderwidth := "borderWidth",
    sample_kt_nconformToText := "conformToText",
    sample_k_cractivate := 10;
```

(continued on next page)

Example B–2 (Cont.) Procedure That Creates a "Mouse Pad"

```
! These constants are intended for use only in this sample module
! because their values are specific to the mouse pad application.

CONSTANT
    sample_k_x_pos := 500,            ! Screen position for mouse pad.
    sample_k_x_pos := 500,
    sample_k_keypad_border := 5,      ! Width of border between keys and edge.
    sample_k_key_height := 30,        ! Key dimensions.
    sample_k_key_width := 60,
    sample_k_button_border_frac := 3, ! Determines spacing between keys.

    sample_k_overall_height := (sample_k_key_height * 5)
                               + ((sample_k_key_height
                                   / sample_k_button_border_frac) * 5)
                               + sample_k_keypad_border,

    sample_k_overall_width := (sample_k_key_width * 4)
                              + ((sample_k_key_width
                                  / sample_k_button_border_frac) * 4)
                              + sample_k_keypad_border,

    sample_k_keymap := '',      ! If this constant has a null string
                                ! as its value, the program uses the
                                ! current key map list to determine what
                                ! code to execute when the user
                                ! clicks on a given push button.

    sample_k_pad_title := "Sample mouse pad",    ! Title of the mouse pad.
    sample_k_closure := '';                      ! Not currently used.



PROCEDURE eve_mouse_pad         ! Implements a user-created command MOUSE PAD
ON_ERROR                        ! that the user can invoke from within EVE.
    [TPU$_CONTROLC]:
        eve$learn_abort;
        ABORT;
ENDON_ERROR


! Checks whether the dialog box widget class has already been defined.
! If not, defines the dialog box widget class and creates a widget
! instance to be used as the "container" for the mouse pad.

IF GET_INFO (sample_x_dialog_class, 'type') <> INTEGER
THEN
    sample_x_dialog_class
(1)     := DEFINE_WIDGET_CLASS (sample_k_dialogwidgetclass,
                                "dwt$dialog_box_popup_create");
ENDIF;
```

(continued on next page)

Example B–2 (Cont.) Procedure That Creates a "Mouse Pad"

```
❷ sample_x_keypad := CREATE_WIDGET (sample_x_dialog_class, "Keypad", SCREEN,
                                    "MESSAGE('CALLBACK activated')",
                                    "sample_k_closure ",
                                    sample_k_cstyle, sample_k_modeless,
                                    sample_k_nunits, sample_k_pixelunits,
                                    sample_k_ntitle, sample_k_pad_title,
                                    sample_k_nheight, sample_k_overall_height,
                                    sample_k_nwidth, sample_k_overall_width,
                                    sample_k_nx, sample_k_x_pos,
                                    sample_k_ny, sample_k_y_pos);

! Checks whether the push button widget class has already been defined
! and, if not, defines the class.

IF GET_INFO (sample_x_pushbutton_class, 'type') <> INTEGER
THEN
    sample_x_pushbutton_class                                          ! This statement
        := DEFINE_WIDGET_CLASS (sample_k_pushbuttonwidgetclass,        ! using the built_in
                                "dwt$push_button_create");             ! DEFINE_WIDGET_CLASS
ENDIF;                                                                 ! defines the
                                                                       ! class of the
                                                                       ! push button
                                                                       ! widgets.

! Initializes the array that the program passes repeatedly
! to the procedure "sample_key_def".

sample_x_attributes := CREATE_ARRAY;
sample_x_attributes {sample_k_nactivate_callback} := 0;
sample_x_attributes {sample_k_nborderwidth} := 2;
sample_x_pad_program := COMPILE ("sample_key_dispatch");

! Creates and manages all the "keys" in the mouse pad.  The procedure
! "sample_key_def" returns a variable of type widget, so you can use the
! returned value as an argument to the built-in MANAGE_WIDGET.

❸ MANAGE_WIDGET (sample_key_def ("PF1", 0, 0, 1, 1, sample_x_pad_program),
                 sample_key_def ("PF2", 1, 0, 1, 1, sample_x_pad_program),
                 sample_key_def ("PF3", 2, 0, 1, 1, sample_x_pad_program),
                 sample_key_def ("PF4", 3, 0, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP7", 0, 1, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP8", 1, 1, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP9", 2, 1, 1, 1, sample_x_pad_program),
                 sample_key_def ("-", 3, 1, 1, 1, sample_x_pad_program, "minus"),
                 sample_key_def ("KP4", 0, 2, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP5", 1, 2, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP6", 2, 2, 1, 1, sample_x_pad_program),
                 sample_key_def (",", 3, 2, 1, 1, sample_x_pad_program, "comma"),
                 sample_key_def ("KP1", 0, 3, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP2", 1, 3, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP3", 2, 3, 1, 1, sample_x_pad_program),
                 sample_key_def ("Enter", 3, 3, 2, 1, sample_x_pad_program,
                                 "enter"),
                 sample_key_def ("KP0", 0, 4, 1, 2, sample_x_pad_program),
                 sample_key_def (".", 2, 4, 1, 1, sample_x_pad_program,
                                 "period"));

sample_x_shift_was_last := FALSE;      ! The program starts out assuming that
                                       ! no GOLD key has been pressed.
```

(continued on next page)

Example B–2 (Cont.) Procedure That Creates a "Mouse Pad"

```
❹ MANAGE_WIDGET (sample_x_keypad);      ! This statement displays the
                                        ! resulting mouse pad.

RETURN (TRUE);
ENDPROCEDURE; ! End of procedure eve_mouse_pad.


PROCEDURE sample_key_def                ! Creates a mouse pad "key" push button
                                        ! widget.
    (the_legend,                ! What characters to show on the push button label.
     the_row, the_col,          ! Location of the key in relation to the parent
                                ! widget's upper left corner.
     the_width, the_height,     ! Dimensions of the key.
     the_pgm;                   ! Program to use as the callback routine; used
                                ! as a parameter to the CREATE_WIDGET built-in.
     the_string);               ! The string representation of the name
                                ! of a key if the key name is not going
                                ! to be the same as the legend (as in
                                ! the case of the comma).  Specify the null
                                ! string if the key name and the legend are
                                ! the same.
IF GET_INFO (the_string, 'type') = UNSPECIFIED
THEN
    the_string := the_legend;       ! Determines whether the optional parameter
                                    ! the_string is provided.
ENDIF;

RETURN CREATE_WIDGET (sample_k_pushbutton_class, "Key", sample_x_keypad,
                      the_pgm,
                      (sample_k_keymap + ' ' + the_string),
                      sample_x_attributes,
                      sample_kt_nconformToText, 0,
                      sample_k_nlabel, the_legend,
                      sample_k_nheight, sample_key_height (the_width),
                      sample_k_nwidth, sample_key_width (the_height),
                      sample_k_nx, sample_col_to_pix (the_row),
                      sample_k_nx, sample_row_to_pix (the_col));

ENDPROCEDURE; ! End of the procedure "sample_key_def".


PROCEDURE sample_key_dispatch  ! Handles push button widget callbacks.

LOCAL   status,                ! Variable to contain the return value from
                               ! GET_INFO (WIDGET, "callback_parameters",).
        blank_index,           ! Position of the blank space in the tag string.
        temp_array,            ! Holds callback parameters.
        a_shift_key,           ! The SHIFT key in the current key map list.
        the_key,               ! A string naming a key.
        gold_key;              ! Name of the GOLD key.
```

(continued on next page)

Example B–2 (Cont.) Procedure That Creates a "Mouse Pad"

```
ON_ERROR
    [TPU$_CONTROLC]:
        eve$learn_abort;
        ABORT;
ENDON_ERROR

❺ status := GET_INFO (widget, "callback_parameters", temp_array);
$widget := temp_array {'widget'};
$widget_tag := temp_array {'closure'};
$widget_reason := temp_array {'reason_code'};

❻ the_key := EXECUTE ("RETURN(KEY_NAME (" + $widget_tag + "))");
gold_key := GET_INFO (eve$current_key_map_list, "shift_key");
IF the_key = gold_key
THEN
    sample_shift_was_last := TRUE;      ! User pressed Gold Key

ELSE
    IF sample_shift_was_last
    THEN
        the_key := KEY_NAME (the_key, SHIFT_KEY);

    ENDIF;
    CASE $widget_reason
        [sample_kt_cractivate]:
            EXECUTE (the_key);
        [OTHERWISE]:
            eve_show_key (the_key)
    ENDCASE;
    sample_shift_was_last := FALSE;
ENDIF;
RETURN;
ENDPROCEDURE; ! End of the procedure "sample_key_dispatch".


! These procedures implement position and
! size calculations for the push button widgets.
PROCEDURE sample_row_to_pix (row)        ! Converts a row number to the
                                         ! pixel-based measuring system.
RETURN sample_k_keypad_border +
    (row * (sample_k_key_height + (sample_k_key_height
                                   / sample_k_button_border_frac)));
ENDPROCEDURE; ! End of the procedure "sample_row_to_pix".


PROCEDURE sample_col_to_pix (col)       ! Converts a column number to the
                                        ! pixel-based measuring system.
RETURN sample_k_keypad_border +
    (col * ((sample_kt_key_width + sample_kt_key_width)
                                  / sample_kt_button_border_frac ));

ENDPROCEDURE; ! End of the procedure "sample_col_to_pix".
```

(continued on next page)

Example B–2 (Cont.) Procedure That Creates a "Mouse Pad"

```
PROCEDURE sample_key_height (given_height) ! Converts the y dimension
                                           ! from rows to pixels.
IF given_height = 1
THEN
    RETURN sample_k_key_height;
ELSE
    RETURN ((sample_k_key_height * given_height)
            + (sample_k_key_height / sample_k_button_border_frac)
            * (given_height - 1));
ENDIF;
ENDPROCEDURE; ! End of the procedure "sample_key_height".


PROCEDURE sample_key_width (given_width)    ! Converts the x dimension
                                            ! from rows to pixels.
IF given_width = 1
THEN
    RETURN sample_k_key_width;
ELSE
    RETURN ((sample_k_key_width * given_width)
            + (sample_k_key_width / sample_k_button_border_frac)
            * (given_width - 1));
ENDIF;
ENDPROCEDURE; ! End of the procedure "sample_key_width".
```

❶ When you create widgets directly in VAXTPU (that is, without using the XUI Resource Manager to manipulate widgets defined in a UIL file) you must define each **class** of widget. For example, a widget can belong to the push button, dialog box, menu, or another similar class of widget. The DEFINE_WIDGET_CLASS built-in procedure tells VAXTPU the widget class name and creation entry point for the class of widget. DEFINE_WIDGET_CLASS also returns a widget ID for that widget class. Define a widget class for each widget only once in a VAXTPU session.

❷ The CREATE_WIDGET built-in allows you to create an **instance** of a widget for which you have a widget ID. An instance is one occurrence of a widget of a given class. For example, EVE has many menu widgets, each of which is an instance of a menu widget.

This example creates a dialog box widget to contain the mouse pad.

❸ Each of the keys of the mouse pad is managed. However, they do not become visible until their parent, the dialog box widget in variable SAMPLE_X_KEYPAD, is managed.

❹ Managing a widget whose parent is visible causes that widget and all its managed children to become visible.

❺ GET_INFO (WIDGET, "callback_parameters", array) returns the callback information in the array parameter. For more information about using this built-in, see the built-in's description in the VAXTPU Reference Section.

❻ When each key widget of the mouse pad is created, the closure value for the widget is set to the string corresponding to the name of the key that the widget represents. This statement uses the EXECUTE built-in to translate the string into a key name.

B.4 Implementing an EDT-Style APPEND Command

Example B–3 shows one of the ways an application can use the GET_CLIPBOARD built-in. This procedure is a modified version of the EVE procedure EVE$EDT_APPEND. You can find the original version in SYS$EXAMPLES:EVE$EDT.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

The procedure EVE$EDT_APPEND appends the currently selected text to the contents of the clipboard if the user has activated the clipboard; otherwise, the procedure appends the current selection to the contents of the Insert Here buffer.

This example uses the following global variables and procedures from EVE:

- EVE$MESSAGE — A procedure that translates the specified message code into text and displays the text in the Messages buffer.
- EVE$$RESTORE_POSITION — A procedure that repositions the editing point to the location indicated by the specified window and marker. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.
- EVE$LEARN_ABORT — A procedure that aborts a learn sequence.
- EVE$SELECTION — A procedure that returns a range containing the current selection. This can be the select range, the found range, or the text of the global selection.
- EVE$$TEST_IF_MODIFIABLE — A procedure that checks whether a buffer can be modified. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.
- EVE$X_DECWINDOWS_ACTIVE — A Boolean global variable that is true if VAXTPU is using DECwindows. If VAXTPU is not using DECwindows, the DECwindows features are not available.
- EVE$$X_STATE_ARRAY — A global variable of type array describing various EVE flags and data. This variable is private to EVE and should not be used by user routines.
- EVE$$EDT_APPEND_PASTE — Procedure that appends text to the Insert Here buffer. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.

Example B–3 EVE Procedure That Implements a Variant of the EDT APPEND Command

```
PROCEDURE eve$edt_append            ! Implements EVE's version of
                                    ! the EDT APPEND command.
LOCAL   saved_mark,                 ! Marks the editing point at the
                                    ! beginning of the procedure.

        remove_range,               ! Stores the currently selected text.

        old_string,                 ! Stores the text that was in the clipboard.

        new_string,                 ! Stores the old contents of the clipboard
                                    ! plus the currently selected text.

        remove_status;              ! Indicates whether the selected text
                                    ! should be removed.

ON_ERROR
    [TPU$_CLIPBOARDNODATA]:
        eve$message (EVE$_NOINSUSESEL);
        eve$$restore_position (saved_mark);
        eve$learn_abort;
        RETURN (FALSE);
    [TPU$_CLIPBOARDLOCKED]:
        eve$message (EVE$_CLIPBDREADLOCK);
        eve$$restore_position (saved_mark);
        eve$learn_abort;
        RETURN (FALSE);

    [TPU$_CONTROLC]:
        eve$$restore_position (saved_mark);
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:        eve$$restore_position (saved_mark);
        eve$learn_abort;
ENDON_ERROR;


remove_range := eve$selection (TRUE);
IF remove_range <> 0
THEN
    saved_mark := MARK (NONE);
    remove_status := eve$test_if_modifiable (GET_INFO (saved_mark, "buffer"));
    IF eve$x_decwindows_active
    THEN
        IF eve$$x_state_array {eve$$k_clipboard}
        THEN
❶           old_string := GET_CLIPBOARD;
            string_range := old_string + str (remove_range);
❷           WRITE_CLIPBOARD ("", new_string);
```

(continued on next page)

Example B–3 (Cont.) EVE Procedure That Implements a Variant of the EDT APPEND Command

```
            IF remove_status
            THEN
                ERASE (remove_range);
                eve$message (EVE$_REMCLIPBOARD);
            ENDIF;
        ELSE
            eve$$edt_append_paste (remove_range, remove_status);
        ENDIF;
    ELSE
        eve$$edt_append_paste (remove_range, remove_status);
    ENDIF;


    POSITION (saved_mark);
    remove_range := 0;
    RETURN (TRUE);
ENDIF;

eve$learn_abort;
RETURN (FALSE);

ENDPROCEDURE;
```

❶ The GET_CLIPBOARD built-in procedure returns a copy of the text stored in the clipboard. Only data of type string can be retrieved from the clipboard. Any other data type causes VAXTPU to signal an error.

❷ The WRITE_CLIPBOARD built-in procedure stores data in the clipboard. The first parameter allows you to specify the label for this data. However, the clipboard currently supports only one entry at a time, so you can use any string for the first parameter.

B.5 Testing and Returning a Select Range

The code fragment in Example B–4 shows how a layered application can use GET_GLOBAL_SELECT. This code fragment is a portion of the EVE procedure EVE$SELECTION. You can find the original version in SYS$EXAMPLES:EVE$CORE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

The procedure EVE$SELECTION returns a select range, found range, or global selection for use with EVE commands that operate on the select range.

This example uses the following global variables and procedures from EVE:

- EVE$MESSAGE — A procedure that translates the specified message code into text and displays the text in the message buffer.
- EVE$LEARN_ABORT — A procedure that aborts a learn sequence.

- EVE$X_DECWINDOWS_ACTIVE — A Boolean global variable that is true if VAXTPU is using DECwindows. If VAXTPU is not using DECwindows, the DECwindows features are not available.

Example B–4 EVE Procedure That Returns a Select Range

```
PROCEDURE eve$selection (
                         do_messages;           ! Display error messages?
                         found_range_arg,       ! Use found range? (D=TRUE).
                         global_arg,            ! Use global select? (D=FALSE).
                         null_range_arg,        ! Extend null ranges? (D=TRUE).
                         cancel_arg)            ! Cancel selection? (D=TRUE).

! Return Values:        range           The selected range.
!                       0               There was no select range.
!                       NONE            There was a null range and
!                                        null_range_arg is FALSE.
!                       string          Text of the global selection
!                                        if "global_arg" is TRUE.

LOCAL   possible_selection,
        use_found_range,
        use_global,
        extend_null_range,
        cancel_range;

ON_ERROR
    [TPU$_SELRANGEZERO]:
    [TPU$_GBLSELOWNER]:
        eve$message (EVE$_NOSELECT);
        eve$learn_abort;
        RETURN (FALSE);
    [OTHERWISE]:
ENDON_ERROR;

                                ! The procedure first tests whether it
                                ! has received a parameter directing
                                ! it to return a found range or global
                                ! selection if no select range has been
                                ! created by the user.

IF GET_INFO (found_range_arg, "type") = INTEGER
THEN
    use_found_range := found_range_arg;
ELSE
    use_found_range := TRUE;
ENDIF;

IF GET_INFO (global_arg, "type") = INTEGER
THEN
    use_global := global_arg;
ELSE
    use_global := FALSE;
ENDIF;

!  .
!  .
!  .
```

(continued on next page)

Example B–4 (Cont.) EVE Procedure That Returns a Select Range

```
                                        ! In the code omitted from this example,
                                        ! eve$selection returns the appropriate
                                        ! range if the calling procedure has
                                        ! requested the user's select range
                                        ! or a found range.
!   .
!   .
!   .

                                        ! If there is no found range or select
                                        ! range, the procedure returns
                                        ! the primary global selection
                                        ! if it exists.

IF use_global and eve$x_decwindows_active
THEN
(1)     possible_selection := GET_GLOBAL_SELECT (PRIMARY,
                                                 "STRING");
    IF GET_INFO (possible_selection, "type") = STRING
    THEN
        RETURN (possible_selection);
    ENDIF;
ENDIF;
  .
  .
  .
RETURN (0);                     ! Indicates failure.
ENDPROCEDURE;
```

(1) DECwindows allows you to designate more than one global selection. The two most common global selections are the primary and secondary selections. A global selection can be owned by only one DECwindows application at a time.

The GET_GLOBAL_SELECT built-in returns the data for the requested selection in the requested format. If the requested selection is not currently owned by any application, or if the owner cannot return it in the requested format, then GET_GLOBAL_SELECT returns unspecified.

If the selected information contains multiple records, the records are separated by the line-feed character (ASCII (10)).

B.6 Resizing Windows

Example B–5 shows the procedure SAMPLE_NEW_SCREEN_SIZE, which manipulates visible windows when the user makes the screen smaller. It removes visible VAXTPU windows from the screen, starting at the bottom of the VAXTPU screen, until the combined length of the remaining windows is less than or equal to the new smaller screen size or until there is only one window left. (For more information about the difference between VAXTPU windows and DECwindows windows, see Chapter 4.) If only one window remains, the procedure adjusts the window to fit the screen. If two or more windows remain, the procedure adjusts the current window and the bottom window.

The procedure uses the following variants of the built-in GET_INFO (window_variable):

- GET_INFO (window_variable, "bottom")
- GET_INFO (window_variable, "length")
- GET_INFO (window_variable, "top")

This example uses the following global variables and procedure from EVE:

- EVE$GET_WINDOW — A procedure that returns the window associated with a number. The windows are numbered sequentially, from top to bottom.
- EVE$X_NUMBER_OF_WINDOWS — A global variable that holds the count of the visible windows.
- EVE$$GET_WINDOW_NUMBER — A procedure that returns a number for the current window. EVE associates a value with each window so EVE can save information about specific windows. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.
- EVE$$REMOVE_WINDOW — A procedure that removes a window from the screen. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.

Example B–5 Procedure That Resizes Windows

```
PROCEDURE sample_new_screen_size

LOCAL  overhead,
       new_screen_length,
       number,
       the_count,
       total_length,
       some_window,
       a_window,
       new_top,
       a_length,
       top_adjust,
       bottom_window,
       bottom_adjust;


overhead := 2;  ! This provides lines for the command window and message
                ! window, assuming each window has a length of 1.

❶ new_screen_length := get_info (SCREEN, "new_length");

number := eve$$get_window_number;       ! This sets "number" to be
                                        ! the number of the current window.

the_count := eve$x_number_of_windows;   ! This sets "the_count" to
                                        ! be the total number of
                                        ! visible windows.


! The following lines determine the combined lengths of all
! user-created  windows visible on the screen, plus the lengths of the
! command window and message window.

total_length := overhead;
the_count := eve$x_number_of_windows;
LOOP
  EXITIF the_count < 1;
  some_window := eve$get_window (the_count);

  ! "Some_window" is the bottommost window not yet measured.

  total_length := total_length +
❷                 GET_INFO (some_window, "length", WINDOW);

  the_count := the_count - 1;

ENDLOOP;


        ! The following statements delete windows from the screen, starting
        ! with the bottommost window, until the sum of the lengths
        ! of all remaining windows is less than or equal to the new screen
        ! length.
the_count := eve$x_number_of_windows;
LOOP
    EXITIF the_count <= 1;
    a_window := eve$get_window (the_count);  ! This statement sets "a_window" to
                                             ! be the bottommost
                                             ! window not yet examined
                                             ! in this loop.
```

(continued on next page)

Example B–5 (Cont.) Procedure That Resizes Windows

```
❸   IF number > the_count

    THEN
❹       new_top := GET_INFO (a_window, "top", WINDOW);

    ENDIF;

    IF number <> the_count  ! If the current window is still
                            ! above the window to which you're
                            ! comparing it
    THEN
        a_length := GET_INFO (a_window, "length", WINDOW);


           ! The following clause prevents the loop from deleting
           ! the bottom window if the new screen length
           ! is greater than or equal to the old screen length.

        IF new_screen_length > total_length


            ! The following statement decreases "total_length" by the length
            ! of the window being examined.

        THEN
            total_length := total_length - a_length;

            ! The following statement removes the window
            ! being examined.

            eve$$remove_window (the_count);


        ENDIF;

        EXITIF total_length <= new_screen_length;

    ENDIF;

    the_count := the_count - 1; ! Next time through the loop, the window
                                ! being examined will be the window
                                ! just above the window examined this time.
ENDLOOP;


IF eve$x_number_of_windows = 1
THEN
    adjust_window (CURRENT_WINDOW,
                   1 - get_info (CURRENT_WINDOW, "top", WINDOW),
                   new_screen_length - overhead
❺                  -get_info (CURRENT_WINDOW, "bottom", WINDOW));
ELSE

     ! The following statements adjust the top of the current
     ! window and the bottom of the bottom window, if needed,
     ! to occupy the space left by deleting windows.
```

(continued on next page)

Example B–5 (Cont.) Procedure That Resizes Windows

```
    IF new_top <> 0
    THEN
        top_adjust := new_top - GET_INFO (CURRENT_WINDOW,
                                          "top", WINDOW);
        ADJUST_WINDOW (CURRENT_WINDOW, top_adjust, 0);
    ENDIF;

    bottom_window := eve$get_window (eve$x_number_of_windows);
    bottom_adjust := new_screen_length -
                     overhead -
                     GET_INFO (bottom_window,      ! This statement using
                               "bottom", WINDOW);  ! GET_INFO (window, "bottom")
                                                   ! calculates the amount
                                                   ! by which to adjust the
                                                   ! bottom of the bottom
                                                   ! window.

    ADJUST_WINDOW (bottom_window, 0, bottom_adjust);
ENDIF;

ENDPROCEDURE;
```

❶ GET_INFO (SCREEN "new_length") returns the size of the screen after a resize occurs.

❷ GET_INFO (window, "length", WINDOW) returns the length of the window.

❸ *Number* is greater than *the_count* only when the current window is below the window to which you are comparing it.

❹ GET_INFO (window, "top", WINDOW) returns the top line of the window.

❺ GET_INFO (window, "bottom", WINDOW) returns the line number of the last line in the window.

B.7 Unmapping Saved Windows

Example B–6 shows the procedure SAMPLE_SAVE_WINDOW_INFO_AND_UNMAP, which saves information about all visible VAXTPU windows in the array *window_array* and then unmaps all visible VAXTPU windows. The windows can be reconstructed later using the information in *window_array*.

The procedure uses the following variants of the built-in GET_INFO (window_variable):

- GET_INFO (window_variable, "width")
- GET_INFO (window_variable, "key_map_list")

- GET_INFO (window_variable, "scroll_bar", VERTICAL)
- GET_INFO (window_variable, "scroll_bar_auto_thumb", VERTICAL)

Warning: Digital does not guarantee that this example will work successfully with future versions.

Example B–6 EVE Procedure That Unmaps Saved Windows

```
PROCEDURE sample_save_window_info_and_unmap (; window_array)

LOCAL   the_count,
        the_window,
        saved_buffer,
        the_row,
        temp;

ON_ERROR
    [TPU$_CONTROLC]:
        IF GET_INFO (saved_buffer, "type") = BUFFER
        THEN
            eve$message (EVE$_REBLDWINDOWS);
            eve$setup_windows (saved_buffer);
            eve$message (EVE$_WINDOWSREBLT);
            UPDATE (ALL);
        ENDIF;
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
ENDON_ERROR;

the_count := 0;
the_window := GET_INFO (WINDOWS, "first");
LOOP
    EXITIF the_window = 0;
    IF GET_INFO (the_window, "buffer") <> 0
    THEN
        the_count := the_count + 1;
    ENDIF;
    the_window := GET_INFO (WINDOWS, "next");
ENDLOOP;
window_array := CREATE_ARRAY (the_count + 1, 0);
window_array {0} := eve$x_number_of_windows;
the_window := eve$main_window;
IF GET_INFO (the_window, "type") = WINDOW
THEN
    saved_buffer := GET_INFO (the_window, "buffer");
ENDIF;
```

(continued on next page)

Example B–6 (Cont.) EVE Procedure That Unmaps Saved Windows

```
LOOP
    EXITIF the_count = 0;
    the_window := CURRENT_WINDOW;
    EXITIF the_window = 0;
    temp := CREATE_ARRAY (29);
    temp {1} := the_window;
    temp {2} := GET_INFO (the_window, "buffer");
    temp {3} := GET_INFO (the_window, "top", WINDOW);
    temp {4} := GET_INFO (the_window, "length", WINDOW);
    temp {8} := get_info (the_window, "status_line");
    IF temp {8} <> 0
    THEN
        temp {5} := ON;
    ELSE
        temp {5} := OFF;
    ENDIF;
    POSITION (the_window);
    temp {6} := MARK (FREE_CURSOR);
    the_row := GET_INFO (the_window, "current_row");
    IF the_row = 0
    THEN
        the_row := temp {3};
    ENDIF;
    temp {7} := the_row + 1 - temp {3};
    temp {9} := GET_INFO (the_window, "width");     ! This statement uses
                                                    ! GET_INFO (window, "width").

    temp {10} := GET_INFO (the_window, "scroll_top");
    temp {11} := GET_INFO (the_window, "scroll_bottom");
    temp {12} := GET_INFO (the_window, "scroll_amount");
    temp {13} := GET_INFO (the_window, "text");
    temp {14} := GET_INFO (the_window, "blink_video");
    temp {15} := GET_INFO (the_window, "blink_status");
    temp {16} := GET_INFO (the_window, "bold_video");
    temp {17} := GET_INFO (the_window, "bold_status");
    temp {18} := GET_INFO (the_window, "reverse_video");
    temp {19} := GET_INFO (the_window, "reverse_status");
    temp {20} := GET_INFO (the_window, "underline_video");
    temp {21} := GET_INFO (the_window, "underline_status");
    temp {22} := GET_INFO (the_window, "special_graphics_status");
    IF GET_INFO (the_window, "pad")
    THEN
        temp {23} := ON;
    ELSE
        temp {23} := OFF;
    ENDIF;
    temp {24} := GET_INFO (the_window,
                           "shift_amount");
    temp {25} := GET_INFO (the_window,      ! This statement uses
                           "key_map_list"); ! GET_INFO (window, "key_map_list").
```

(continued on next page)

Example B–6 (Cont.) EVE Procedure That Unmaps Saved Windows

```
    IF GET_INFO (SCREEN, "decwindows")
    THEN
        temp {26} := (GET_INFO (the_window,    ! This statement uses
                                "scroll_bar", ! GET_INFO (window, "scroll_bar").
                                VERTICAL) <> 0);
                                                    ! If the vertical
                                                    ! scroll bar is
                                                    ! on, save the
                                                    ! information.

        IF temp {26}
        THEN
            temp {27} := GET_INFO (the_window,              ! This statement uses
                                   "scroll_bar_auto_thumb", ! the GET_INFO
                                   VERTICAL);               ! ("scroll_bar_auto_thumb)
                                                            ! built-in.
        ELSE
            temp {27} := FALSE;
        ENDIF;
        temp {28} := (GET_INFO (the_window, "scroll_bar", HORIZONTAL) <> 0);
        IF temp {28}
        THEN
            temp {29} := GET_INFO (the_window, "scroll_bar_auto_thumb",
                                   HORIZONTAL);
        ELSE
            temp {29} := FALSE;
        ENDIF;
    ELSE
        temp {26} := FALSE;
        temp {27} := FALSE;
        temp {28} := FALSE;
        temp {29} := FALSE;
    ENDIF;
    window_array {the_count} := temp;
    UNMAP (the_window);
    the_count := the_count - 1;
ENDLOOP;
eve$x_number_of_windows := 0;

ENDPROCEDURE;
```

B.8 Mapping Saved Windows

Example B–7 shows the procedure SAMPLE_MAP_SAVED_WINDOWS, which maps windows whose descriptions have been saved previously. SAMPLE_MAP_SAVED_WINDOWS is passed the array *window_array* containing information about windows that have previously been saved and then unmapped. You can see an example of how such an array is created in Example B–6. The procedure maps the windows to buffers, giving the windows the same characteristics they had before they were unmapped.

The procedure includes the following built-ins:

- SET (SCROLL_BAR)

- SET (SCROLL_BAR_AUTO_THUMB)
- SET (WIDGET)
- SET (WIDGET_CALLBACK)

Warning: Digital does not guarantee that this example will work successfully with future versions.

Example B–7 Procedure That Maps Saved Windows

```
PROCEDURE sample_map_saved_windows (window_array)

LOCAL   temp,
        the_length,
        length_remaining,
        the_top,
        the_count,
        scroll_bar_widget,
        screen_length;


ON_ERROR
    [TPU$_CONTROLC]:
        eve$message (EVE$_RESETUPWINDOWS);
        eve$setup_windows (window_array);
        UPDATE (ALL);
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
endon_error;

screen_length := eve$get_screen_height;

eve$$unmap_all_windows;

        eve$x_number_of_windows := window_array {0};
        the_count := 1;
        LOOP
            EXITIF the_count > GET_INFO (window_array, "high_index");
            temp := window_array {the_count};
            eve$$map_window (temp {1}, temp {2}, temp {3}, temp {4}, temp {5},
                             temp {6}, temp {7});
            IF temp {5} = ON
            THEN
                SET (STATUS_LINE, temp {1}, NONE, temp {8});
                IF temp {15}
                THEN
                    SET (STATUS_LINE, temp {1}, BLINK, temp {8});
                ENDIF;
                IF temp {17}
                THEN
                    SET (STATUS_LINE, temp {1}, BOLD, temp {8});
                ENDIF;
                IF temp {19}
                THEN
                    SET (STATUS_LINE, temp {1}, REVERSE, temp {8});
                ENDIF;
                IF temp {21}
                THEN
                    SET (STATUS_LINE, temp {1}, UNDERLINE, temp {8});
                ENDIF;
```

(continued on next page)

Example B–7 (Cont.) Procedure That Maps Saved Windows

```
            IF temp {22}
            THEN
                SET (STATUS_LINE, temp {1}, SPECIAL_GRAPHICS, temp {8});
            ENDIF;
        ENDIF;
        SET (WIDTH, temp {1}, temp {9});
        SET (TEXT, temp {1}, temp {13});
        IF temp {14}
        THEN
            SET (VIDEO, temp {1}, BLINK);
        ENDIF;
        IF temp {16}
        THEN
            SET (VIDEO, temp {1}, BOLD);
        ENDIF;
        IF temp {18}
        THEN
            SET (VIDEO, temp {1}, REVERSE);
        ENDIF;
        IF temp {20}
        THEN
            SET (VIDEO, temp {1}, UNDERLINE);
        ENDIF;
        SET (PAD, temp {1}, temp {23});
        SHIFT (temp {1}, temp {24});
        the_count := the_count + 1;
    ENDLOOP;
    IF GET_INFO (temp {25}, "type") = STRING
    THEN
        SET (KEY_MAP_LIST, temp {25}, temp {1});
    ENDIF;
    IF GET_INFO (SCREEN, "decwindows")
    THEN
        IF temp {26}
        THEN

            scroll_bar_widget := SET (SCROLL_BAR,     ! This statement
                                      temp {1},       ! uses the
                                      VERTICAL, ON);  ! SET (SCROLL_BAR)
                                                      ! built-in.

            SET (WIDGET_CALLBACK,           ! This statement uses the
                 scroll_bar_widget,         ! SET (WIDGET_CALLBACKS)
                 "eve$scroll_dispatch",     ! built-in.
                 'v');

            SET (WIDGET,                            ! This statement uses
                 scroll_bar_widget,                 ! the SET (WIDGET)
                 eve$$scroll_bar_callbacks);        ! built-in.

            eve$$scroll_bar_window (scroll_bar_widget) := temp {1};
            IF temp {27}

            THEN

                SET (SCROLL_BAR_AUTO_THUMB,    ! This statement uses the
                     temp {1}, VERTICAL, ON);  ! SET (SCROLL_BAR_AUTO_THUMB)
                                               ! built-in.
            ENDIF;
```

(continued on next page)

Example B–7 (Cont.) Procedure That Maps Saved Windows

```
            ENDIF;
            IF temp {28}
            THEN
                scroll_bar_widget := SET (SCROLL_BAR, temp {1}, HORIZONTAL,
                                          ON);
                SET (WIDGET_CALLBACK, scroll_bar_widget, "eve$scroll_dispatch",
                     'h');
                SET (WIDGET, scroll_bar_widget, eve$$scroll_bar_callbacks);
                eve$$scroll_bar_window (scroll_bar_widget) := temp {1};
                IF temp {29}
                THEN
                    SET (SCROLL_BAR_AUTO_THUMB, temp {1}, HORIZONTAL, ON);
                ENDIF;
            ENDIF;
        ENDIF;
        UPDATE (ALL);
        the_count := 1;
        LOOP
            EXITIF the_count > GET_INFO (window_array, "high_index");
            temp := window_array {the_count};
            SET (SCROLLING, temp {1}, ON, temp {10}, temp {11}, temp {12});
            the_count := the_count + 1;
        ENDLOOP;
        SET (PROMPT_AREA, screen_length - 1, 1, REVERSE);

ENDPROCEDURE;
```

B.9 Handling Callbacks from a Scroll Bar Widget

Example B–8 shows one of the ways an application can use the statements POSITION (integer) and SET (WIDGET). The procedure is a portion of the EVE procedure *eve$scroll_dispatch*. You can find the original version in SYS$EXAMPLES:EVE$DECWINDOWS.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

The procedure *eve$scroll_dispatch* is the callback routine handling callbacks from scroll bar widgets. The portion of the procedure shown here determines where to position the editing point based on how the user has changed the scroll bar slider. The procedure fetches the position of the slider with the built-in GET_INFO (widget_variable, "widget_info") and positions the editing point to the line in the buffer equivalent to the slider's position in the scroll bar. Finally, the procedure updates the scroll bar's resource values. For more information about the resource names used with the scroll bar widget, see the *VMS DECwindows Toolkit Routines Reference Manual*.

EVE uses the following constants in this procedure:

- EVEDWTC_NINC — A constant for the string "inc". This is the resource name for the amount that the scroll bar slider position is to be incremented or decremented when a scroll bar button is pressed.

- EVEDWTC_NPAGE_INC — A constant for the string "pageInc". This is the resource name for the amount that the scroll bar slider position is to be incremented or decremented when a click occurs within the scroll bar above or below the slider.
- EVEDWTC_NMAX_VALUE — A constant for the string "maxValue". This is the resource name for the maximum value of the scroll bar slider position.
- EVEDWTC_NMIN_VALUE — A constant for the string "minValue". This is the resource name for the minimum value of the scroll bar slider position.
- EVEDWTC_NVALUE — A constant for the string "value". This is the resource name for the top of the scroll bar slider position.
- EVEDWTC_NSHOWN — A constant for the string "shown". This is the resource name for the size of the slider.
- EVEDWTC_CRVALUE_CHANGE_CALLBACK — A constant for the callback reason code DWT$C_CR_VALUE_CHANGED. This reason code indicates that the user changed the value of the scroll bar slider.
- EVE$K_CLOSURE — A constant for the string "closure", used as an index for the array returned by GET_INFO (WIDGET, "callback_parameters", array).
- EVE$K_REASON_CODE — A constant for the string "reason_code", used as an index for the array returned by GET_INFO (WIDGET, "callback_parameters", array).
- EVE$K_WIDGET — A constant for the string "widget", used as an index for the array returned by GET_INFO (WIDGET, "callback_parameters", array).

Example B–8 EVE Procedure That Handles Callbacks from a Scroll Bar Widget

```
PROCEDURE eve$scroll_dispatch
LOCAL   status,
        widget_called,
        widget_tag,
        widget_reason,
        scroll_bar_values,
        linenum,
        temp_array,
           .
           .
           .;

ON_ERROR
    [TPU$_CONTROLC]:
        eve$learn_abort;
        ABORT;
ENDON_ERROR

❶ status := GET_INFO (WIDGET, "callback_parameters", temp_array);
```

(continued on next page)

Example B–8 (Cont.) EVE Procedure That Handles Callbacks from a Scroll Bar Widget

```
widget_called := temp_array {eve$k_widget};
widget_tag := temp_array {eve$k_closure};
widget_reason := temp_array {eve$k_reason_code};

POSITION (eve$$scroll_bar_window {widget_called});
   .
   .
   .

scroll_bar_values := CREATE_ARRAY;
scroll_bar_values {eve$dwt$c_ninc} := 0;
scroll_bar_values {eve$dwt$c_npage_inc} := 0;
scroll_bar_values {eve$dwt$c_nmax_value} := 0;
scroll_bar_values {eve$dwt$c_nmin_value} := 0;
scroll_bar_values {eve$dwt$c_nvalue} := 0;
scroll_bar_values {eve$dwt$c_nshown} := 0;

❷ status := GET_INFO (widget_called, "widget_info", scroll_bar_values);

! The deleted statements scroll the window as dictated
! by the callback reason.

   .
   .
   .

CASE widget_reason
   .
   .
   .

        [eve$dwt$c_crvalue_change_callback]:

           IF (scroll_bar_values {eve$dwt$c_nvalue} =
               scroll_bar_values {eve$dwt$c_nmin_value})
           THEN
               POSITION (beginning_of (current_buffer));
           ELSE
❸              POSITION (scroll_bar_values {eve$dwt$c_nvalue});

            ENDIF;
                  .
                  .
                  .

scroll_bar_values {eve$dwt$c_ninc} := 1;
scroll_bar_values {eve$dwt$c_npage_inc} := scroll_bar_values {eve$dwt$c_nshown}
                                           - 1;

❹ SET (WIDGET, widget_called, scroll_bar_values);

!      .
!      .
!      .

ENDPROCEDURE;
```

❶ GET_INFO (WIDGET, "callback_parameters", array) returns an array containing the values for the current callback. The array elements are indexed by the strings "widget", "closure", and "reason_code" that

reference the widget that is calling back, the widget's closure value, and the reason code for the callback.

❷ GET_INFO (WIDGET, "widget_info", array) allows you to fetch information from a widget. The array parameter is indexed by the resource names associated with the specified widget. Note that resource names are case sensitive. Note, too, that the set of supported resources varies from one widget type to another. When you use GET_INFO (widget, "widget_info", array), VAXTPU queries the widget for the requested information and puts the returned informaion in the array elements. Any previous values in the array are lost.

❸ POSITION (integer) allows you to move the editing point to the record specified by the parameter *integer*. VAXTPU interprets this parameter as a record number.

❹ SET (WIDGET, widget_variable, array) allows you to set resource values for the specified widget. The array parameter is indexed by the resource names associated with the specified widget. Note that resource names are case sensitive. Note, too, that the set of supported resources varies from one widget type to another.

B.10 Implementing the COPY SELECTION Operation

Example B–9 shows one of the ways an application can use the READ_GLOBAL_SELECT built-in. The procedure is a modified version of the EVE procedure EVE$STUFF_GLOBAL_SELECTION. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

The procedure performs the following tasks:

- Saves the location of the editing point and the buffer's current mode.
- Checks that DECwindows EVE is enabled and that EVE does not have input focus.
- Obtains the location of the pointer cursor and positions the editing point at that location.
- Sets the text mode to insert.
- Reads the string-formatted contents of the primary global selection. (In this context, the parameter *"STRING"* means that the calling application is asking the application that owns the global selection for the string-formatted information in the specified global selection.)
- Restores the editing point location and text mode to their previous values.

EVE binds this procedure to the MB3 key. Thus, using MB3, the user can direct EVE to copy selected material from a non-EVE DECwindows application to a DECwindows EVE buffer. In DECwindows documentation, this operation is called COPY SELECTION.

Example B–9 EVE Procedure That Implements the COPY SELECTION Operation

```
PROCEDURE eve$stuff_global_selection

LOCAL   saved_position,
        saved_mode,
        this_buffer,
        the_window,
        the_column,
        the_row;


ON_ERROR
    [TPU$_CONTROLC]:
        IF saved_mode = OVERSTRIKE
        THEN
            SET (saved_mode, this_buffer);
        ENDIF;
        eve$$restore_position (saved_position);
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
        IF saved_mode = OVERSTRIKE
        THEN
            SET (saved_mode, this_buffer);
        ENDIF;
        eve$$restore_position (saved_position);
ENDON_ERROR;


this_buffer := current_buffer;
saved_position := MARK (FREE_CURSOR);
saved_mode := GET_INFO (this_buffer, "mode");

IF eve$x_decwindows_active
THEN
    IF not GET_INFO (SCREEN, "input_focus")
    THEN
        IF LOCATE_MOUSE (the_window,              ! This statement uses
                         the_column, the_row)     ! the LOCATE_MOUSE built-in.
        THEN
            IF the_row <> 0
                IF the_window <> eve$choice_window
                THEN
                    POSITION (MOUSE);
                    SET (INSERT, this_buffer);

                    READ_GLOBAL_SELECT (PRIMARY, "STRING");  ! This statement
                                                             ! using READ_GLOBAL_SELECT
                                                             ! reads the string-
                                                             ! formatted contents
                                                             ! of the primary
                                                             ! global selection.
```

(continued on next page)

Example B–9 (Cont.) EVE Procedure That Implements the COPY SELECTION Operation

```
                    eve$$restore_position (saved_position);
                    SET (saved_mode, this_buffer);
                    UPDATE (CURRENT_WINDOW);
                    RETURN (TRUE);
                ENDIF;
            ENDIF;
        ENDIF;
    ENDIF;
ENDIF;

RETURN (FALSE);

ENDPROCEDURE;
```

B.11 Reactivating a Select Range

Example B–10 shows one of the ways an application can use the SET (GLOBAL_SELECT) built-in. The procedure is a modified version of the EVE procedure EVE$RESTORE_PRIMARY_SELECTION. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

The procedure *eve$restore_primary_selection* reactivates EVE's select range when EVE regains input focus.

Example B–10 EVE Procedure That Reactivates a Select Range

```
PROCEDURE eve$restore_primary_selection

LOCAL   saved_position;

ON_ERROR
    [TPU$_CONTROLC]:
        eve$$restore_position (saved_position);
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
        eve$$restore_position (saved_position);
ENDON_ERROR;


IF NOT eve$x_decwindows_active
THEN
    RETURN (FALSE);
ENDIF;

saved_position := MARK (FREE_CURSOR);
```

(continued on next page)

Example B–10 (Cont.) EVE Procedure That Reactivates a Select Range

```
IF GET_INFO (eve$$x_save_select_array, "type") = ARRAY
THEN
    CASE eve$$x_save_select_array {"type"}
        [RANGE]:
            eve$select_a_range (eve$$x_save_select_array {"start"},
                                eve$$x_save_select_array {"end"});
            eve$$x_state_array {eve$$k_select_all_active} :=
                                                    eve$$x_save_select_array
                                                        {"select_all"};
            POSITION (eve$$x_save_select_array {"current"});
            eve$start_pending_delete;
        [MARKER]:
            POSITION (eve$$x_save_select_array {"start"});
            eve$x_select_position := select (eve$x_highlighting);
            POSITION (eve$$x_save_select_array {"end"});
            eve$start_pending_delete;
        [OTHERWISE]:
            RETURN (FALSE);
    ENDCASE;


    eve$$restore_position (saved_position);
    eve$$found_post_filter;             ! This is necessary if the
                                        ! cursor is outside the selection.

    eve$$x_save_select_array {"type"} := 0;
    UPDATE (current_window);
    IF eve$x_decwindows_active
    THEN

        SET (GLOBAL_SELECT, SCREEN, PRIMARY);  ! This statement using
                                               ! SET (GLOBAL_SELECT)
                                               ! requests ownership of
                                               ! the primary global selection.

    ENDIF;
    RETURN (TRUE);
ENDIF;
RETURN (FALSE);

ENDPROCEDURE;
```

B.12 Copying Selected Material from EVE to Another DECwindows Application

Example B–11 shows one of the ways a layered application can use the WRITE_GLOBAL_SELECT built-in. The procedure is a modified version of the EVE procedure EVE$WRITE_GLOBAL_SELECT. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section B.1.

The procedure implements the operation of copying selected material from DECwindows EVE to another DECwindows application. in DECwindows documentation, this operation is called COPY SELECTION.

The procedure determines what property of the primary global selection is being requested, obtains the value of the appropriate property using a GET_INFO statement or an EVE procedure, and sends the information to the requesting application.

Example B–11 EVE Procedure That Implements COPY SELECTION

```
PROCEDURE eve$write_global_select        ! EVE uses this procedure
                                         ! to respond to requests
                                         ! for information about
                                         ! selections.

LOCAL   saved_position,
        the_data,
        temp_array,
        total_lines,
        the_line,
        status,
        eob_flag,
        percent;


ON_ERROR
    [OTHERWISE]:
        eve$$restore_position (saved_position);
ENDON_ERROR;

saved_position := MARK (FREE_CURSOR);

IF NOT eve$x_decwindows_active

THEN
    RETURN (FALSE);
ENDIF;


the_data := "";
temp_array := GET_INFO (SCREEN, "event", GLOBAL_SELECT);

    ! Finds out which global selection and which property
    ! of the global selection are the subject of the
    ! information request.


CASE temp_array {2}      ! Determines the property requested by the other application.

    ["STRING", "TEXT"]: ! If one of these strings is requested, the
                        ! procedure sends the text in the global
                        ! selection to the requesting application.

        CASE temp_array {1}  ! Checks which global selection was specified.
            [PRIMARY]:

                IF eve$x_select_position <> 0
                THEN
                    POSITION (GET_INFO (eve$x_select_position, "buffer"));
```

(continued on next page)

Example B–11 (Cont.) EVE Procedure That Implements COPY SELECTION

```
                    IF GET_INFO (eve$x_select_position, "type") = RANGE
                    THEN
                        the_data := STR (eve$x_select_position);
                    ELSE
                        IF GET_INFO (eve$x_select_position, "type") = MARKER
                        THEN
                            the_data := STR (eve$select_a_range (eve$x_select_position,
                                                                 MARK (FREE_CURSOR)));
                        ELSE
                            the_data := NONE;
                        ENDIF;
                    ENDIF;
                    eve$$restore_position (saved_position);
                ENDIF;
            [OTHERWISE]:
                the_data := NONE;
        ENDCASE;
    [OTHERWISE]:
        the_data := NONE;       ! The procedure does not send data if
                                ! the requesting application has asked
                                ! for something other than the text,
                                ! the file name, or the line number.
ENDCASE;

WRITE_GLOBAL_SELECT (the_data);         ! This statement sends the
                                        ! requested information to
                                        ! the requesting application.

ENDPROCEDURE;
```

C VAXTPU Terminal Support

This appendix lists the terminals that support screen-oriented editing and describes how differences among these terminals affect the way VAXTPU performs. This appendix also describes how VAXTPU can be run on terminals that do not support screen-oriented editing. Finally, this appendix tells you how VAXTPU manages wrapping and how you can modify that.

C.1 Screen-Oriented Editing on Supported Terminals

VAXTPU supports screen-oriented editing only on terminals that respond to ANSI control functions and that operate in ANSI mode. By default, your VAXTPU session runs with the screen management file TPU$CCTSHR.EXE. To check your terminal setting, enter the command SHOW TERMINAL at the DCL level.

VAXTPU screen-oriented editing is designed to optimize the features available with the Digital VT300 and VT200 families of terminals and the Digital VT100 family of terminals. VAXTPU does not support screen-oriented editing on Digital VT52-compatible terminals. Optimum VAXTPU performance is achieved on the VT300-series, VT200-series, and VT100-series terminals. Some of the high-performance characteristics of VAXTPU may not be apparent on the terminals listed in Table C–1 for the reasons stated.

Table C–1 Terminal Behavior That Affects VAXTPU's Performance

Terminal	Characteristic
VT102	Slow autorepeat rate
VT240	Slow autorepeat rate Slower scrolling region setup time than the VT220.
GIGI	One form of scrolling region (VAXTPU repaints screen, rather than use this scrolling mechanism) Variable autorepeat rate (cursor keys pick up speed when used repeatedly)

C.1.1 Terminal Settings That Affect VAXTPU

The following settings may affect the behavior of VAXTPU, depending on the terminal that you use.

132-Column Mode

Only terminals that set the DEC_CRT mode bit and the advanced video mode bit can alter their physical width from 80 columns to 132 and back. All other terminals keep the physical width that is set when you enter the editor.

For the VAXTPU screen manager to behave predictably on GIGI terminals, you should report the terminal width as 84 to VMS. Use the DCL command SET TERMINAL/DEVICE=VK100 to set the proper terminal width.

Autorepeat ON/OFF and Auxiliary Keypad Enabling

To take advantage of the built-in procedure SET (AUTO_REPEAT) or to enable the auxiliary keypad for applications mode, the terminal must be set to DEC_CRT3, DEC_CRT2, DEC_CRT, or VK100. Use the DCL command SET TERMINAL/DEVICE=characteristic to set the terminal.

Control Sequence Introducer

A feature of VAXTPU is that it can use one 8-bit control sequence introducer (CSI) to introduce a terminal control sequence. (Normally, the 2-character combination of the ESCAPE key and the left bracket ([) is used.) To take advantage of this feature, set your terminal to DEC_CRT2 mode. The Digital VT300-series and VT220 and VT240 terminals currently support this feature.

Cursor Positioning

If your terminal sets the DEC_CRT mode bit, VAXTPU assumes that when control sequences that position the cursor to row 1 or column 1 are sent to the terminal, the 1 can be omitted. If your terminal does not behave correctly when it receives these control sequences, you must turn off the DEC_CRT mode bit. Some foreign terminals may not be fully compatible with VAXTPU and may exhibit this behavior.

Edit Mode

Terminals that are operating in edit mode allow the editor to take advantage of special edit-mode control sequences during deletion and insertion of text for optimization purposes. Some current Digital terminals that support edit mode include the VT102, the VT220, the VT240, the VT241, and VT300-series terminals.

8-Bit Characters

ANSI terminals operating in 8-bit mode have the ability to use the supplemental characters and control sequences in the DEC Multinational Character Set. The Digital VT300 series and the VT220 and VT240 terminals currently support 8-bit character mode. If you have the 8-bit mode bit set, VAXTPU designates the DEC Multinational Character Set into G2 and invokes it into GR. For more information on how your terminal interacts with the DEC Multinational Character Set, refer to the programming manual for your specific terminal.

Scrolling

Scrolling regions are only used for terminals that have the DEC_CRT mode bit set. On other terminals, VAXTPU repaints the window when a scroll would have been used (for example, when a line is deleted or inserted).

Video Attributes

When you set the video attributes of windows, markers, or ranges, only those attributes supported by your terminal type give predictable results. Most ANSI CRTs support reverse video. However, only terminals that support DEC_CRT mode with the advanced video option (AVO) have the full range of video attributes (reverse, bold, blink, underline) that VAXTPU supports.

C.1.2 The DCL Command SET TERMINAL

When you use the DCL command SET TERMINAL to specify characteristics for your terminal, make sure to set only those characteristics that are supported by your terminal. If you set characteristics that the terminal does not support, the screen-oriented functions of VAXTPU may behave unpredictably. For example, if you run VAXTPU on a VT100 terminal and you set the DEC_CRT2 characteristic that VT100s do not support, VAXTPU tries to use 8-bit CSI controls. This could cause ";7m" to appear on the screen where the reverse video attribute should be set.

Most users do not knowingly set characteristics that are not supported by their terminals. However, if you temporarily move to a different type of terminal, your LOGIN.COM file may have characteristics set for your usual terminal that do not apply to the current terminal. This problem may also occur if, before running VAXTPU, you run a program that modifies your terminal characteristics without your knowledge.

If you see unexpected video attributes or extraneous characters on the screen, exit from VAXTPU and check your terminal characteristics with the DCL command SHOW TERMINAL.

Recover your files using the same terminal characteristics with which your files were created. Otherwise, a journal file inconsistency may occur, depending on how your interface is written.

C.2 Line-Mode Editing on Unsupported Terminals

If you want to run VAXTPU from an unsupported terminal, you must inform VAXTPU that you do not want to use screen capabilities. To invoke VAXTPU on an unsupported terminal, use the qualifier /NODISPLAY after the command EDIT/TPU. See Chapter 5 for more information on this qualifier. While in no-display mode, VAXTPU uses the RTL generic LIB$PUT_OUTPUT routine to display prompts and messages at the current location in SYS$OUTPUT. By using a combination of the built-in procedures READ_LINE and MESSAGE, you can devise your own line-mode editing functions or perform editing tasks from a batch job. See the sample line-mode editor in Appendix A.

C.3 Terminal Wrap

If you have enabled an automatic wrap setting on your terminal, VAXTPU disables this setting in order to manage the screen more efficiently. When you exit from VAXTPU, VAXTPU restores all terminal characteristics to the setting of the DCL command SET TERMINAL before invoking VAXTPU. If the DCL command SET TERM/NOWRAP is active, VAXTPU leaves the hardware wrap off. However, if the DCL command SET TERM /WRAP is active, VAXTPU assumes that you want hardware wrap on, so it turns it on when you exit from VAXTPU.

If you do not want this behavior of VAXTPU, you can prevent VAXTPU from turning on hardware wrap by specifying SET TERM/NOWRAP before invoking VAXTPU. You can enter the command interactively, or you can write a DCL command procedure that makes this setting part of your VAXTPU environment. Example C–1 shows a DCL command procedure that is used to control this terminal setting before and after a VAXTPU session.

Example C–1 DCL Command Procedure for SET TERM/NOWRAP

```
$ SET TERM/NOWRAP
$ ASSIGN/USER SYS$COMMAND SYS$INPUT
$ EDIT/TPU/SECTION = EDTSECINI
$ SET TERM/WRAP
```

D VAXTPU Messages

This appendix presents the messages produced by VAXTPU. The messages are listed alphabetically by their abbreviations in Table D–1. The text of the message and its severity level appears with each abbreviation. For an explanation of the severity levels for messages, see the *VMS System Messages and Recovery Procedures Reference Manual.*

The *VMS System Messages and Recovery Procedures Reference Manual* also contains the VAXTPU messages, including the appropriate explanations of the messages and the suggested actions to recover from the errors which provoke the messages.

Table D–1 VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
ACCVIO	Access violation, reason mask= 'xx', virtual address='xxxxxxxx', PC='xxxxxxxx', PSL='xxxxxxxx'	FATAL
ADJSCROLLREG	Scrolling parameters altered to top: 'top', bottom: 'bottom', amount: 'amount'	INFORMATIONAL
AMBIGSYMUSED	Ambiguous symbol 'name' used as procedure parameter	INFORMATIONAL
ARGMISMATCH	Parameter 'number"s data type, 'type', unsupported	ERROR
ASYNCACTIVE	Journal file prohibited with asynchronous handlers declared	WARNING
ATLINE	At line 'integer'	INFORMATIONAL
ATPROCLINE	At line 'integer' of procedure 'name'	INFORMATIONAL
BADASSIGN	Target of the assignment cannot be a function/keyword	ERROR
BADBUFWRITE	Error occurred writing buffer 'buffer name'	WARNING
BADCASELIMIT	CASE constant outside CASE limits	ERROR
BADCASERANGE	Invalid CASE range	ERROR
BADCHAR	Unrecognized character in input	ERROR
BADDELETE	Cannot modify constant integer, keyword, or string	ERROR
BADDISPVAL	display value = 'integer', must be between 'integer' and 'integer'	WARNING
BADEXITIF	EXITIF occurs outside a LOOP	ERROR
BADFIRSTLINE	First line = 'integer', must be between 'integer' and 'integer'	WARNING
BADJOUCHAR	Expected character in journal file	WARNING
BADJOUCOM	Journaled command file was 'string', recovering with 'string'	ERROR
BADJOUCPOS	Journaled starting character was 'integer', recovering with 'integer'	ERROR
BADJOUEDIT	Journaled edit mode was 'string', recovering with 'string'	ERROR

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
BADJOUEIGHT	Journaled eightbit was 'string', recovering with 'string'	ERROR
BADJOUFILE	Operation terminated due to error in journal file access	ERROR
BADJOUINIT	Journaled init file was 'string', recovering with 'string'	ERROR
BADJOUINPUT	Journaled input file was 'string', recovering with 'string'	ERROR
BADJOUKEY	Expected key in journal file	WARNING
BADJOULINE	Journaled line editing was 'string', recovering with 'string'	ERROR
BADJOULPOS	Journaled starting line was 'integer', recovering with 'integer'	ERROR
BADJOUPAGE	Journaled page length was 'integer', recovering with 'integer'	ERROR
BADJOUSEC	Journaled section file was 'string', recovering with 'string'	ERROR
BADJOUSTR	Expected string in journal file	WARNING
BADJOUTERM	Journaled terminal type was 'string', recovering with 'string'	ERROR
BADJOUWIDTH	Journaled width was 'integer', recovering with 'integer'	ERROR
BADKEY	'keyword' is an invalid keyword	WARNING
BADLENGTHCHANGE	Terminal will not support change of length	WARNING
BADLOGIC	Internal logic error detected	FATAL
BADMARGINS	Margins specified incorrectly	WARNING
BADPROCNAME	Variable used as a procedure	ERROR
BADPROMPTLEN	Prompt area length = 'integer', must be between 'integer' and 'integer'	WARNING
BADREAD	Read next or read prev with current record of 0, dscb: 'address'	FATAL
BADREFCNT	Ref count: 'ccc', zap count: 'zzz', address: 'xxxxxxxx'	FATAL
BADREQUEST	Request "'name'" of 'name' is not understood	WARNING
BADSCREENWIDTH	Terminal must be wider than widest window, 'integer' columns	WARNING
BADSECTION	Bad section file	ERROR
BADSTATUS	Return status 'xxxxxxxx' different from last signal 'xxxxxxxx'	FATAL
BADSTRCNT	Invalid string count found in journal file	WARNING
BADSYMTAB	Bad symbol table	ERROR
BADUSERDESC	Descriptor from user routine invalid or memory inaccessible	ERROR
BADVALUE	Integer value 'integer' is outside specified limits	ERROR
BADWIDTHCHANGE	Terminal will not support change of width	WARNING
BADWINDADJUST	Attempt to make window less than 1 line long, no adjustment	WARNING
BADWINDLEN	Window length = 'integer', must be between 'integer' and 'integer'	WARNING

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
BEGOFBUF	Attempt to move past the beginning of buffer 'buffer name'	WARNING
BINARYOPER	Operand combination 'type' 'oper' 'type' unsupported	WARNING
BOTLINETRUNC	Calculated new last line 'integer', changed to 'integer'	INFORMATIONAL
BUILTININV	Builtin is invalid at this time	ERROR
CALLUSERFAIL	CALL_USER routine failed with status %X'status'	WARNING
CANCELQUIT	QUIT canceled by request	WARNING
CANNOTUNSEL	Cannot unselect item from unselect action routine	ERROR
CAPTIVE	Unable to create a subprocess in a captive account	WARNING
CLIPBOARDFAIL	Unexpected clipboard failure	WARNING
CLIPBOARDLOCKED	Clipboard is locked by another process	WARNING
CLIPBOARDNODATA	Clipboard does not contain the requested data	WARNING
CLIPBOARDZERO	Clipboard data has 0 length	WARNING
CLOSEDIC	Error closing the dictionary file	ERROR
CLOSEIN	Error closing input file 'file-spec'	ERROR
CLOSEOUT	Error closing output file 'file-spec'	ERROR
CNVERR	Error occurred in the conversion routine	ERROR
COMPILED	Compilation completed without errors	SUCCESS
COMPILEFAIL	Compilation aborted	WARNING
CONSTRTOOLARGE	Constant string too large	ERROR
CONTRADEF	Contradictory definition for variable or constant 'name'	ERROR
CONTROLC	Operation aborted by CTRL/C	ERROR
CREATED	'file-spec' created	SUCCESS
CREATEFAIL	Unable to activate subprocess	WARNING
DDIFIN	'count' line(s), 'count' frame(s) read from file 'name'	SUCCESS
DDIFOUT	'count' line(s), 'count' frame(s) written to file 'name'	SUCCESS
DEBUG	Breakpoint at line 'integer'	SUCCESS
DELETEFAIL	Unable to terminate subprocess	WARNING
DICADD	'word' has been added to a dictionary as 'word'	SUCCESS
DICDEL	'word' has been removed from 'word' of a dictionary	SUCCESS
DICUPDERR	Error updating dictionary file	ERROR
DIVBYZERO	Divide by zero	ERROR
DUPBUFNAME	Buffer 'name' already exists	WARNING
DUPKEYMAPLIST	Attempt to define a duplicate key-map list 'key-map-list-name'	WARNING
DUPKEYMAP	Attempt to define a duplicate key map 'key-map-name'	WARNING
EMPTYKMLIST	Key-map list 'key-map-list-name' does not contain any key maps	WARNING

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
ENDOFBUF	Attempt to move past the end of buffer 'buffer name'	WARNING
ENDOFLINE	Returning a range of text with an end of line	SUCCESS
ERRSYMACTIVE	Special error symbol already active	WARNING
EXECUTEFAIL	Execution aborted	WARNING
EXITFAIL	Attempt to EXIT was unsuccessful	WARNING
EXITING	Editor exiting	SUCCESS
EXPCLA	The current clause has been expanded	INFORMATIONAL
EXPCOMPLEX	Expression too complex	ERROR
EXPECTED	One of the following symbols was expected:	INFORMATIONAL
EXTNOTFOUND	Extension 'name' not found	ERROR
EXTRANEOUSARGS	One or more extraneous arguments specified	ERROR
FACTOOLONG	Facility name, 'name', exceeds maximum length 'integer'	WARNING
FAILURE	Internal VAXTPU failure detected at PC 'number'	FATAL
FAILURE_STATUS	Facility 'name' returned failure status of 'xxxxxxxx'	ERROR
FENCEPOST	No visible record found in specified range	WARNING
FILECONVERTED	File format is being converted to a supported type	ERROR
FILEIN	'count' line(s) read from file 'name'	SUCCESS
FILEOUT	'count' line(s) written to file 'name'	SUCCESS
FLAGTRUNC	Value of message flags exceeds maximum value 15, truncated	WARNING
FREEMEM	Memory deallocation failure	FATAL
FROMBUILTIN	Called from builtin 'name'	INFORMATIONAL
FROMLINE	Called from line 'integer'	INFORMATIONAL
FROMPROCLINE	Called from line 'integer' of procedure 'name'	INFORMATIONAL
GBLSELOWNER	You are the global selection owner	WARNING
GETMEM	Memory allocation failure (Insufficient virtual memory)	ERROR
HIDDEN	Global variable 'name' by declaration	INFORMATIONAL
IDMISMATCH	Section NOT restored, section file must be rebuilt	FATAL
ILLCONDIT	Illegal compilation conditional	ERROR
ILLEGALTYPE	Illegal data type	ERROR
ILLPATAS	Pattern assignment target only valid in procedure 'name'	ERROR
ILLREQUEST	Request "'name'" is invalid	WARNING
ILLSEVERITY	Illegal severity of 'value' specified, error severity used	WARNING
INBUILTIN	Occurred in builtin 'name'	INFORMATIONAL
INCKWDCOM	Inconsistent keyword combination	WARNING
INDEXTYPE	Array index data type, 'type', unsupported	WARNING

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
INPUT_CANCELED	Input request canceled	WARNING
INSVIRMEM	Insufficient virtual memory	FATAL
INVACCESS	Invalid file access specified	FATAL
INVBUFDELETE	Cannot delete a permanent buffer	WARNING
INVDEVTYPE	Invalid device type	FATAL
INVFAOPARAM	FAO parameter 'integer' must be string or integer	WARNING
INVGBLSELDATA	The selected data cannot be processed	WARNING
INVIOCODE	Invalid Operation Code passed to an I/O operation	ERROR
INVITEMCODE	Invalid item code specified in list	FATAL
INVNUMSTR	Invalid numeric string	WARNING
INVPARAM	Parameter 'number"s data type, 'type', illegal; expected 'type'	ERROR
INVRANGE	Invalid range enclosure specified	WARNING
INVTABSPEC	Tabs specification incorrect, not changed	WARNING
INVTIME	Invalid time interval	WARNING
JNLACTIVE	Asynchronous actions prohibited when journal file open	WARNING
JNLNOTOPEN	Journal file not open, recovery aborted	ERROR
JNLOPEN	Journal file already open	ERROR
JOURNALBEG	Journal of edit session started	INFORMATIONAL
JOURNALCLOSE	Journal file successfully closed, journaling stopped	SUCCESS
JOURNALEOF	End of journal file found unexpectedly	WARNING
JRNLBUFBEG	Journaling started for buffer 'buffer name'	INFORMATIONAL
JRNLNOTSAFE	Buffer 'buffer name' is not safe for journaling	WARNING
JRNLOPEN	Journal file already open for buffer 'buffer name'	WARNING
KEYMAPNOTFND	Key map 'key-map-name' not found in key-map list 'key-map-list-name'	WARNING
KEYSUPERSEDED	Definition of key 'name' superseded	INFORMATIONAL
KEYWORDPARAM	Keyword 'name' used as procedure/variable/constant	ERROR
LINENOTMOD	Attempt to change unmodifiable line(s)	WARNING
LINETOOLONG	Line is maximum length, cannot add text to it	WARNING
MAXMAPPEDBUF	A single buffer can be mapped to at most 'count' window(s)	WARNING
MAXVALUE	Maximum value is 'integer'	WARNING
MINVALUE	Minimum value is 'integer'	WARNING
MISSINGQUOTE	Missing quote	ERROR
MISSYMTAB	Missing symbol table	ERROR
MIXEDTYPES	Operator with mixed or unsupported data types	ERROR
MODRANGEMARKS	MODIFY_RANGE requires either two marks or none	ERROR

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
MOUSEINV	Mouse location information is invalid	WARNING
MOVETOCOPYTEXT	Moving unmodifiable line(s) from buffer 'string' changed to copy	WARNING
MOVETOCOPY	Move from unmodifiable buffer 'string' changed to copy	WARNING
MSGBUFSET	Attempt to change modifiable setting of message buffer	WARNING
MSGNOTFND	Message was not found; the default message has been returned	WARNING
MULTIDEF	Parameter/local/constant 'name' multiply defined	ERROR
MULTIPLENAMES	There is more than one name matching, all are returned	WARNING
MULTISELECT	Multiple identical CASE selectors	ERROR
MUSTBECONST	Expression must be a compile-time constant	ERROR
MUSTBEONE	String must be 1 character long	WARNING
NEEDFILENAME	Type filename for buffer 'name' (press RETURN to not write it):	SUCCESS
NEEDTOASSIGN	Built-in must return a value	ERROR
NOASSIGNMENT	Expression without assignment	ERROR
NOBREAKPOINT	No breakpoint is active	WARNING
NOCACHE	Insufficient virtual memory to allocate a new cache	ERROR
NOCALLUSER	Could not find a routine for CALL_USER to invoke	ERROR
NOCHARREAD	No character was read by the READ_CHAR builtin	WARNING
NOCLA	No conversion source has been specified yet	WARNING
NOCOPYBUF	Cannot COPY a buffer to itself	WARNING
NOCURRENTBUF	No buffer has been selected as default	WARNING
NODEFINITION	Key 'keyname' currently has no definition	WARNING
NODICENT	No entry found in a dictionary	WARNING
NODICUPD	The dictionary is restricted updating	WARNING
NODIC	No dictionary available in this editing session	WARNING
NOENDOFLINE	Returning a range of text with no end of line	SUCCESS
NOEOBSTR	Cannot return a string at end of buffer	WARNING
NOFILEACCESS	Unable to access file 'name'	ERROR
NOFILEROUTINE	No routine specified to perform FILE I/O	FATAL
NOFOCUSOWNER	There is no input focus owner	WARNING
NOGBLSELDATA	No global selection data	WARNING
NOGBLSELOWNER	There is no global selection owner	WARNING
NOJOURNAL	Editing session is not being journaled	WARNING
NOKEYMAPLIST	Attempt to access an undefined key-map list 'key-map-list-name'	WARNING

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
NOKEYMAP	Attempt to access an undefined key map 'key-map-name'	WARNING
NONAMES	There are no names matching the one requested	WARNING
NONANSICRT	SYS$INPUT must be supported CRT	ERROR
NOPARENT	There is no parent process to attach to	WARNING
NOPROCESS	No subprocess to interact with	WARNING
NOREDEFINE	Built-in procedure 'name' cannot be redefined	ERROR
NORETURNVALUE	Built-in does not return a value	ERROR
NOSELECT	No select active	WARNING
NOSHOWBUF	Variable SHOW_BUFFER does not exist or is not a buffer	WARNING
NOTARRAY	Indexed variable is not an array	WARNING
NOTDEFINABLE	That key is not definable	WARNING
NOTERRORKEYWORD	Error handler selector is not an error keyword	ERROR
NOTIMPLEMENTED	Builtin compiled by 'name' is not implemented by 'name'	ERROR
NOTJOURNAL	'file' is not a journal file	ERROR
NOTLEARNING	You have not begun a learn sequence	WARNING
NOTMODIFIABLE	Attempt to change unmodifiable buffer 'string'	WARNING
NOTSAMEBUF	The markers are not in the same buffer	WARNING
NOTSUBCLASS	Object is not a subclass of WindowObjClass	WARNING
NOTYET	Not yet implemented	WARNING
NOWINDOW	Attempt to position the cursor outside all of the mapped windows	WARNING
NO	NO	INFORMATIONAL
NULLSTRING	Null string used	WARNING
OCCLUDED	Builtin/keyword 'name' occluded by declaration	INFORMATIONAL
ONDELRECLIST	Attempt to access a record on the deleted list	FATAL
ONELEARN	Cannot start a learn sequence while one is active	WARNING
ONESELECT	Select already active, maximum 1 per buffer	WARNING
OPENDIC	Error opening the dictionary file	ERROR
OPENIN	Error opening 'input-file' as input	ERROR
OPENOUT	Error opening 'output-file' as output	ERROR
OVERLAPRANGE	Overlapping ranges, operation terminated	WARNING
PARSEFAIL	Error parsing 'file-spec'	WARNING
PARSEOVER	Parser stack overflow	ERROR
PREMATUREEOF	Premature end-of-file detected	ERROR
PRESSRET	Press RETURN to continue...	SUCCESS
PROCESSBEG	Subprocess activated	SUCCESS

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
PROCESSEND	Subprocess terminated	SUCCESS
PROCSUPERSEDED	Definition of procedure 'name' superseded	INFORMATIONAL
QUITTING	Editor quitting	SUCCESS
READABORTED	READ_CHAR, READ_KEY, or READ_LINE builtin was aborted	WARNING
READERR	Error reading 'input-file-spec'	ERROR
READQUEHEADER	Attempt to read queue header of dscb: 'address'	FATAL
READZERO	Read of record id 0, dscb: 'address'	FATAL
REALLYQUIT	Buffer modifications will not be saved, continue quitting (Y or N)?	SUCCESS
REALLYRECOVER	Continue recovering (Y or N)?	SUCCESS
RECJNLOPEN	Journal file open, recovery status unchanged	ERROR
RECOVERABORT	Recovery aborted by journal file inconsistency, journal file closed	WARNING
RECOVERBEG	Recovery started	SUCCESS
RECOVERBUFBEG	Recovery started for buffer 'buffer name'	SUCCESS
RECOVERBUFEND	Recovery complete for buffer 'buffer name'	SUCCESS
RECOVERBUFFILE1	Can not recover from file 'file name'	SUCCESS
RECOVERBUFFILE2	Please type in a new file specification:	SUCCESS
RECOVEREND	Recovery complete	SUCCESS
RECOVERFAIL	Recovery terminated abnormally, journal file inconsistency	ERROR
RECOVERQUIT	No file name specified, nothing recovered	WARNING
RECURLEARN	Learn sequence replay halted due to recursion	WARNING
RECURREAD	READ_LINE cannot be part of user defined READ_LINE procedure	WARNING
REFRESH_NEEDED	Screen refresh needed	WARNING
REGWIDDUP	Registration string already associated with a different widget	WARNING
REPLAYFAIL	An inconsistency has been discovered, halting execution	WARNING
REPLAYWARNING	An inconsistency has been discovered, continuing execution	WARNING
REQARGSMISSING	One or more required arguments missing	ERROR
REQUIRESDECW	Feature requires the VAXTPU DECwindows screen updater	ERROR
REQUIRESTERM	Feature requires a terminal	ERROR
RESTOREFAIL	Error during RESTORE operation	ERROR
REVERSECASE	CASE limits were reversed	INFORMATIONAL
ROUND	FORWARD was rounded to the top	INFORMATIONAL
SAVEAMBIGSYM	Saving ambiguous symbol 'name'	INFORMATIONAL

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
SAVEERROR	Error during SAVE operation	ERROR
SAVEUNDEFPROC	Saving undefined procedure 'name'	INFORMATIONAL
SCANADVANCE	*** Scanner advanced to "'name'" ***	ERROR
SEARCHFAIL	Error searching for 'file-spec'	WARNING
SECTRESTORED	'count' procedure(s), 'count' variable(s), 'count' key(s) restored	INFORMATIONAL
SECTSAVED	'count' procedure(s), 'count' variable(s), 'count' key(s) saved	SUCCESS
SECTUNDEFPROC	Saved 'count' undefined procedure(s), 'count' ambiguous symbol(s)	WARNING
SELRANGEZERO	Select range has 0 length	WARNING
SENDFAIL	Unable to send to subprocess	WARNING
SHRCLA	The current clause has been shrunk	INFORMATIONAL
SOURCELINE	At source line 'integer'	INFORMATIONAL
STACKOVER	Stack overflow during compilation	ERROR
STATOOLONG	Truncating status line to 'count' characters	INFORMATIONAL
STRNOTFOUND	String not found	WARNING
STRTOOLARGE	String greater than 65535 characters	ERROR
STRTOOLNG	String is too long for a conversion source	ERROR
SUCCESS	Successful completion	SUCCESS
SYMDELETE	*** Error symbol deleted ***	ERROR
SYMINSERT	*** "'name'" inserted before error symbol ***	ERROR
SYMREPLACE	*** Error symbol replaced by "'name'" ***	ERROR
SYMTBLFUL	All symbol tables are full	ERROR
SYNTAXERROR	Syntax error	ERROR
SYSERROR	System service error	ERROR
TEXT	'message'	INFORMATIONAL
TIMEOUT	Builtin timed out	WARNING
TOOFEW	Too few arguments	ERROR
TOOMANYCLA	Too many clauses are found while converting	WARNING
TOOMANYPARAM	Too many formal parameters/local variables	ERROR
TOOMANYRECS	Too many records	ERROR
TOOMANY	Too many arguments	ERROR
TOPLINETRUNC	Calculated new first line 'integer', changed to 1	INFORMATIONAL
TRUNCATE	Line truncated to 'count' characters	WARNING
UKNFACILITY	Unknown facility code specified	WARNING
UNARYOPER	Operand combination 'oper' 'type' unsupported	WARNING

(continued on next page)

Table D–1 (Cont.) VAXTPU Messages and Their Severity Levels

Abbreviation	Message	Severity Level
UNDEFINEDPROC	Undefined procedure call 'name'	ERROR
UNDWIDCLA	Undefined widget class specified	WARNING
UNKESCAPE	Unknown escape sequence read	WARNING
UNKKEYWORD	An unknown keyword has been used as an argument	ERROR
UNKLEXICAL	Unknown lexical element	ERROR
UNKOPCODE	Unknown opcode 'value'	ERROR
UNKTYPE	Unknown data type 'value'	ERROR
UNKWNDESC	Unknown descriptor type	ERROR
UNREACHABLE	Unreachable code	INFORMATIONAL
WIDMISMATCH	Parameter 'number"s class, 'class', unsupported	ERROR
WINDNOTMAPPED	The window is not mapped to a buffer	WARNING
WINDNOTVIS	Built-in cannot operate on an invisible window	WARNING
WRITEERR	Error writing 'output-file-spec'	ERROR
YES	YES	INFORMATIONAL

E DEC Multinational Character Set

This appendix presents the DEC multinational character set. In Table E–1 the control characters are shown as reverse question marks, which is how they appear on the VT300 series and VT200 series of terminals. On the VT100 series of terminals, control characters appear as rectangles.

Table E–1 DEC Multinational Character Set

Graphic	Decimal Value	Abbreviation	Description
?	0	NUL	null character
?	1	SOH	start of heading
?	2	STX	start of text
?	3	ETX	end of text
?	4	EOT	end of transmission
?	5	ENQ	enquiry
?	6	ACK	acknowledge
?	7	BEL	bell
?	8	BS	backspace
HT	9	HT	horizontal tabulation
LF	10	LF	line feed
VT	11	VT	vertical tabulation
FF	12	FF	form feed
CR	13	CR	carriage return
?	14	SO	shift out
?	15	SI	shift in
?	16	DLE	data link escape
?	17	DC1	device control 1
?	18	DC2	device control 2
?	19	DC3	device control 3
?	20	DC4	device control 4
?	21	NAK	negative acknowledge
?	22	SYN	synchronous idle
?	23	ETB	end of transmission block
?	24	CAN	cancel
?	25	EM	end of medium
?	26	SUB	substitute

(continued on next page)

Table E–1 (Cont.) DEC Multinational Character Set

Graphic	Decimal Value	Abbreviation	Description
?	27	ESC	escape
?	28	FS	file separator
?	29	GS	group separator
?	30	RS	record separator
?	31	US	unit separator
?	32	SP	space
!	33	!	exclamation point
"	34	"	quotation marks (double quote)
#	35	#	number sign
$	36	$	dollar sign
%	37	%	percent sign
&	38	&	ampersand
'	39	'	apostrophe (single quote)
(	40	(	opening parenthesis
)	41	)	closing parenthesis
*	42	*	asterisk
+	43	+	plus
,	44	,	comma
-	45	-	hyphen or minus
.	46	.	period or decimal point
/	47	/	slash
0	48	0	zero
1	49	1	one
2	50	2	two
3	51	3	three
4	52	4	four
5	53	5	five
6	54	6	six
7	55	7	seven
8	56	8	eight
9	57	9	nine
:	58	:	colon
;	59	;	semicolon
<	60	<	less than
=	61	=	equals

(continued on next page)

Table E–1 (Cont.) DEC Multinational Character Set

Graphic	Decimal Value	Abbreviation	Description
>	62	>	greater than
?	63	?	question mark
@	64	@	commercial at
A	65	A	uppercase A
B	66	B	uppercase B
C	67	C	uppercase C
D	68	D	uppercase D
E	69	E	uppercase E
F	70	F	uppercase F
G	71	G	uppercase G
H	72	H	uppercase H
I	73	I	uppercase I
J	74	J	uppercase J
K	75	K	uppercase K
L	76	L	uppercase L
M	77	M	uppercase M
N	78	N	uppercase N
O	79	O	uppercase O
P	80	P	uppercase P
Q	81	Q	uppercase Q
R	82	R	uppercase R
S	83	S	uppercase S
T	84	T	uppercase T
U	85	U	uppercase U
V	86	V	uppercase V
W	87	W	uppercase W
X	88	X	uppercase X
Y	89	Y	uppercase Y
Z	90	Z	uppercase Z
[	91	[	opening bracket
\	92	\	back slash
]	93	]	closing bracket
^	94	^	circumflex
_	95	_	underline (underscore)
`	96	`	grave accent

(continued on next page)

Table E–1 (Cont.) DEC Multinational Character Set

Graphic	Decimal Value	Abbreviation	Description
a	97	a	lowercase a
b	98	b	lowercase b
c	99	c	lowercase c
d	100	d	lowercase d
e	101	e	lowercase e
f	102	f	lowercase f
g	103	g	lowercase g
h	104	h	lowercase h
i	105	i	lowercase i
j	106	j	lowercase j
k	107	k	lowercase k
l	108	l	lowercase l
m	109	m	lowercase m
n	110	n	lowercase n
o	111	o	lowercase o
p	112	p	lowercase p
q	113	q	lowercase q
r	114	r	lowercase r
s	115	s	lowercase s
t	116	t	lowercase t
u	117	u	lowercase u
v	118	v	lowercase v
w	119	w	lowercase w
x	120	x	lowercase x
y	121	y	lowercase y
z	122	z	lowercase z
{	123	{	opening brace
|	124	|	vertical line
}	125	}	closing brace
~	126	~	tilde
DEL	127	DEL	delete, rubout
?	128	—	[reserved]
?	129	—	[reserved]
?	130	—	[reserved]
?	131	—	[reserved]

(continued on next page)

Table E–1 (Cont.) DEC Multinational Character Set

Graphic	Decimal Value	Abbreviation	Description
?	132	IND	index
?	133	NEL	next line
?	134	SSA	start of selected area
?	135	ESA	end of selected area
?	136	HTS	horizontal tab set
?	137	HTJ	horizontal tab set with justification
?	138	VTS	vertical tab set
?	139	PLD	partial line down
?	140	PLU	partial line up
?	141	RI	reverse index
?	142	SS2	single shift 2
?	143	SS3	single shift 3
?	144	DCS	device control string
?	145	PU1	private use 1
?	146	PU2	private use 2
?	147	STS	set transmit state
?	148	CCH	cancel character
?	149	MW	message waiting
?	150	SPA	start of protected area
?	151	EPA	end of protected area
?	152	—	[reserved]
?	153	—	[reserved]
?	154	—	[reserved]
?	155	CSI	control sequence introducer
?	156	ST	string terminator
?	157	OSC	operating system command
?	158	PM	privacy message
?	159	APC	application program command
?	160	—	[reserved]
¡	161	¡	inverted exclamation mark
¢	162	¢	cent sign
£	163	£	pound sign
?	164	—	[reserved]
¥	165	¥	yen sign
?	166	—	[reserved]

(continued on next page)

Table E–1 (Cont.) DEC Multinational Character Set

Graphic	Decimal Value	Abbreviation	Description
§	167	§	section sign
¤	168	¤	general currency sign
©	169	©	copyright sign
ª	170	ª	feminine ordinal indicator
«	171	«	angle quotation mark left
?	172	—	[reserved]
?	173	—	[reserved]
?	174	—	[reserved]
?	175	—	[reserved]
°	176	°	degree sign
±	177	±	plus/minus sign
²	178	²	superscript 2
³	179	³	superscript 3
?	180	—	[reserved]
µ	181	µ	micro sign
¶	182	¶	paragraph sign, pilcrow
·	183	·	middle dot
?	184	—	[reserved]
¹	185	¹	superscript 1
º	186	º	masculine ordinal indicator
»	187	»	angle quotation mark right
¼	188	¼	fraction one quarter
½	189	½	fraction one half
?	190	—	[reserved]
¿	191	¿	inverted question mark
À	192	À	uppercase A with grave accent
Á	193	Á	uppercase A with acute accent
Â	194	Â	uppercase A with circumflex
Ã	195	Ã	uppercase A with tilde
Ä	196	Ä	uppercase A with umlaut, (diaeresis)
Å	197	Å	uppercase A with ring
Æ	198	Æ	uppercase AE diphthong
Ç	199	Ç	uppercase C with cedilla
È	200	È	uppercase E with grave accent
É	201	É	uppercase E with acute accent

(continued on next page)

Table E–1 (Cont.) DEC Multinational Character Set

Graphic	Decimal Value	Abbreviation	Description
Ê	202	Ê	uppercase E with circumflex
Ë	203	Ë	uppercase E with umlaut, (diaeresis)
Ì	204	Ì	uppercase I with grave accent
Í	205	Í	uppercase I with acute accent
Î	206	Î	uppercase I with circumflex
Ï	207	Ï	uppercase I with umlaut, (diaeresis)
?	208	—	[reserved]
Ñ	209	Ñ	uppercase N with tilde
Ò	210	Ò	uppercase O with grave accent
Ó	211	Ó	uppercase O with acute accent
Ô	212	Ô	uppercase O with circumflex
Õ	213	Õ	uppercase O with tilde
Ö	214	Ö	uppercase O with umlaut, (diaeresis)
Œ	215	Œ	uppercase OE ligature
Ø	216	Ø	uppercase O with slash
Ù	217	Ù	uppercase U with grave accent
Ú	218	Ú	uppercase U with acute accent
Û	219	Û	uppercase U with circumflex
Ü	220	Ü	uppercase U with umlaut, (diaeresis)
Ÿ	221	Ÿ	uppercase Y with umlaut, (diaeresis)
?	222	—	[reserved]
ß	223	ß	German lowercase sharp s
à	224	à	lowercase a with grave accent
á	225	á	lowercase a with acute accent
â	226	â	lowercase a with circumflex
ã	227	ã	lowercase a with tilde
ä	228	ä	lowercase a with umlaut, (diaeresis)
å	229	å	lowercase a with ring
æ	230	æ	lowercase ae diphthong
ç	231	ç	lowercase c with cedilla
è	232	è	lowercase e with grave accent
é	233	é	lowercase e with acute accent
ê	234	ê	lowercase e with circumflex
ë	235	ë	lowercase e with umlaut, (diaeresis)
ì	236	ì	lowercase i with grave accent

(continued on next page)

Table E–1 (Cont.) DEC Multinational Character Set

Graphic	Decimal Value	Abbreviation	Description
í	237	í	lowercase i with acute accent
î	238	î	lowercase i with circumflex
ï	239	ï	lowercase i with umlaut, (diaeresis)
?	240	—	[reserved]
ñ	241	ñ	lowercase n with tilde
ò	242	ò	lowercase o with grave accent
ó	243	ó	lowercase o with acute accent
ô	244	ô	lowercase o with circumflex
õ	245	õ	lowercase o with tilde
ö	246	ö	lowercase o with umlaut, (diaeresis)
œ	247	œ	lowercase oe ligature
ø	248	ø	lowercase o with slash
ù	249	ù	lowercase u with grave accent
ú	250	ú	lowercase u with acute accent
û	251	û	lowercase u with circumflex
ü	252	ü	lowercase u with umlaut, (diaeresis)
ÿ	253	ÿ	lowercase y with umlaut, (diaeresis)
?	254	—	[reserved]
?	255	—	[reserved]

F VAXTPU File Support

When you edit with VAXTPU, some file attributes may be changed. VAXTPU supports some file attributes in that it preserves the particular file attribute. VAXTPU does not support other file attributes; it converts the file attributes to VAXTPU's default attribute. For more information on file attributes, see the *VMS Record Management Services Manual*. Table F–1 shows the file attributes that VAXTPU supports. It also lists the default file attributes for VAXTPU.

Table F–1 VAXTPU Support of File Attributes

File Organization	Status as Supported or Unsupported
Index	Unsupported
Relative	Unsupported
Sequential	Supported (default)
Record Format	
Fixed length	Unsupported
Stream	Supported
Stream-CR	Supported
Stream-LF	Supported
Undefined	Supported
Variable length	Supported (default)
VFC	Unsupported
Record Attribute	
Block	Supported
Carriage return	Supported (default)
FORTRAN	Unsupported
None	Supported
Print	Unsupported

G EVE$BUILD Module

VAXTPU includes a module, EVE$BUILD, for building applications on EVE.

EVE$BUILD is a tool for modifying EVE or layering other products on EVE. EVE$BUILD compiles VAXTPU code with an existing EVE section file to produce a new section file. This new file can define either a new version of EVE or a new product. Both customers and Digital developers can use EVE$BUILD.

In using these instructions, type uppercase strings exactly as they appear here. Replace lowercase strings with appropriate values. For example, in the expression *product_MASTER.FILE,* the string "product" indicates that you should substitute the product name of your choice. The string "MASTER.FILE" must be appended to the product name exactly as it appears in these comments.

These instructions cover the following:

- How to prepare code for use with EVE$BUILD
- How to invoke EVE$BUILD
- What happens when you use EVE$BUILD

G.1 How to Prepare Code for Use with EVE$BUILD

For purposes of this section, it is assumed you have VAXTPU code that modifies EVE or layers another product on EVE. To turn this code into a section file using EVE$BUILD, follow the guidelines in this section.

There are seven areas in which you must observe special coding conventions:

- Module identifiers
- Parsers
- Initialization
- Command synonyms
- Status line fields
- Exit handlers
- Quit handlers

G.1.1 Module Identifiers

Organize the VAXTPU code into one or more modules. (This section defines "module" in more detail below.) Once you set up one or more modules, EVE$BUILD provides an audit trail showing what version of each module was used to build each new section file. Digital recommends that you put into one or more modules all the code to be used with EVE$BUILD.

To define a module, create a file containing one or more VAXTPU procedures and (if appropriate) one or more executable statements. All procedures and statements in a module should be related to the same task or subject. Then insert a new procedure at the beginning of the module. This procedure will return an "ident," or module identifier, which EVE$BUILD tracks during the build process. Use the following format for this procedure:

```
PROCEDURE facility_MODULE_IDENT
   RETURN "version-number";
ENDPROCEDURE;
```

In place of "facility," use a unique module identifier of up to 15 characters. If the VAXTPU code in the module is part of a Digital product, begin the identifier with the registered product facility code such as EVE or NOTES, followed by a dollar sign and the specific module name. For example, the facility used in the major EVE module is EVE$CORE. As a result, the module containing EVE$CORE has the identifier EVE$CORE_MODULE_IDENT.

If the code is not part of a Digital product, do not use a dollar sign in the module identifier.

In place of "version-number," use any string of up to 15 characters identifying the version number of the module.

EVE$BUILD keeps a list showing the ident of each module it uses in a build. The list is kept in a file referred to as the .LIST file. This file is discussed in Section G.2. In EVE, the format used for the version number string is *Vnn-mmm*. The characters "nn" represent the major version number of EVE to which the module belongs. The characters "mmm" represent the edit number.

The following code is the _MODULE_IDENT procedure used by the module EVE$CORE.TPU:

```
PROCEDURE eve$core_module_ident
   RETURN "V02-242";
ENDPROCEDURE;
```

G.1.2 Parsers

EVE$BUILD can accommodate one or more user-written parsing routines in addition to the parser included in EVE. If you choose to include a parser in your product, the parser can either supplement or replace EVE's parser.

If you include one or more parsers in your product, the module containing the parser should define a variable of the following form:

```
EVE$X_ENABLE_PARSER_facility
```

Replace the term "facility" with the name of the module in which the parsing routine appears. For example, if the parser occurs in the module SCHEDULER, the variable is as follows:

```
EVE$X_ENABLE_PARSER_SCHEDULER
```

Next, name the procedure implementing the parser. If the product is not a Digital product, use the following format:

```
facility_PROCESS_COMMAND
```

Replace the term "facility" with the name of the module in which the parsing routine appears. For example, if the parser occurs in the module with the ident SCHEDULER_MODULE_IDENT, the procedure has the following name:

```
SCHEDULER_PROCESS_COMMAND
```

If the product is a Digital product, name the procedure using the following format:

```
facility$PROCESS_COMMAND
```

EVE has a procedure named EVE$PARSER_DISPATCH that defaults to the following code:

```
PROCEDURE EVE$PARSER_DISPATCH (the_command)
   EVE$PROCESS_COMMAND (the_command);
ENDPROCEDURE;
```

If you do not define a parser-related variable, then the default EVE$PARSER_DISPATCH is put into the .INIT file.

If you do define one or more parser-related variables, EVE$BUILD verifies that a corresponding *facility*_PROCESS_COMMAND procedure exists for each variable. If not, the build fails. If the corresponding procedure does exist, EVE$BUILD then adds the following code to EVE$PARSER_DISPATCH just before the call to EVE$PROCESS_COMMAND:

```
IF EVE$X_ENABLE_PARSER_facility
   THEN
       IF (facility_PROCESS_COMMAND (THE_COMMAND))
       THEN
           RETURN;
       ENDIF;
   ENDIF;
```

If you want a particular module's parser to supersede EVE's parser, your parser should return a true status whether or not it can parse a command. If you want your parser to supplement EVE's parser, your parser should return a false status if it cannot parse a command. The false status allows the parsers in other modules, and finally EVE's parser, to try to parse the command. The parsers are called in the order in which they appear in the master file. (The master file is discussed in Section G.2.)

G.1.3 Initialization

EVE$BUILD allows module-specific initialization. To perform initialization in a module, put an initializing procedure in the module and name the procedure using the following format:

```
facility_MODULE_INIT
```

Replace the term "facility" with the name of the module in which the procedure appears. For example, if it occurs in the module SCHEDULER_MODULE_IDENT, the procedure is named as follows:

```
SCHEDULER_MODULE_INIT
```

The EVE module EVE$CORE.TPU contains a null procedure called EVE$INIT_MODULES. EVE$BUILD replaces EVE$INIT_MODULES with a procedure that calls each procedure whose name ends with _MODULE_INIT. The initialization procedures are called in the order in which they are found in the master file. (The master file is discussed in Section G.2.)

EVE performs initialization in the following order:

1 Processing of the procedure TPU$INIT_PROCEDURE

  VAXTPU executes the procedure TPU$INIT_PROCEDURE immediately after processing the /DEBUG qualifier. TPU$INIT_PROCEDURE performs the following tasks:

  - Initialization of EVE's variables and settings
  - Package preinitialization
  - Initialization of EVE's buffers, windows, and files
  - Initialization of user-written modules
  - Call to the end user's initialization file, TPU$LOCAL_INIT

2 Processing of the /COMMAND qualifier if it is present on the DCL command line

3 Processing of the procedure TPU$INIT_POSTPROCEDURE

  VAXTPU executes the procedure TPU$INIT_POSTPROCEDURE after processing the /COMMAND qualifier. TPU$INIT_POSTPROCEDURE performs the following tasks:

  - Execution of EVE commands in the initialization file

- Creation and initialization of the $DEFAULTS$ buffer. This buffer is a template for all buffers created during an editing session. New buffers obtain settings from the $DEFAULTS$ buffer for attributes such as margin settings, direction, mode, and so on.

During the "preinitialization" phase, you can redefine EVE's variables and settings to be compatible with your product.

Do *not* redefine any EVE variable or setting unless you are sure you understand all the possible side effects on EVE and on your product. Use of this option is recommended only for experienced EVE programmers.

To use preinitialization, put an initializing procedure in a module and name the procedure using the following format:

```
facility_MODULE_PRE_INIT
```

Replace the term "facility" with the name of the module in which the initializing procedure appears. For example, if it occurs in the module SCHEDULER_MODULE_IDENT, the procedure is named as follows:

```
SCHEDULER_MODULE_PRE_INIT
```

The EVE module EVE$CORE.TPU contains a null procedure called EVE$PRE_INIT_MODULES. EVE$BUILD replaces EVE$PRE_INIT_MODULES with a procedure that calls each procedure whose name ends with _MODULE_PRE_INIT. The initialization procedures are called in the order in which they are found in the master file. (The master file is discussed in Section G.2.)

Most programmers who are layering a product onto EVE should initialize modules by using procedures of the type *facility*_MODULE_INIT. Use of TPU$LOCAL_INIT should be reserved for the end user. Use of procedures of the type *facility*_MODULE_PRE_INIT should be reserved for very experienced EVE programmers.

G.1.4 Command Synonyms

A command synonym is a string that produces exactly the same effect as an EVE command or phrase. Command synonyms are useful for creating foreign-language versions of EVE or a product layered onto EVE. For example, you could designate the Swedish string "näasta_bild" to have the same effect as the EVE command NEXT SCREEN.

EVE$BUILD allows you to create synonyms both for EVE commands and for user-written commands. This discussion assumes that when you create a command synonym, you first choose a **root command** (the EVE command or user-written command for which you want to create a synonym), and then equate to the root command a **synonym** (the string that is to produce the same effect as the root command does).

You can create synonyms in each module of your product. To create synonyms in a module, you perform two steps:

1 Create an initializing procedure

2 Place synonym declaration statements in the initializing procedure

Name the initializing procedure using the following format:

```
facility_DECLARE_SYNONYM
```

Replace the term "facility" with the name of the module in which the procedure appears. For example, if you create the procedure in the module SCHEDULER, you would name the procedure as follows:

```
SCHEDULER_DECLARE_SYNONYM
```

To declare a synonym, use the EVE$BUILD_SYNONYM statement in the DECLARE_SYNONYM procedure. This command enters the root command and the synonym into EVE'S data structure associating synonyms with root commands. Use one EVE$BUILD_SYNONYM statement for each synonym you want to declare. The statement has the following format:

```
EVE$BUILD_SYNONYM ("root_command", "synonym", integer)
```

The parameters are as follows:

root-command — A quoted string naming the command for which you want to declare a synonym. The string must not contain spaces. If the command contains more than one word, place an underscore between the words.

synonym — A quoted string naming the synonym you want to associate with the root command. The string must not contain spaces. If the command usually contains more than one word, place an underscore between the words.

integer — Either 0, 1, or 2.

The value 0 tells EVE$BUILD that the programmer, not EVE$BUILD, will create the procedure and parameters implementing the synonym. This value instructs EVE$BUILD simply to verify that the root command exists and to associate the root command with the synonym.

The value 1 causes EVE$BUILD to perform the following tasks:

- Verify that the root command exists.
- Associate the root command with the synonym.
- Create a new procedure giving the synonym the same effect as the root command.
- Declare how many parameters are expected by the procedure implementing the synonym. (That is, if the procedure implementing the root command requires two parameters, then the procedure implementing the synonym also requires two parameters.)
- Initialize the parameters of the synonym procedure so they equal the parameters of the corresponding root procedure.

The value 2 causes EVE$BUILD simply to associate the root with the synonym. Use this value if you are creating a synonym for a phrase rather than a command synonym.

Example

The following statement creates a Spanish synonym for the ONE WINDOW command and instructs EVE$BUILD to create the necessary structures for the synonym:

```
EVE$BUILD_SYNONYM ("one_window", "una_ventana", 1)
```

You can declare a synonym to be a terminator. A terminator is a command that, if bound to a key and executed with a keystroke, tells an EVE prompt to stop prompting. For example, when the DO command is bound to the DO key, pressing the DO key terminates the prompts resulting from several commands, including DEFINE KEY and FIND.

To make a synonym a terminator, use a EVE$MAKE_SYNONYM_A_TERMINATOR statement in the *facility*_MODULE_INIT procedure. For example, if you wanted to make the string "Haga" a synonym for "DO" and to declare "Haga" as a terminator, you would place the following statement in the *facility*_MODULE_INIT procedure for the module:

```
EVE$MAKE_SYNONYM_A_TERMINATOR ("DO", "Haga");
```

G.1.5 Status Line Fields

Using EVE$BUILD, you can create new areas for displaying information in the status line that EVE displays under each window. These areas are called "fields." By default, the EVE status line contains fields to display the following information:

- The buffer mapped to the window
- The text entry mode
- The direction of the buffer

A field can display more than one message. For example, the direction field in the default EVE status line can display either the string "Forward" or the string "Reverse."

To add a field to the status line, write a procedure creating the field and include the procedure in the appropriate module. The following sample procedure creates a field indicating whether a buffer is a read-only buffer:

```
! Procedure to put up the "Read-Only" indicator on NO_WRITE buffers

PROCEDURE eve$nowrite_status_field (the_length, ! Status line indicator
                                    the_format)
   ON_ERROR
       [OTHERWISE]:
   ENDON_ERROR;

   IF GET_INFO (CURRENT_BUFFER, "no_write")
      THEN
          RETURN FAO (the_format, eve$x_read_only);
      ELSE
          RETURN "";
   ENDIF;

ENDPROCEDURE;
```

You will find it helpful to observe the following conventions:

- Use the following format for the procedure name:

  ```
  field_name_STATUS_FIELD
  ```

  For example, if you are adding a field to display the current line number and if your facility is called SCHEDULER, the first line of the procedure appears as follows:

  ```
  PROCEDURE SCHEDULER_LINE_NUMBER_STATUS_FIELD
  ```

- Give the procedure the following input parameters:
  - **max_size** — The number of unused column spaces in the status line before the new field is added to the line. Use this parameter to ensure that all messages fit on the status line.
  - **the_format** — The FAO directive to be used to format the field.

The module EVE$CORE.TPU contains a procedure called EVE$GET_STATUS_FIELDS that simply returns the null string. EVE$BUILD replaces EVE$GET_STATUS_FIELDS with the following procedure:

```
PROCEDURE EVE$GET_STATUS_FIELDS (the_length, the_format)

   LOCAL  remaining,
          the_fields,
          the_field;

   the_fields := "";
   remaining := the_length;

   RETURN the_fields

ENDPROCEDURE;
```

For each _STATUS_FIELD procedure you put in a module, EVE$BUILD inserts the following code just before the "RETURN the_fields" statement:

```
the_field := field_name_STATUS_FIELD (remaining, the_format);
IF LENGTH (the_field) <= remaining
THEN
    the_fields := the_field + the_fields;
    remaining := remaining - LENGTH (the_field);
ENDIF;
```

G.1.6 Exit and Quit Handlers

When you create a new or layered product, you can provide one or more user-written exit handlers, one or more user-written quit handlers, or one or more of both. Depending on how you write the handlers, EVE$BUILD uses your exit or quit handlers either in addition to or instead of those provided by EVE. This section contains pointers on writing both supplementary and replacement handlers.

When you write an exit handling procedure, name the procedure using the following format for a non-Digital product:

```
facility_EXIT_HANDLER
```

Use the following format for a non-Digital quit handler:

```
facility_QUIT_HANDLER
```

Replace the term "facility" with the name of the module in which the handler appears. For example, if the handler occurs in the module with the ident SCHEDULER_MODULE_IDENT, you name an exit handling procedure as follows:

```
SCHEDULER_EXIT_HANDLER
```

You would name a quit handling procedure as follows:

```
SCHEDULER_QUIT_HANDLER
```

If the product is a Digital product, name the procedure using the following format for an exit handler:

```
facility$EXIT_HANDLER
```

Use the following format for a quit handler:

```
facility$QUIT_HANDLER
```

EVE has procedures named EVE$EXIT_DISPATCH and EVE$QUIT_DISPATCH. By default, EVE$EXIT_DISPATCH contains the following code:

```
PROCEDURE EVE$EXIT_DISPATCH (the_command)
   EVE$EXIT;
ENDPROCEDURE;
```

By default, EVE$QUIT_DISPATCH contains the following code:

```
PROCEDURE EVE$QUIT_DISPATCH (the_command)
   EVE$QUIT;
ENDPROCEDURE;
```

If you do not create an exit or quit handling procedure, EVE$BUILD puts the default versions of EVE$EXIT_DISPATCH and EVE$QUIT_DISPATCH into the .INIT file. If you create an exit handling procedure, EVE$BUILD adds the following code to EVE$EXIT_DISPATCH just before the call to EVE$EXIT:

```
IF facility_EXIT_HANDLER
THEN
    RETURN;
ENDIF;
```

If you create a quit handling procedure, EVE$BUILD adds the following code to EVE$QUIT_DISPATCH just before the call to EVE$QUIT:

```
IF facility_QUIT_HANDLER
THEN
    RETURN;
ENDIF;
```

If you want a particular module's exit or quit handler to supersede EVE's handler, your handler should return a true status. If you want your handler to supplement EVE's handler, your handler should return a false

status. The false status allows EVE$BUILD to call the handlers in other modules and in EVE.

G.1.7 How to Invoke EVE$BUILD

To prepare to use EVE$BUILD, define the following foreign command:

```
$ BUILD == "EDIT/TPU/NODISPLAY/SECTION=EVE$SECTION-
_$ /COMMAND=device:[dir]EVE$BUILD/NOINITIALIZATION
```

If you specify /SECTION=EVE$SECTION, EVE$BUILD builds your product on top of the standard EVE section file. To build your product with a different version of EVE, specify a different section file with the /SECTION qualifier.

In most circumstances, you specify either the standard EVE section file or your own enhanced EVE section file. No matter which section file you specify, you must use the /NODISPLAY qualifier if you use the /SECTION qualifier.

If for some reason you want to rebuild EVE from scratch, you build it /NOSECTION and use the EVE$MASTER.FILE that comes with the EVE sources.

After defining the foreign command, create a master file. This file tells EVE$BUILD what modules to compile. If your product is not a Digital product, name your master file using the following format:

```
facility_MASTER.FILE
```

For example, a valid name for a non-Digital product's master file might be as follows:

```
SCHEDULER_MASTER.FILE
```

If your product is a Digital product, name your master file using the following format:

```
facility$MASTER.FILE
```

Replace "facility" with the name of your product. For example, a valid name for a Digital product's master file might be as follows:

```
NOTES$MASTER.FILE
```

When you have created and named the master file, type into it the name of each file whose contents you want to compile. Usually this means you type in the name of each file containing a module that is part of your product. If the files containing the modules are not in the same directory as the master file, then you must specify the directory name of each module file.

If one or more of your modules declare synonyms, enter the names of those modules at the end of the file. This ensures that all root commands have been created before synonyms for root commands are declared.

EVE$BUILD processes the modules in the order in which they appear in the master file. For example, EVE$BUILD calls exit and quit handlers in the same order that they occur in the master file.

Once you have completed the master file, create a version file in the same directory that contains the master file. If your product is not a Digital product, name the version file as follows:

```
facility_VERSION.DAT
```

If your product is a Digital product, name the version file using the following format:

```
facility$VERSION.DAT
```

The version file is a text file containing only the version number for the product. This version number is built into the section file as part of the value of the procedure EVE$VERSION.

When you have a foreign command, a master file, and a version file, you can invoke EVE$BUILD with the following command:

```
$  BUILD facility
```

For example, if the name of your product was SCHEDULER, you would build it by typing the following:

```
$  BUILD SCHEDULER
```

You can use the /OUTPUT qualifier to specify the name of the section file to create. If you do not use the qualifier, EVE$BUILD prompts for a file name. If you respond with a null file name, EVE$BUILD gives the output file the same name as the product.

EVE$BUILD does not produce a log file if /NODISPLAY is used on the DCL command line. In addition, EVE$BUILD does not produce a log file if /DISPLAY is used on the DCL command line and the build produces errors.

G.2 What Happens When You Use EVE$BUILD

Each file specified in the master file is read in and compiled. If there are any executable statements after the procedure definitions, the statements are compiled and executed. Any SAVE or QUIT statements or calls to DEBUGON (this procedure is defined in TPU$DEBUG.TPU) are removed before execution.

EVE$BUILD creates the following three output files:

- The new section file, with a file type of .TPU$SECTION
- A file preserving the dynamically generated code, with a file type of .INIT
- A file tracking what happened during the build, with a file type of .LIST

All three files have the same device, directory, and file name.

The .INIT file contains the following:

- EVE$DYNAMIC_MODULE_IDENT
- EVE$PARSER_DISPATCH

- EVE$MODULE_PRE_INIT
- EVE$MODULE_INIT
- EVE$GET_HELP_LIBRARY_TOPIC
- EVE$VERSION

The .LIST file contains the following:

- The date and time of the build
- The version of EVE used
- The full file specifications of the master file, section file, version file, and .INIT file
- A synopsis on each source module, including the module ident, the number of lines in the module, and the full file specification of the file containing the module
- A list of all global variables used in the build
- A list of all procedures used in the build

Index

A

@ command • 4–32
Abort
  resulting from exceeding virtual address space • 5–1
ABORT statement • 3–26, 3–33, 7–16
Action routine
  designating for client messages • 7–357
  detached cursor
    defining • 7–367
    fetching • 7–197
  for handling client messages
    fetching • 7–197
Active area • 7–350
  determining location of • 7–196
Active editing point • 2–4
ADD_KEY_MAP built-in procedure • 7–17 to 7–18
ADJUST_WINDOW built-in procedure • 7–19 to 7–23
Algorithm
  for naming buffer change journal file • 1–12
ALL keyword
  with EXPAND_NAME • 7–135
  with REMOVE_KEY_MAP • 7–313
  with SET (BELL) • 7–355
  with SET (DEBUG) • 7–364
  with UPDATE • 7–538
Alternation
  pattern (|) • 2–16
Anchored search • 7–24
ANCHOR keyword • 7–24 to 7–25
  with SEARCH • 7–327, 7–328
  with SEARCH_QUIETLY • 7–332
AND operator • 3–7
"Ansi_crt" string constant parameter to GET_INFO • 7–196
ANY built-in procedure • 7–26 to 7–27
APPEND_LINE built-in procedure • 7–28 to 7–29
Application
  use of DECwindows VAXTPU built-in procedures in • B–1 to B–33
ARB built-in procedure • 7–30 to 7–31
Arithmetic expression • 3–9
ARRAY data type • 2–2 to 2–3
  See also CREATE_ARRAY built-in procedure
ASCII built-in procedure • 7–32 to 7–34
Assignment statement • 3–21
ATTACH built-in procedure • 7–35 to 7–36
Attribute
  buffer • 7–60
  window • 7–78
Attribute for TPU
  setting records • 7–448
AUTO_REPEAT keyword • 7–353
"Auto_repeat" string constant parameter to GET_INFO • 7–196

B

Base
  of numeric constant
    specifying • 3–37
Batch job • 5–5
Batch-like editing • 5–3
BEGINNING_OF built-in procedure • 7–37 to 7–38
BELL keyword • 7–355
  with SET (MESSAGE_ACTION_TYPE) • 7–426
"Bell" string constant parameter to GET_INFO • 7–205
"Beyond_eob" string constant parameter to GET_INFO • 7–185
"Beyond_eol" string constant parameter to GET_INFO • 7–185, 7–220
BLANK_TABS keyword • 7–483
BLINK keyword
  with MARK • 7–261
  with SELECT • 7–337
  with SET (PROMPT_AREA) • 7–446
  with SET (STATUS_LINE) • 7–476
  with SET (VIDEO) • 7–492
"Blink_status" string constant parameter to GET_INFO • 7–221
"Blink_video" string constant parameter to GET_INFO • 7–221
BOLD keyword
  with MARK • 7–261
  with SELECT • 7–337
  with SET (PROMPT_AREA) • 7–446
  with SET (STATUS_LINE) • 7–476
  with SET (VIDEO) • 7–492
"Bold_status" string constant parameter to GET_INFO • 7–221

"Bold_video" string constant parameter to GET_INFO • 7–221
Boolean expression • 3–11
Bound marker • 2–9 to 2–10
"Bound" string constant parameter to GET_INFO • 7–171, 7–185, 7–221
BREAK built-in procedure • 7–39
"Breakpoint" string constant parameter to GET_INFO • 7–179
BROADCAST keyword
  with SET (BELL) • 7–355
Buffer
  attributes • 7–60
  controlling modification indicator • 7–431
  converting contents of to string format using STR • 7–520
  converting name to journal file name • 7–172
  current • 7–59
  deleting • 7–107
  determining if unmodifiable records are present in • 7–175
  direction
    current • 7–85
    setting • 7–379
  erasing • 2–4, 7–117
  erasing unmodifiable records from
    preventing or allowing • 7–375
  getting file name of journal • 7–172
  journal file • 1–11
  margin action settings • 7–414, 7–456
  margin settings • 7–412, 7–419, 7–454
  multiple • 7–59
  recovering contents of • 7–307
  sensing safe journaling • 7–175
  sensing unmodifiable records erasable state • 7–169
  tab stops • 7–481
  variables • 2–4
  visible • 7–59
Buffer, multiple • 2–4
Buffer change journaling • 1–11
  and keystroke journaling • 7–307
  converting buffer to journal file name • 7–172
  default file naming • 1–12
  enabling • 7–405
  getting file name of journal • 7–172
  getting information on journal file • 7–203
  recovery • 7–307
  sensing safe state • 7–175
  sensing the enable • 1–12, 5–10
  specifying file name • 7–405
BUFFER command
  for message buffer • 4–18
BUFFER data type • 2–3 to 2–4
Buffer names • 2–4
"Buffer" string constant parameter to GET_INFO • 7–185, 7–193, 7–222
BUFFER_BEGIN keyword • 7–69, 7–273
  with POSITION • 7–287
  with SEARCH • 7–327
  with SEARCH_QUIETLY • 7–332
BUFFER_END keyword • 7–69, 7–273
  with POSITION • 7–287
  with SEARCH • 7–327
  with SEARCH_QUIETLY • 7–332
Building applications on EVE • G–1 to G–12
Built-in procedure
  descriptions • 7–15 to 7–548
  functions listed • 7–1 to 7–15
  name of as reserved word • 3–12
  occluded • 3–12

C

Callable interface • 4–1, 7–41
Callback data structure
  of widget
    using in VAXTPU • 7–496
Callback routines
  levels of • 4–9
Callbacks • 4–8 to 4–10
  handling in EVE • 4–11
CALL_USER built-in procedure • 7–40 to 7–43
Case sensitivity
  of widget names • 7–74
CASE statement • 3–23 to 3–25
Case-style error handler • 3–28 to 3–31
CHANGE_CASE built-in procedure • 7–44 to 7–46
Character-cell measuring system
  converting to coordinate system • 7–50
Character set • 3–1
"Character" string constant parameter to GET_INFO • 7–171
Character_cell display • 5–8
Child
  of widget
    fetching in VAXTPU • 7–210
Children
  of widget
    fetching in VAXTPU • 7–210

"children" string constant parameter to GET_INFO • 7–210
Class
  of widget
    fetching in VAXTPU • 7–214
  of widget resource
    fetching in VAXTPU • 7–215
"class" string constant parameter to GET_INFO • 7–214
Client message
  designating routine to handle • 7–357
  fetching action routine for handling • 7–197
  finding out type of • 7–197
  sending from VAXTPU • 7–344
CLIENT_MESSAGE
  keyword parameter to SET built-in procedure • 7–357
"client_message" string constant parameter to GET_INFO • 7–197
"client_message_routine" string constant parameter to GET_INFO • 7–197
Clipboard
  fetching data from • 7–149
  overview of • 7–149
  reading data from • 7–295
  writing data to • 7–540
Closures • 4–11
COLUMN_MOVE_VERTICAL keyword • 7–359
"Column_move_vertical" string constant parameter to GET_INFO • 7–206
Command files • 4–29 to 4–31
  debugging • 4–34
  default • 4–21
  definition • 1–10
  sample • 4–30
Command line
  DCL
    determining whether /RECOVER specified on • 7–408
  fetching values from • 7–176, 7–177
  /JOURNAL command qualifier • 1–11, 1–12
  /NOJOURNAL command qualifier • 1–12
  /RECOVER command qualifier • 1–11, 7–307
Command parameter
  See EDIT/TPU command parameter
/COMMAND qualifier • 4–25, 5–3 to 5–4, 5–6 to 5–7
Command qualifiers
  See EDIT/TPU command qualifiers
"Command" string constant parameter to GET_INFO • 7–176
Command synonyms • G–5 to G–7
Command window
  in EVE • 4–16
"Command_file" string constant parameter to GET_INFO • 7–176
Comment character • 1–5
COMMENT keyword
  with LOOK_UP_KEY • 7–254
Compilation
  conditional • 3–36
COMPILE built-in procedure • 4–19, 7–47 to 7–49
Compiler limits • 7–47
Compiling
  in a VAXTPU buffer • 4–19
  in EVE • 4–19
  programs • 4–18 to 4–19
  to create section file • 4–24
Concatenation
  pattern (+) • 2–15
  string • 3–4
Conditional compilation • 3–36
Conditional statements • 3–22 to 3–23
Constant
  specifying radix of • 3–37
  TPU$K_DISJOINT • 7–198, 7–368
  TPU$K_INVISIBLE • 7–198, 7–368
  TPU$K_OFF_LEFT • 7–198, 7–368
  TPU$K_OFF_RIGHT • 7–198, 7–368
  TPU$K_UNMAPPED • 7–198, 7–368
CONSTANT declaration • 3–35
Constants • 3–5 to 3–6
  local • 3–20
  predefined • 3–13
Control character
  entering • 3–2
  translation
    example • A–2
Control code
  function key • 7–241
Control sequence
  function key • 7–241
Conventions • xxiv
CONVERT built-in procedure • 7–50
  example of use • B–1 to B–4
Coordinate measuring system
  converting to character-cell system • 7–50
COPY_TEXT built-in procedure • 7–53 to 7–54
/CREATE qualifier • 5–7
"Create" string constant parameter to GET_INFO • 7–177
CREATE_ARRAY built-in procedure • 7–55 to 7–57
CREATE_BUFFER built-in procedure • 7–58 to 7–62, 7–203

CREATE_KEY_MAP built-in procedure • 7–63 to 7–64
CREATE_KEY_MAP_LIST built-in procedure • 7–65 to 7–66
CREATE_PROCESS built-in procedure • 7–67 to 7–68
CREATE_RANGE built-in procedure • 7–69 to 7–71
CREATE_WIDGET built-in procedure • 7–72
  example of use • B–4 to B–11
  using to specify callback routine • 4–9
  using to specify resource values • 4–12
CREATE_WINDOW built-in procedure • 2–26, 7–77 to 7–79
CROSS_WINDOW_BOUNDS keyword • 7–361
"Cross_window_bounds" string constant parameter to GET_INFO • 7–197
CTRL/C • 4–20
  with case-style error handler • 3–29, 3–30
  with procedural error handler • 3–27, 3–28
Current buffer • 7–59
  active editing point • 2–4
  definition • 7–80
Current buffer direction • 7–85
Current date • 7–138, 7–268, 7–271
Current pointer position • 7–252
"Current" string constant parameter to GET_INFO • 7–166, 7–167, 7–169, 7–184, 7–191, 7–218
Current time • 7–138, 7–268, 7–271
Current window • 2–27, 7–77
CURRENT_BUFFER built-in procedure • 7–80
CURRENT_CHARACTER built-in procedure • 7–81 to 7–82
CURRENT_COLUMN built-in procedure • 7–83 to 7–84
"Current_column" string constant parameter to GET_INFO • 7–197, 7–222
CURRENT_DIRECTION built-in procedure • 7–85
CURRENT_LINE built-in procedure • 7–86 to 7–87
CURRENT_OFFSET built-in procedure • 7–88 to 7–89
CURRENT_ROW built-in procedure • 7–90 to 7–91
"Current_row" string constant parameter to GET_INFO • 7–197, 7–222
CURRENT_WINDOW built-in procedure • 7–92 to 7–93
Cursor
  detached
    defining routine to handle • 7–367
    fetching action routine to handle • 7–197
    fetching reason for • 7–198
Cursor movement • 7–94, 7–96
  free • 7–95
Cursor position
  compared to editing point • 6–10
  effect of scrolling on • 7–324
  padding effects • 6–11 to 6–12
CURSOR_HORIZONTAL built-in procedure • 7–94
CURSOR_VERTICAL built-in procedure • 7–96 to 7–98

D

Data type
  checking • 4–12, 7–432
  definition • 2–1
  keywords
    ARRAY • 2–2 to 2–3
    BUFFER • 2–3 to 2–4
    INTEGER • 2–5
    KEYWORD • 2–5 to 2–7
    LEARN • 2–7 to 2–8
    MARK • 2–8 to 2–10
    PATTERN • 2–11 to 2–20
    PROCESS • 2–20 to 2–21
    PROGRAM • 2–21
    RANGE • 2–21 to 2–22
    STRING • 2–23 to 2–24
    UNSPECIFIED • 2–24
    WIDGET • 2–24 to 2–25
    WINDOW • 2–25 to 2–29
Data types • 1–6 to 1–7
Date
  inserting with FAO • 7–138
  inserting with MESSAGE • 7–268
  inserting with MESSAGE_TEXT • 7–271
DCL command line
  overriding /RECOVER qualifiers on • 7–408
DCL command procedure
  example • A–5
$DEBUG$INI$ buffer • 4–22
DEBUG command • 4–35
Debugger
  invoking • 4–33
Debugging • 4–33 to 4–37
  ATTACH command • 4–36
  CANCEL BREAKPOINT command • 4–36
  command files • 4–34
  DEPOSIT command • 4–36
  DISPLAY SOURCE command • 4–36
  EXAMINE command • 4–36
  GO command • 4–34, 4–36
  HELP command • 4–36

Debugging (Cont.)
program • 4–35
QUIT command • 4–36
SCROLL command • 4–37
section files • 4–34
SET BREAK POINT command • 4–34, 4–37
SET WINDOW command • 4–37
SHIFT command • 4–37
SHOW BREAKPOINTS command • 4–37
source code • 4–35
SPAWN command • 4–37
STEP command • 4–35, 4–37
to examine contents of local variable • 4–36
TPU command • 4–37
DEBUG keyword • 7–362, 7–363, 7–364
DEBUGON procedure • 4–35
/DEBUG qualifier • 4–33, 5–8
DEBUG_LINE built-in procedure • 7–99
DEC Multinational Character Set • 3–1 to 3–2, E–1 to E–8
DECwindows
version of VAXTPU
sample uses of built-ins • B–1 to B–33
DECwindows VAXTPU
determining if present • 7–197
invoking with /DISPLAY • 5–8
DEC_CRT2 mode • C–3
"Dec_crt2" string constant parameter to GET_INFO • 7–197
DEC_CRT mode • C–2
"Dec_crt" string constant parameter to GET_INFO • 7–197
Default directory
fetching in VAXTPU • 7–206
setting in VAXTPU • 7–366
Default file naming algorithm
buffer change journal • 1–12
$DEFAULTS$ buffer • 4–32
DEFAULT_DIRECTORY parameter to SET built-in procedure • 7–366
"default_directory" string constant parameter to GET_INFO • 7–206
"Defined" string constant parameter to GET_INFO • 7–190
DEFINE_KEY built-in procedure • 7–100 to 7–104
DEFINE_WIDGET_CLASS built-in procedure • 7–105
example of use • B–4 to B–11
DELETE built-in procedure • 7–107 to 7–110
Deleting records • 6–5
Deletion
buffer • 2–4
line terminator • 7–28
Deletion (Cont.)
marker • 2–10
range • 2–22, 7–70
subprocess • 7–67
VAXTPU structure • 7–109
window • 2–28
Detached cursor
defining routine to handle • 7–367
fetching action routine to handle • 7–197
fetching reason for • 7–198
DETACHED_ACTION parameter to SET built-in • 7–367
"detached_action" string constant parameter to GET_INFO • 7–197
"detached_reason" string constant parameter to GET_INFO • 7–198
DEVICE keyword
with FILE_PARSE • 7–140
with FILE_SEARCH • 7–143
Direction
of buffer • 7–85
setting • 7–379
"Direction" string constant parameter to GET_INFO • 7–171
Directory
default
fetching in VAXTPU • 7–206
setting in VAXTPU • 7–366
DIRECTORY keyword
with FILE_PARSE • 7–140
with FILE_SEARCH • 7–143
Display
definition of in VAXTPU • 4–16
Displaying version number • 4–2
/DISPLAY qualifier • 5–8
See also /NODISPLAY
"Display" string constant parameter to GET_INFO • 7–177, 7–206
Display value
fetching • 7–222
setting for window • 7–370
setting records • 7–448
DISPLAY_VALUE parameter to SET built-in procedure • 7–370
"display_value" string constant parameter to GET_INFO • 7–186, 7–222
Drag operation
determining where started • 7–188
Dynamic selection
in EVE • 4–16 to 4–17

E

EDIT built-in procedure • 7–111 to 7–114
Editing context status
  built-in procedures
    CURRENT_BUFFER • 7–80
    CURRENT_CHARACTER • 7–81
    CURRENT_COLUMN • 7–83
    CURRENT_DIRECTION • 7–85
    CURRENT_LINE • 7–86
    CURRENT_OFFSET • 7–88
    CURRENT_ROW • 7–90
    CURRENT_WINDOW • 7–92
    DEBUG_LINE • 7–99
    ERROR • 7–123
    ERROR_LINE • 7–125
    ERROR_TEXT • 7–127
  built-in procedures for defining
    SET • 7–347
    SHOW • 7–505
Editing interface
  See EVE
Editing point
  built-in procedures for moving
    MARK • 7–261
    MOVE_HORIZONTAL • 7–278
    MOVE_VERTICAL • 7–282
    POSITION • 7–287
  compared to cursor position • 6–10
  effect of scrolling on • 7–324
EDIT/TPU command • 1–9, 5–1 to 5–20
  parameter • 5–19
  qualifiers • 5–5 to 5–20
    /COMMAND • 5–6 to 5–7
    /CREATE • 5–7
    /DEBUG • 4–33, 5–8
    /DISPLAY • 5–8
    /INITIALIZATION • 5–9 to 5–10
    /INTERFACE • 5–10
    /JOURNAL • 5–10
    /MODIFY • 5–12
    /OUTPUT • 5–12
    /READ_ONLY • 5–13
    /RECOVER • 5–14, 7–408
    /SECTION • 5–16
    /START_POSITION • 5–17
    /WRITE • 5–17
EDIT/TPU command qualifiers • 1–9 to 1–10
"Edit_mode" string constant parameter to GET_INFO • 7–198
"Eightbit" string constant parameter to GET_INFO • 7–198
ELSE clause • 3–22
%ELSE lexical keyword • 3–36
%ENDIF lexical keyword • 3–36
ENDIF statement • 3–22 to 3–23
ENDLOOP statement • 3–21 to 3–22
ENDMODULE statement • 3–14 to 3–15
ENDON_ERROR statement • 3–25 to 3–31
ENDPROCEDURE statement • 3–15 to 3–21
END_OF built-in procedure • 7–115 to 7–116
Entering control characters • 3–2
EOB_TEXT keyword • 7–374
"Eob_text" string constant parameter to GET_INFO • 7–171
EQUIVALENCE statement • 3–33 to 3–34
ERASE built-in procedure • 7–117 to 7–118
ERASE_CHARACTER built-in procedure • 7–119 to 7–120
ERASE_LINE built-in procedure • 7–121 to 7–122
ERASE_UNMODIFIABLE
  keyword parameter to SET built-in procedure • 7–375
ERASE_UNMODIFIABLE mode
  and APPEND_LINE • 7–376
  and CHANGE_CASE • 7–376
  and COPY_TEXT • 7–376
  and EDIT • 7–376
  and ERASE (buffer) • 7–376
  and ERASE (range) • 7–376
  and ERASE_CHARACTER • 7–376
  and ERASE_LINE • 7–376
  and FILL • 7–376
  and MOVE_TEXT • 7–376
  and SPLIT_LINE • 7–376
  and TRANSLATE • 7–377
"erase_unmodifiable" string constant parameter GET_INFO built-in • 7–169
"Erase_unmodifiable" string constant parameter to GET_INFO • 7–171
Erasing unmodifiable records • 7–375
Error
  resulting from exceeding virtual address space • 5–1
Error handler
  case-style • 3–28 to 3–31
  procedural • 3–26 to 3–28
Error handling • 3–25 to 3–31, 4–38
ERROR lexical element • 3–25
ERROR statement • 7–123 to 7–124
ERROR_LINE lexical element • 3–26
ERROR_LINE statement • 7–125 to 7–126

ERROR_TEXT lexical element • 3–26
ERROR_TEXT statement • 7–127 to 7–128
EVE
  building applications on • G–1 to G–12
  command window • 4–16
  $DEFAULTS$ buffer • 4–32
  initialization files • 4–31 to 4–33
    during a session • 4–32
    effects on buffer settings • 4–32
  Initialization files • 5–10
  input files • 5–20
  message buffer • 4–18
  message window • 4–16
  order of initialization • G–4
  output file • 5–13, 5–20
  restriction on defining GOLD key • 7–472
  sample procedures • B–1 to B–33
  source files • 4–3
  status line • G–7
  use of EDIT/TPU command qualifiers • 5–18
  user window • 4–16
  wildcard characters in file specifications • 5–20
  wildcards in file names • 5–20
EVE$BUILD • G–1 to G–12
  exit and quit handlers • G–8
  initialization modules • G–4 to G–5
  invoking • G–10 to G–11
  output • G–11 to G–12
  status line field • G–7 to G–8
  synonym creation • G–5 to G–7
  using parsing routines with • G–3 to G–4
EVE$GET_STATUS_FIELDS procedure • G–8
EVE$INIT logical name • 4–31
EVE$PARSER_DISPATCH procedure • G–3
EVE$SELECTION procedure
  using to obtain EVE's current selection • 4–17
EVE default settings • 4–32 to 4–33
EVE source files • 1–11
EXACT keyword
  with LEARN_BEGIN • 7–244
  with SEARCH • 7–328
  with SEARCH_QUIETLY • 7–333
"Examine" string constant parameter to GET_INFO • 7–179
Examples of DECwindows VAXTPU built-in procedures • B–1 to B–33
Examples of VAXTPU procedures
  ADJUST_HELP • 7–23
  ANCHOR • 7–25
  ANY • 7–27
  APPEND_LINE • 7–29
Examples of VAXTPU procedures (Cont.)
  ARB • 7–31
  ASCII • 7–33, 7–34
  BEGINNING_OF • 7–38
  BREAK • 7–39
  CALL_USER • 7–42
  CHANGE_CASE • 7–46
  COPY_TEXT • 7–54
  CREATE BUFFER • 7–62
  CREATE_KEY_MAP • 7–64
  CREATE_KEY_MAP_LIST • 7–66
  CREATE_PROCESS • 7–68
  CREATE_RANGE • 7–71
  CREATE_WINDOW • 7–79
  CURRENT_BUFFER • 7–80
  CURRENT_CHARCTER • 7–82
  CURRENT_COLUMN • 7–84
  CURRENT_DIRECTION • 7–85
  CURRENT_LINE • 7–87
  CURRENT_OFFSET • 7–89
  CURRENT_ROW • 7–91
  CURRENT_WINDOW • 7–93
  CURRSOR_HORIZONTAL • 7–95
  CURSOR_VERTICAL • 7–98
  DEFINE_KEY • 7–103
  DELETE • 7–109
  EDIT • 7–114
  END_OF • 7–116
  ERASE • 7–118
  ERASE_CHARACTER • 7–120
  ERROR • 7–124
  ERROR_LINE • 7–126
  ERROR_TEXT • 7–128
  EXECUTE • 7–131, 7–132
  EXPAND_NAME • 7–137
  FAO • 7–139
  FILE_PARSE • 7–142
  FILE_SEARCH • 7–145
  GET_INFO • 7–160 to 7–161
  HELP_TEXT • 7–229
  INDEX • 7–231
  INT • 7–233
  KEY_NAME • 7–240
  LENGTH • 7–248
  LINE_BEGIN • 7–250
  LINE_END • 7–251
  LOCATE_MOUSE • 7–253
  LOOKUP_KEY • 7–256 to 7–257
  MAP • 7–260
  MARK • 7–263
  MATCH • 7–265
  MESSAGE • 7–269

Examples of VAXTPU procedures (Cont.)
MOVE_HORIZONTAL • 7–279
MOVE_TEXT • 7–281
MOVE_VERTICAL • 7–283
NOTANY • 7–285
PAGE_BREAK • 7–286
POSITION • 7–290
QUIT • 7–292
READ_CHAR • 7–294
READ_FILE • 7–298
READ_KEY • 7–302
REFRESH • 7–311
REMAIN • 7–312
RETURN • 7–315
SAVE • 7–318
SCAN • 7–320 to 7–321
SCANL • 7–323
SCROLL • 7–326
SEARCH • 7–330 to 7–331
SEARCH_QUIETLY • 7–335 to 7–336
SELECT • 7–339
SELECT_RANGE • 7–341
SEND • 7–343
SET (AUTO_REPEAT) • 7–354
SET (BELL) • 7–356
SET (DEBUG) • 7–365
SET (LINE_NUMBER) • 7–417
SET (SELF_INSERT) • 7–471
SET (TEXT) • 7–485
SET (TRACEBACK) • 7–489
SLEEP • 7–509
SPANL • 7–514
SPLIT_LINE • 7–519
STR • 7–522
SUBSTR • 7–524
TRANSLATE • 7–528
UNANCHOR • 7–531
UNDEFINE_KEY • 7–533
UNMAP • 7–537
UPDATE • 7–539
WRITE_FILE • 7–545
EXECUTE built-in procedure • 4–19
EXIT built-in procedure • 7–133 to 7–134
EXITIF statement • 3–21 to 3–22
EXPAND_NAME built-in procedure • 7–135 to 7–137
Expressions • 3–8 to 3–12
arithmetic • 3–9
Boolean • 3–11
evaluation by compiler • 3–9
pattern • 3–11
relational • 3–10
Expressions (Cont.)
types of • 3–9
Extensible VAX Editor
See EVE

F

FACILITY_NAME keyword • 7–378
"Facility_name" string constant parameter to GET_INFO • 7–206
FAO built-in procedure • 7–138 to 7–139
FAO directives
with MESSAGE • 7–267
with MESSAGE_TEXT • 7–270
Fatal internal error
resulting from exceeding virtual address space • 5–1
File
default name for journaling • 1–12
File organization • F–1
"File_name" string constant parameter to GET_INFO • 7–171, 7–177
FILE_PARSE built-in procedure • 7–140 to 7–142
FILE_SEARCH built-in procedure • 7–143 to 7–145
FILL built-in procedure • 7–146 to 7–148
"Find_buffer" string constant parameter to GET_INFO • 7–169
"first" string parameter to ADD_KEY_MAP • 7–17
"First" string constant parameter to GET_INFO • 7–166, 7–167, 7–169, 7–181, 7–183, 7–184, 7–191, 7–218
"First_marker" string constant parameter to GET_INFO • 7–172
"First_range" string constant parameter to GET_INFO • 7–172
FORWARD keyword • 7–85, 7–379
with SEARCH • 7–328
with SEARCH_QUIETLY • 7–333
Found range selection
in EVE • 4–18
Free cursor movement • 7–95, 7–96
Free marker • 2–9 to 2–10
Free markers • 7–70
FREE_CURSOR keyword
with MARK • 7–261
Function key
control code • 7–241
control sequence • 7–241
Function procedures • 3–19

G

Gadget • 2–25
GET_CLIPBOARD built-in procedure • 7–149
  example of use • B–11 to B–13
GET_DEFAULT built-in procedure • 7–151
GET_GLOBAL_SELECT built-in procedure • 7–153
  example of use • B–13 to B–15
GET_INFO built-in procedure • 7–156 to 7–161
  buffer variable parameter
    "read_routine" • 7–174, 7–201
  COMMAND_LINE keyword parameter
    "line" • 7–176, 7–177
  key_name parameter
    "key_modifiers" • 7–162
  marker_variable parameter
    "record_number" • 7–186
  mouse_event_keyword parameter
    "mouse_button" • 7–188
    "window" • 7–188
  SCREEN keyword parameter
    "active_area" • 7–196
    "decwindows" • 7–197
    "event" • 7–199
    "global_select" • 7–199
    "grab_routine" • 7–199
    "icon_name" • 7–199
    "input_focus" • 7–199
    "length" • 7–199
    "new_length" • 7–200
    "new_width" • 7–200
    "old_length" • 7–200
    "old_width" • 7–200
    "original_length" • 7–200
    "read_routine" • 7–201
    "screen_limits" • 7–201
    "time" • 7–202
    "ungrab_routine" • 7–202
  string constant parameter
    "active_area" • 7–196
    "Ansi_crt" • 7–196
    "auto_repeat" • 7–196
    "bell" • 7–205
    "beyond_eob" • 7–185
    "beyond_eol" • 7–185, 7–220
    "blink_status" • 7–221
    "blink_video" • 7–221
    "bold_status" • 7–221
    "bold_video" • 7–221
    "bottom" • 7–222
GET_INFO built-in procedure
  string constant parameter (Cont.)
    "bound" • 7–171, 7–185, 7–221
    "breakpoint" • 7–179
    "buffer" • 7–185, 7–193, 7–222
    "callback_parameters" • 7–209
    "callback_routine" • 7–214
    "character" • 7–171
    "children" • 7–210
    "class" • 7–214
    "client_message" • 7–197
    "client_message_routine" • 7–197
    "column_move_vertical" • 7–206
    "command" • 7–176
    "command_file" • 7–176
    "create" • 7–177
    "cross_window_bounds" • 7–197
    "current" • 7–166, 7–167, 7–169, 7–184, 7–191, 7–218
    "current_column" • 7–197, 7–222
    "current_row" • 7–197, 7–222
    "decwindows" • 7–197
    "dec_crt2" • 7–197
    "dec_crt" • 7–197
    "default_directory" • 7–206
    "defined" • 7–190
    "detached_action" • 7–197
    "detached_reason" • 7–198
    "direction" • 7–171
    "display" • 7–177, 7–206
    "display_value" • 7–186, 7–222
    "edit_mode" • 7–198
    "eightbit" • 7–198
    "enable_resize" • 7–206
    "eob_text" • 7–171
    "erase_unmodifiable" • 7–169, 7–171
    "event" • 7–199
    "examine" • 7–179
    "facility_name" • 7–206
    "file_name" • 7–171, 7–177
    "find_buffer" • 7–169
    "first" • 7–166, 7–167, 7–169, 7–181, 7–183, 7–184, 7–191, 7–218
    "first_marker" • 7–172
    "first_range" • 7–172
    "global_select" • 7–199
    "grab_routine" • 7–199
    "high_index" • 7–167
    "icon_name" • 7–199
    "informational" • 7–206
    "initialization" • 7–177
    "initialization_file" • 7–177

GET_INFO built-in procedure
string constant parameter (Cont.)
"init_file" • 7–177
"input_focus" • 7–199
"is_managed" • 7–214
"is_subclass" • 7–214
"journaling" • 1–12, 5–10, 7–172
"journaling_frequency" • 7–206
"journal" • 7–177, 7–203
"journal_file" • 1–12, 5–11, 7–172, 7–177, 7–206
"journal_name" • 7–172
"key_map_list" • 7–222
"key_map_list" • 7–172
"key_modifiers" • 7–162
"key_type" • 7–162
"last" • 7–166, 7–167, 7–169, 7–181, 7–183, 7–184, 7–191, 7–218
"left" • 7–222
"left_margin" • 7–172, 7–186
"left_margin_action" • 7–172
"length" • 7–199, 7–223
"line" • 7–176, 7–177
"line" • 7–172
"line_editing" • 7–199
"line_number" • 7–179, 7–206
"local" • 7–179
"map_count" • 7–173
"maximum_parameters" • 7–190
"max_lines" • 7–173
"menu_position" • 7–210
"message_action_level" • 7–206
"message_action_type" • 7–206
"message_flags" • 7–207
"middle_of_tab" • 7–223
"minimum_parameters" • 7–190
"mode" • 7–173
"modifiable" • 7–173
"modified" • 7–173
"modify" • 7–177
"mouse" • 7–200
"mouse_button" • 7–188
"name" • 7–215
"name" • 7–164, 7–173, 7–182
"new_length" • 7–200
"new_width" • 7–200
"next" • 7–166, 7–168, 7–169, 7–180, 7–181, 7–183, 7–184, 7–191, 7–218, 7–223
"next_marker" • 7–173
"next_range" • 7–173
"nomodify" • 7–177
"no_video" • 7–223

GET_INFO built-in procedure
string constant parameter (Cont.)
"no_video_status" • 7–223
"no_write" • 7–174
"offset" • 7–174, 7–186
"offset_column" • 7–174, 7–186
"old_length" • 7–200
"old_width" • 7–200
"original_bottom" • 7–223
"original_length" • 7–200
"original_length" • 7–223
"original_top" • 7–223
"original_width" • 7–200
"output" • 7–177
"output_file" • 7–174, 7–178
"pad" • 7–223
"pad_overstruck_tabs" • 7–207
"parameter" • 7–180
"parent" • 7–215
"permanent" • 7–174
"pid" • 7–192
"post_key_procedure" • 7–204
"previous" • 7–166, 7–168, 7–169, 7–180, 7–181, 7–183, 7–184, 7–191, 7–218, 7–223
"pre_key_procedure" • 7–204
"procedure" • 7–180
"prompt_length" • 7–200
"prompt_row" • 7–201
"read_only" • 7–178
"read_routine" • 7–174, 7–201
"record_count" • 7–175
"record_number" • 7–186
"record_number" • 7–175
"record_size" • 7–175
"recover" • 7–207
"recover" • 7–178
"resize_action" • 7–207
"resources" • 7–215
"reverse_status" • 7–224
"reverse_video" • 7–224
"right" • 7–224
"right_margin" • 7–175, 7–186
"right_margin_action" • 7–175
"safe_for_journaling" • 7–175
"screen_limits" • 7–201
"screen_update" • 7–201
"scroll" • 7–201, 7–224
"scroll_amount" • 7–224
"scroll_bar" • 7–224
"scroll_bar_auto_thumb" • 7–224
"scroll_bottom" • 7–224

GET_INFO built-in procedure
string constant parameter (Cont.)
"scroll_top" • 7–225
"section" • 7–178
"section_file" • 7–178, 7–207
"self_insert" • 7–204
"shift_amount" • 7–225
"shift_key" • 7–204, 7–207
"special_graphics_status" • 7–225
"start_character" • 7–178
"start_record" • 7–178
"status_line" • 7–225
"status_video" • 7–225
"success" • 7–207
"system" • 7–175
"tab_stops" • 7–175
"text" • 7–215
"text" • 7–225
"time" • 7–202
"timed_message" • 7–207
"timer" • 7–207
"top" • 7–225
"traceback" • 7–207
"type" • 7–165
"undefined_key" • 7–204
"underline_status" • 7–225
"underline_video" • 7–225
"ungrab_routine" • 7–202
"unmodifiable_records" • 7–175, 7–186, 7–193
"update" • 7–208
"version" • 7–208
"video" • 7–187, 7–193, 7–226
"visible" • 7–226
"visible_bottom" • 7–226
"visible_length" • 7–202, 7–226
"visible_top" • 7–226
"vk100" • 7–202
"vt100" • 7–202
"vt200" • 7–202
"vt300" • 7–202
"widget_id" • 7–209
"widget_info" • 7–216
"width" • 7–226
"width" • 7–202
"window" • 7–188
"within_range" • 7–187
"write" • 7–178
SYSTEM keyword parameter
"enable_resize" • 7–206
"recover" • 7–207
"resize_action" • 7–207
GET_INFO built-in procedure
SYSTEM keyword parameter (Cont.)
"timer" • 7–207
WIDGET keyword parameter
"callback_parameters" • 4–11, 7–209
"widget_id" • 7–209
widget variable parameter
"name" • 7–215
"text" • 7–215
"widget_info" • 7–216
widget_variable parameter
"callback_routine" • 7–214
window variable parameter
"left" • 7–222
"length" • 7–223
"right" • 7–224
"scroll_bar" • 7–224
"scroll_bar_auto_thumb" • 7–224
"top" • 7–225
"width" • 7–226
window_variable parameter
"bottom" • 7–222
example of use • B–16 to B–19, B–19 to B–22
"key_map_list" • 7–222
Global selection
determining ownership of • 7–199
fetching grab routine for • 7–199
fetching information about • 7–153
fetching read request for • 7–199
fetching read routine for • 7–174, 7–201
fetching ungrab routine for • 7–202
fetching wait time for • 7–202
obtaining data from • 7–300
reading information about • 7–299
requesting ownership of • 7–380
sending information about to an application • 7–546
specifying expiration period for • 7–387
specifying grab routine for • 7–382
specifying read routine for • 7–385
specifying ungrab routine for • 7–389
support for • 4–6 to 4–8
Global variable • 3–4
GOLD key
restriction on defining in EVE • 7–472
Grab routine
fetching event in • 7–199
global selection
fetching • 7–199
specifying • 7–382
input focus • 7–398

Grab routine
input focus (Cont.)
fetching • 7–199
specifying • 7–400
GRAPHIC_TABS keyword • 7–483

H

HEIGHT parameter to SET built-in procedure • 7–391
HELP_TEXT built-in procedure • 7–228 to 7–229
"High_index" string constant parameter to GET_INFO • 7–167

I

Icon
fetching text of • 7–199
implementing in DECwindows VAXTPU • 7–393, 7–395
specifying text for • 7–392
ICONIFY_PIXMAP parameter to SET built-in • 7–395
ICON_PIXMAP parameter to SET built-in • 7–393
Identifier • 3–4
Ident produced by EVE$BUILD • G–2
IDENT statement • 3–14 to 3–15
%IFDEF lexical keyword • 3–36
%IF lexical keyword • 3–36
IF statement • 3–22 to 3–23
INDEX built-in procedure • 7–230 to 7–231
INFORMATIONAL keyword • 7–397
"Informational" string constant parameter to GET_INFO • 7–206
INFO_WINDOW identifier • 7–506
INFO_WINDOW variable • 4–29
Initialization files
default handling • 4–22
definition • 1–11
during a session • 4–32
effects on buffer settings • 4–32
EVE • 4–31 to 4–33
/INITIALIZATION qualifier • 5–9 to 5–10
"Initialization" string constant parameter to GET_INFO • 7–177
"Initialization_file" string constant parameter to GET_INFO • 7–177
Initializing variables • 2–24
"Init_file" string constant parameter to GET_INFO • 7–177
Input files • 1–9, 5–19
Input focus
determining ownership of • 7–199
fetching grab routine for • 7–199
fetching ungrab routine for • 7–202
requesting • 7–398
specifying grab routine for • 7–400
specifying ungrab routine for • 7–402
support for • 4–5 to 4–6
INRANGE case constant • 3–24
Inserted records • 6–5
Inserting date • 7–138, 7–268, 7–271
Inserting time • 7–138, 7–268, 7–271
INSERT keyword • 7–404
Insert mode
COPY_TEXT • 7–53
MOVE_TEXT • 7–280
INT built-in procedure • 7–232 to 7–233
Integer constants • 3–5
INTEGER data type • 2–5
/INTERFACE qualifier • 5–10
Interruption of program • 4–20
Invisible record • 7–448
Invoking • 1–9
Invoking VAXTPU • 5–1
from a batch job • 5–5
from DCL command procedure • 5–2
interactively • 5–1
restriction to consider before • 5–1
"is_managed" string constant parameter to GET_INFO • 7–214
"is_subclass" string constant parameter to GET_INFO • 7–214

J

/JOURNAL command qualifier • 1–11, 1–12
Journal file • 7–307
default name • 1–12
getting characteristics of • 7–203
getting name of • 1–12, 5–11
recovering buffer contents • 7–307
security caution • 1–12, 7–59, 7–234, 7–235, 7–406
Journaling
buffer change • 1–11
converting buffer to journal file name • 7–172
default file name • 1–12
EVE default behavior • 1–12
getting file name of buffer change journal • 7–172

Journaling (Cont.)
getting journal file information • 7–203
keystroke
enabling and disabling • 7–408
layered application control • 1–12
recovery of buffer contents • 7–307
role of source file • 7–308
sensing a safe buffer • 7–175
sensing the enable of buffer change journaling • 1–12, 5–10
sensing the enable of keystroke journaling • 1–12, 5–11
using both keystroke and buffer change journaling • 1–12
JOURNALING keyword • 7–405
JOURNALING parameter
SET built-in procedure • 7–405
"journaling" string constant parameter
GET_INFO built-in • 1–12, 5–10
"Journaling" string constant parameter to GET_INFO • 7–172
"Journaling_frequency" string constant parameter to GET_INFO • 7–206
/JOURNAL qualifier • 5–10
"journal" string constant parameter
GET_INFO built-in • 7–203
"Journal" string constant parameter to GET_INFO • 7–177
JOURNAL_CLOSE built-in procedure • 7–234
"Journal_file" GET_INFO request_string • 7–177
"journal_file" string constant parameter
GET_INFO built-in • 1–12, 5–11, 7–172
"Journal_file" string constant parameter to GET_INFO • 7–206
"journal_name" string constant parameter
GET_INFO built-in • 7–172
JOURNAL_OPEN built-in procedure • 1–12, 5–11, 7–235 to 7–237
controlling errors related to • 7–408

K

Key
See also Key map
built-in procedures for defining
DEFINE_KEY • 7–100
LAST_KEY • 7–242
LOOKUP_KEY • 7–254
SET (POST_KEY_PROCEDURE) • 7–442
SET (PRE_KEY_PROCEDURE) • 7–444
Key
built-in procedures for defining (Cont.)
SET (SELF_INSERT) • 7–470
SET (UNDEFINED_KEY) • 7–490
UNDEFINE_KEY • 7–532
creating a name for • 7–238
Key map
built-in procedures
ADD_KEY_MAP • 7–17
CREATE_KEY_MAP • 7–63
REMOVE_KEY_MAP • 7–313
SHOW (KEY_MAP) • 7–505
SHOW (KEY_MAPS) • 7–505
Key map list
See also Key
built-in procedures
CREATE_KEY_MAP_LIST • 7–65
SET (KEY_MAP_LIST) • 7–410
SHOW (KEY_MAP_LIST) • 7–505
SHOW (KEY_MAP_LISTS) • 7–505
example of fetching • B–19 to B–22
Key name
table • 2–6
Keystroke journaling
and buffer change journaling • 7–307
comparative to buffer change journaling • 1–11
enabling and disabling • 7–408
sensing the enable • 1–12, 5–11
KEYSTROKE_RECOVERY keyword • 7–408
KEYSTROKE_RECOVERY parameter
SET built-in procedure • 7–408
Keyword • 3–12
ALL
with EXPAND_NAME • 7–135
with REMOVE_KEY_MAP • 7–313
with SET (BELL) • 7–355
with SET (DEBUG) • 7–364
with UPDATE • 7–538
ANCHOR • 7–24 to 7–25
with SEARCH • 7–327, 7–328
with SEARCH_QUIETLY • 7–332
BELL • 7–355
with SET (MESSAGE_ACTION_TYPE) • 7–426
BLANK_TABS • 7–483
BLINK
with SELECT • 7–337
with SET (PROMPT_AREA) • 7–446
with SET (STATUS_LINE) • 7–476
with SET (VIDEO) • 7–492
BOLD
with SELECT • 7–337

Keyword
BOLD (Cont.)
with SET (PROMPT_AREA) • 7–446
with SET (STATUS_LINE) • 7–476
with SET (VIDEO) • 7–492
BROADCAST
with SET (BELL) • 7–355
BUFFER_BEGIN
with POSITION • 7–287
with SEARCH • 7–327
with SEARCH_QUIETLY • 7–332
BUFFER_END
with POSITION • 7–287
with SEARCH • 7–327
with SEARCH_QUIETLY • 7–332
COMMENT
with LOOK_UP_KEY • 7–254
CROSS_WINDOW_BOUNDS • 7–361
DEBUG • 7–362, 7–363, 7–364
DEVICE
with FILE_PARSE • 7–140
with FILE_SEARCH • 7–143
DIRECTORY
with FILE_PARSE • 7–140
with FILE_SEARCH • 7–143
EOB_TEXT • 7–374
EXACT
with LEARN_BEGIN • 7–244
with SEARCH • 7–328
with SEARCH_QUIETLY • 7–333
FACILITY_NAME • 7–378
FORWARD • 7–85, 7–379
with SEARCH • 7–328
with SEARCH_QUIETLY • 7–333
GRAPHIC_TABS • 7–483
INFORMATIONAL • 7–397
INSERT • 7–404
JOURNALING • 7–405
key name • 2–6
KEYSTROKE_RECOVERY • 7–408
KEYWORDS
with EXPAND_NAME • 7–135
KEY_MAP
with LOOK_UP_KEY • 7–254
KEY_MAP_LIST • 7–410
LEFT_MARGIN • 7–412
LEFT_MARGIN_ACTION • 7–414
LINE_BEGIN • 7–249 to 7–250
with POSITION • 7–288
with SEARCH • 7–327
with SEARCH_QUIETLY • 7–332
Keyword (Cont.)
LINE_END • 7–251
with POSITION • 7–288
with SEARCH • 7–327
with SEARCH_QUIETLY • 7–332
LINE_NUMBER • 7–416
MARGINS • 7–419
MAX_LINES • 7–421
MESSAGE_FLAGS • 7–427
MODIFIABLE • 7–429
MOUSE
with POSITION • 7–288, 7–289
NAME
with FILE_PARSE • 7–141
with FILE_SEARCH • 7–144
NODE
with FILE_PARSE • 7–140
with FILE_SEARCH • 7–143
NONE
with SELECT • 7–337
with SET (MESSAGE_ACTION_TYPE) • 7–426
with SET (PROMPT_AREA) • 7–446
with SET (STATUS_LINE) • 7–476
with SET (VIDEO) • 7–492
NO_EXACT
with LEARN_BEGIN • 7–244
with SEARCH • 7–328
with SEARCH_QUIETLY • 7–333
NO_TRANSLATE • 7–483
NO_WRITE • 7–434
occluded • 3–12
OFF
with CREATE_WINDOW • 7–77
with HELP_TEXT • 7–228
with QUIT • 7–291
with SET (AUTO_REPEAT) • 7–353
with SET (BELL) • 7–355
with SET (COLUMN_MOVE_VERTICAL) • 7–359
with SET (CROSS_WINDOW_BOUNDS) • 7–361
with SET (DEBUG) • 7–363, 7–364
with SET (INFORMATIONAL) • 7–397
with SET (LINE_NUMBER) • 7–416
with SET (MODIFIABLE) • 7–429
with SET (MOUSE) • 7–432
with SET (NO_WRITE) • 7–434
with SET (PAD) • 7–437
with SET (PAD_OVERSTRUCK_TABS) • 7–439
with SET (SCREEN_UPDATE) • 7–460

Keyword
OFF (Cont.)
with SET (SCROLLING) • 7–467
with SET (SELF_INSERT) • 7–470
with SET (SUCCESS) • 7–479
with SET (TIMER) • 7–486
with SET (TRACEBACK) • 7–488
with SPAWN • 7–515
ON
with CREATE WINDOW • 7–77
with CREATE_WINDOW • 7–77
with HELP_TEXT • 7–228
with QUIT • 7–291
with SET (AUTO_REPEAT) • 7–353
with SET (BELL) • 7–355
with SET (COLUMN_MOVE_VERTICAL) • 7–359
with SET (CROSS_WINDOW_BOUNDS) • 7–361
with SET (DEBUG) • 7–363
with SET (INFORMATIONAL) • 7–397
with SET (LINE_NUMBER) • 7–416
with SET (MODIFIABLE) • 7–429
with SET (MOUSE) • 7–432
with SET (NO_WRITE) • 7–434
with SET (PAD) • 7–437
with SET (PAD_OVERSTRUCK_TABS) • 7–439
with SET (SCREEN_UPDATE) • 7–460
with SET (SCROLLING) • 7–467
with SET (SELF_INSERT) • 7–470
with SET (SUCCESS) • 7–479
with SET (TIMER) • 7–486
with SET (TRACEBACK) • 7–488
with SPAWN • 7–515
OUTPUT_FILE • 7–435
OVERSTRIKE • 7–436
PAD • 7–437
PAD_OVERSTRUCK_TABS • 7–439
PAGE BREAK • 7–286
PAGE_BREAK
with SEARCH • 7–327
with SEARCH_QUIETLY • 7–332
PERMANENT • 7–441
POST_KEY_PROCEDURE • 7–442
PROCEDURES
with EXPAND_NAME • 7–135
PROGRAM • 7–362
with LOOK_UP_KEY • 7–254
PROMPT_AREA • 7–446
REMAIN • 7–312
with SEARCH • 7–327
Keyword
REMAIN (Cont.)
with SEARCH_QUIETLY • 7–332
returned by CURRENT_DIRECTION • 7–85
returned by READ_KEY • 7–301
REVERSE • 7–85, 7–453
with SEARCH • 7–328
with SEARCH_QUIETLY • 7–333
with SELECT • 7–337
with SET (MESSAGE_ACTION_TYPE) • 7–426
with SET (PROMPT_AREA) • 7–446
with SET (STATUS_LINE) • 7–476
with SET (VIDEO) • 7–492
RIGHT_MARGIN • 7–454
RIGHT_MARGIN_ACTION • 7–456
SCREEN_UPDATE • 7–460
SCROLLING • 7–467
SELF_INSERT • 7–470
SHIFT_KEY • 7–472
SPECIAL_GRAPHICS
with SET (STATUS_LINE) • 7–476
STATUS_LINE • 7–476
SUCCESS • 7–479
SYSTEM • 7–480
TEXT • 7–483
TIMER • 7–486
TRACEBACK • 7–488
TYPE
with FILE_PARSE • 7–141
with FILE_SEARCH • 7–144
UNANCHOR • 7–530 to 7–531
with SEARCH_QUIETLY • 7–333
UNDEFINED_KEY • 7–490
UNDERLINE
with SELECT • 7–337
with SET (PROMPT_AREA) • 7–446
with SET (STATUS_LINE) • 7–476
with SET (VIDEO) • 7–492
VARIABLES
with EXPAND_NAME • 7–135
VERSION
with FILE_PARSE • 7–141
with FILE_SEARCH • 7–144
VIDEO • 7–492
with SET • 7–347 to 7–348
with SHOW • 7–505 to 7–506
Keyword constants • 3–5
KEYWORD data type • 2–5 to 2–7
Keywords
lexical • 3–36

KEYWORDS keyword
with EXPAND_NAME • 7–135
KEY_MAP keyword
with LOOK_UP_KEY • 7–254
KEY_MAP_LIST keyword • 7–410
"Key_map_list" string constant parameter to GET_INFO • 7–172
KEY_NAME built-in procedure • 7–238 to 7–241
"Key_type" string constant parameter to GET_INFO • 7–162
KILL_SELECTION client message • 7–344

L

"last" string parameter to ADD_KEY_MAP • 7–17
"Last" string constant parameter to GET_INFO • 7–166, 7–167, 7–169, 7–181, 7–183, 7–184, 7–191, 7–218
LAST_KEY built-in procedure • 7–242
LEARN data type • 2–7 to 2–8
LEARN_ABORT built-in procedure • 7–243
LEARN_BEGIN built-in procedure • 7–244 to 7–246
LEARN_END built-in procedure • 7–244 to 7–246
Left margin
setting records • 7–448
LEFT_MARGIN keyword • 7–412
"Left_margin" string constant parameter to GET_INFO • 7–172, 7–186
LEFT_MARGIN_ACTION keyword • 7–414
"Left_margin_action" string constant parameter to GET_INFO • 7–172
LENGTH built-in procedure • 7–247 to 7–248
Lexical element • 3–1
Lexical keywords • 3–36 to 3–38
Line break
in data from global selection • 7–300
LINE command • 4–18
Line mode editing • C–3
Line-mode editor
example • A–1
"Line" string constant parameter to GET_INFO • 7–172
Line terminator
deleting • 7–28
LINE_BEGIN keyword • 7–69, 7–249 to 7–250, 7–273
with POSITION • 7–288
with SEARCH • 7–327
with SEARCH_QUIETLY • 7–332
"Line_editing" string constant parameter to GET_INFO • 7–199
LINE_END keyword • 7–69, 7–251, 7–273
with POSITION • 7–288
with SEARCH • 7–327
with SEARCH_QUIETLY • 7–332
LINE_NUMBER keyword • 7–416
"Line_number" string constant parameter to GET_INFO • 7–179, 7–206
List
specifying as a resource value • 4–13
$LOCAL$INI$ buffer • 4–22
LOCAL declaration • 3–34 to 3–35
"Local" string constant parameter to GET_INFO • 7–179
Local variable • 3–4, 3–20
Local variables • 3–34
LOCATE_MOUSE built-in procedure • 7–252 to 7–253
Logical names
EVE$INIT • 4–31
TPU$COMMAND • 5–6
TPU$DEBUG • 5–8
TPU$SECTION • 5–16
Logical operators
AND operator • 3–7
NOT operator • 3–7
OR operator • 3–7
XOR operator • 3–7
Longword
to convert with FAO • 7–138
to convert with MESSAGE • 7–268
to convert with MESSAGE_TEXT • 7–271
LOOKUP_KEY built-in procedure • 7–254 to 7–257
LOOP statement • 3–21 to 3–22
"Low_index" string constant parameter to GET_INFO • 7–167

M

Main window widget • 4–16
MANAGE CHILDREN routine
See MANAGE_WIDGET built-in procedure
MANAGE CHILD routine
See MANAGE_WIDGET built-in procedure
MANAGE_WIDGET built-in procedure • 7–258
example of use • B–4 to B–11
Managing
of widget
controlling whether causes mapping • 7–418

MAP built-in procedure • 7–259 to 7–260
MAPPED_WHEN_MANAGED parameter to SET built-in procedure • 7–418
Mapping
of widget
controlling whether performed during managing • 7–418
"Map_count" string constant parameter to GET_INFO • 7–173
Margin
default • 7–412, 7–419, 7–454
left
setting records • 7–448
setting • 7–412, 7–419, 7–454
margin action
setting • 7–414
Margin action
default • 7–414
Margin Action
default • 7–456
setting • 7–456
MARGINS keyword • 7–419
MARK built-in procedure • 7–261 to 7–263
MARK data type • 2–8 to 2–10
Marker
deleting • 2–10, 7–108
determining if record containing is unmodifiable • 7–186
fetching display value of record containing • 7–186
padding effects • 2–10
video attributes • 2–9, 7–261
MATCH built-in procedure • 7–264 to 7–265
"Maximum_parameters" string constant parameter to GET_INFO • 7–190
MAX_LINES keyword • 7–421
"Max_lines" string constant parameter to GET_INFO • 7–173
Measurement
converting units of • 7–50
Memory
error resulting from exceeding • 5–1
Menu bar widget • 4–16
Menu position
of widget
fetching in VAXTPU • 7–210
setting in VAXTPU • 7–422
MENU_POSITION parameter to SET built-in procedure • 7–422
"menu_position" string constant parameter to GET_INFO • 7–210
Message buffer • 4–18
MESSAGE built-in procedure • 7–266 to 7–269
Messages • D–1 to D–10
Message window
in EVE • 4–16
MESSAGE_ACTION_LEVEL keyword • 7–424
"Message_action_level" string constant parameter to GET_INFO • 7–206
MESSAGE_ACTION_TYPE keyword • 7–426
MESSAGE_BUFFER identifier • 7–266
MESSAGE_BUFFER variable • 4–29
MESSAGE_FLAGS keyword • 7–427
"Message_flags" string constant parameter to GET_INFO • 7–207
MESSAGE_TEXT built-in procedure • 7–270 to 7–272
"Middle_of_tab" string constant parameter to GET_INFO • 7–223
Minimal interface example • 4–26
"Minimum_parameters" string constant parameter to GET_INFO • 7–190
"Mode" string constant parameter to GET_INFO • 7–173
Modifiability
setting records • 7–448
MODIFIABLE keyword • 7–429
"Modifiable" string constant parameter to GET_INFO • 7–173
"Modified" string constant parameter to GET_INFO • 7–173
/MODIFY qualifier • 5–12
"Modify" string constant parameter to GET_INFO • 7–177
MODIFY_RANGE built-in procedure • 7–273 to 7–277
Module declaration
syntax • 3–15
MODULE statement • 3–14 to 3–15
Modules used with EVE$BUILD • G–2
Mouse
determining support for • 7–432
determining where drag operation originated • 7–188
Mouse button
fetching information about • 7–188
MOUSE keyword • 7–432
with POSITION • 7–288, 7–289
Mouse pad
implementing • B–4
"Mouse" string constant parameter to GET_INFO • 7–200
MOVE_HORIZONTAL built-in procedure • 7–278 to 7–279
MOVE_TEXT built-in procedure • 7–280 to 7–281

MOVE_VERTICAL built-in procedure • 7–282 to 7–283
Multinational Character Set
  See DEC Multinational Character Set
Multiple buffers • 7–59

N

Name
  widget
    case sensitivity of • 7–74
NAME keyword
  with FILE_PARSE • 7–141
  with FILE_SEARCH • 7–144
Names for procedures • 3–16
"Name" string constant parameter to GET_INFO • 7–164, 7–173, 7–182
"Next" string constant parameter to GET_INFO • 7–166, 7–168, 7–169, 7–180, 7–181, 7–183, 7–184, 7–191, 7–218, 7–223
"Next_marker" string constant parameter to GET_INFO • 7–173
"Next_range" string constant parameter to GET_INFO • 7–173
NODE keyword
  with FILE_PARSE • 7–140
  with FILE_SEARCH • 7–143
/NODISPLAY qualifier
  effect on LAST_KEY • 7–242
  to disable screen manager • 6–1
  with EVE$BUILD • G–10
/NOJOURNAL command qualifier • 1–12
"Nomodify" string constant parameter to GET_INFO • 7–177
NONE keyword
  with MARK • 7–261
  with SELECT • 7–337
  with SET (MESSAGE_ACTION_TYPE) • 7–426
  with SET (PROMPT_AREA) • 7–446
  with SET (STATUS_LINE) • 7–476
  with SET (VIDEO) • 7–492
NOTANY built-in procedure • 7–284 to 7–285
NOT operator • 3–7
NO_EXACT keyword
  with LEARN_BEGIN • 7–244
  with SEARCH • 7–328
  with SEARCH_QUIETLY • 7–333
NO_TRANSLATE keyword • 7–483
"No_video" string constant parameter to GET_INFO • 7–223
"No_video_status" string constant parameter to GET_INFO • 7–223
"No_write" GET_INFO request_string • 7–174
NO_WRITE keyword • 7–434
Null parameters • 3–18
Numeric constant
  specifying radix of • 3–37

O

OFF keyword
  with CREATE_WINDOW • 7–77
  with HELP_TEXT • 7–228
  with QUIT • 7–291
  with SET (AUTO_REPEAT) • 7–353
  with SET (BELL) • 7–355
  with SET (COLUMN_MOVE_VERTICAL) • 7–359
  with SET (CROSS_WINDOW_BOUNDS) • 7–361
  with SET (DEBUG) • 7–363, 7–364
  with SET (INFORMATIONAL) • 7–397
  with SET (LINE_NUMBER) • 7–416
  with SET (MODIFIABLE) • 7–429
  with SET (MOUSE) • 7–432
  with SET (NO_WRITE) • 7–434
  with SET (PAD) • 7–437
  with SET (PAD_OVERSTRUCK_TABS) • 7–439
  with SET (SCREEN_UPDATE) • 7–460
  with SET (SCROLLING) • 7–467
  with SET (SELF_INSERT) • 7–470
  with SET (SUCCESS) • 7–479
  with SET (TIMER) • 7–486
  with SET (TRACEBACK) • 7–488
  with SPAWN • 7–515
"Offset" string constant parameter to GET_INFO • 7–174, 7–186
"Offset_column" string constant parameter to GET_INFO • 7–174, 7–186
ON keyword
  with CREATE_WINDOW • 7–77
  with HELP_TEXT • 7–228
  with QUIT • 7–291
  with SET (AUTO_REPEAT) • 7–353
  with SET (BELL) • 7–355
  with SET (COLUMN_MOVE_VERTICAL) • 7–359
  with SET (CROSS_WINDOW_BOUNDS) • 7–361
  with SET (DEBUG) • 7–363
  with SET (INFORMATIONAL) • 7–397
  with SET (LINE_NUMBER) • 7–416
  with SET (MODIFIABLE) • 7–429
  with SET (MOUSE) • 7–432

ON keyword (Cont.)
with SET (NO_WRITE) • 7–434
with SET (PAD) • 7–437
with SET (PAD_OVERSTRUCK_TABS) • 7–439
with SET (SCREEN_UPDATE) • 7–460
with SET (SCROLLING) • 7–467
with SET (SELF_INSERT) • 7–470
with SET (SUCCESS) • 7–479
with SET (TIMER) • 7–486
with SET (TRACEBACK) • 7–488
with SPAWN • 7–515
ON_ERROR statement • 3–25 to 3–31
location • 3–25
ON_ERROR Statement • 3–21
Operators • 3–6 to 3–8
partial pattern assignment (@) • 2–17
pattern alternation (|) • 2–16
pattern concatenation (+) • 2–15
pattern linking (&) • 2–15
precedence • 3–7
relational • 2–18
"Original_bottom" string constant parameter to GET_INFO • 7–223
"Original_length" string constant parameter to GET_INFO • 7–223
"Original_top" string constant parameter to GET_INFO • 7–223
"Original_width" string constant parameter to GET_INFO • 7–200
OR operator • 3–7
Output file • 5–12
OUTPUT parameter
SET built-in procedure • 7–203
/OUTPUT qualifier • 5–12
"Output" string constant parameter to GET_INFO • 7–177
OUTPUT_FILE keyword • 7–435
"Output_file" string constant parameter to GET_INFO • 7–174, 7–178
OUTRANGE case constant • 3–24
OVERSTRIKE keyword • 7–436
Overstrike mode
COPY_TEXT • 7–53
MOVE_TEXT • 7–280
Ownership
global selection
determining • 7–199
losing • 7–202
requesting • 7–380
input focus
determining • 7–199
losing • 7–202
Ownership
input focus (Cont.)
requesting • 7–398

P

Padding effects • 6–11 to 6–12
version differences • 7–439
with APPEND_LINE • 7–28
with ATTACH • 7–35
with COPY_TEXT • 7–53
with CURRENT_CHARACTER • 7–81
with CURRENT_LINE • 7–86
with CURRENT_OFFSET • 7–88
with ERASE_CHARACTER • 7–119
with ERASE_LINE • 7–121
with MARK • 7–262
with MOVE_HORIZONTAL • 7–278
with MOVE_TEXT • 7–281
with MOVE_VERTICAL • 7–282
with READ_FILE • 7–297
with SELECT • 7–338
with SELECT_RANGE • 7–341
with SET (PAD) • 7–437
with SPAWN • 7–516
with SPLIT_LINE • 7–518
PAD keyword • 7–437
"Pad" string constant parameter to GET_INFO • 7–223
PAD_OVERSTRUCK_TABS keyword • 7–439
"Pad_overstruck_tabs" string constant parameter to GET_INFO • 7–207
PAGE_BREAK keyword • 7–286
with SEARCH • 7–327
with SEARCH_QUIETLY • 7–332
Parameters
for procedures • 3–16 to 3–19
"Parameter" string constant parameter to GET_INFO • 7–180
Parent
of widget
fetching in VAXTPU • 7–215
""parent"" string constant parameter to GET_INFO • 7–215
Parentheses
in expressions • 3–7
Parser
maximum stack depth of • 4–2
Parsers with EVE$BUILD • G–3 to G–4
Partial pattern assignment (@) • 2–17

Pattern
alternation (|) • 2–16
anchoring • 7–24
built-in procedures • 2–13
compilation • 2–18
concatenation (+) • 2–15
execution • 2–18
expression • 3–11
linking (&) • 2–15
operators • 2–15
searching • 2–11
Pattern assignment
partial (@) • 2–17
PATTERN data type • 2–11 to 2–20
Pattern matching
built-in procedures
ANCHOR • 7–24
ANY • 7–26
ARB • 7–30
LINE_BEGIN • 7–249
LINE_END • 7–251
MATCH • 7–264
NOTANY • 7–284
PAGE_BREAK • 7–286
REMAIN • 7–312
SCAN • 7–319
SCANL • 7–322
SPAN • 7–510
SPANL • 7–512
UNANCHOR • 7–530
PERMANENT keyword • 7–441
"Permanent" string constant parameter to GET_INFO • 7–174
"Pid" string constant parameter to GET_INFO • 7–192
Pixmap
use of to implent icon in DECwindows VAXTPU • 7–393, 7–395
Pointer position • 7–252
POSITION built-in procedure • 7–287 to 7–290
example of use • B–25 to B–27
POST_KEY_PROCEDURE keyword • 7–442
"Post_key_procedure" string constant parameter to GET_INFO • 7–204
Predefined constants
names • 3–13
"Previous" string constant parameter to GET_INFO • 7–166, 7–168, 7–169, 7–180, 7–181, 7–183, 7–184, 7–191, 7–218, 7–223
PRE_KEY_PROCEDURE keyword • 7–444
"Pre_key_procedure" string constant parameter to GET_INFO • 7–204
Procedural error handler • 3–26 to 3–28
Procedure
executing • 4–21
name • 3–16
parameter • 3–16 to 3–19
recommended naming conventions • 4–31
recommended size for • 4–2
recursive • 3–19
returning result • 2–8, 3–19, 7–101
using LEARN_ABORT in • 7–243
Procedures
samples using EVE • B–1 to B–33
PROCEDURES keyword
with EXPAND_NAME • 7–135
PROCEDURE statement • 3–15 to 3–21
"Procedure" string constant parameter to GET_INFO • 7–180
Process
deleting • 7–108
multiple
built-in procedures
ATTACH • 7–35
CREATE_PROCESS • 7–67
RECOVER_BUFFER • 7–307
SEND • 7–342
SEND_EOF • 7–346
SPAWN • 7–515
PROCESS data type • 2–20 to 2–21
Program
add to section file • 4–25
calling VAXTPU from • 4–1, 7–41
compiling • 4–18 to 4–19
complex • 4–2
debugging • 4–33 to 4–37
deleting • 7–108
executing • 4–19 to 4–21
interrupting • 4–20
order • 4–3
simple • 4–2
syntax • 4–3
example • 4–4
writing • 4–1 to 4–14
PROGRAM data type • 2–21
Program execution
built-in procedures
COMPILE • 7–47
SAVE • 7–316
PROGRAM keyword • 7–362
with LOOK_UP_KEY • 7–254
PROMPT_AREA
video attributes • 7–446

PROMPT_AREA keyword • 7–446
"Prompt_length" string constant parameter to GET_INFO • 7–200
"Prompt_row" string constant parameter to GET_INFO • 7–201

Q

Qualifier, command
  See EDIT/TPU command qualifiers
QUIT built-in procedure • 7–291 to 7–292
Quote characters • 7–112, 7–113

R

Radix
  of numeric constant
    specifying • 3–37
Range
  converting contents of to string format using STR • 7–520
  deleting • 2–22, 7–70, 7–108
  determining if unmodifiable records are present in • 7–193
  erasing • 2–22, 7–70, 7–117
  moving delimiters of • 7–273
  video attributes • 2–22
RANGE data type • 2–21 to 2–22
Read request
  fetching • 7–199
Read routine
  fetching • 7–174, 7–201
  specifying • 7–385
READ_CHAR built-in procedure • 7–293 to 7–294
READ_CLIPBOARD built-in procedure • 7–295
READ_FILE built-in procedure • 7–297 to 7–298
READ_GLOBAL_SELECT built-in procedure • 7–299
  example of use • B–28 to B–30, B–30 to B–31
READ_KEY built-in procedure • 7–301 to 7–302
READ_LINE built-in procedure • 7–303 to 7–305
/READ_ONLY qualifier • 5–13
"Read_only" string constant parameter to GET_INFO • 7–178
REALIZE_WIDGET built-in procedure • 7–306
Realizing
  widgets in VAXTPU • 7–306
Record
  determining if unmodifiable is present • 7–175, 7–186, 7–193
  erasing unmodifiable
    preventing or allowing • 7–375
  fetching display value of • 7–186
  sensing unmodifiable erasable state • 7–169
  setting attribute • 7–448
Record attribute • F–1
Record deleting • 6–5
Record format • F–1
Record insertion • 6–5
RECORD_ATTRIBUTE parameter to SET built-in procedure • 7–448
"Record_count" string constant parameter to GET_INFO • 7–175
"Record_number" string constant parameter to GET_INFO • 7–175
"Record_size" string constant parameter to GET_INFO • 7–175
/RECOVER command qualifier • 1–11, 7–307
"Recover" GET_INFO request_string • 7–178
/RECOVER qualifier • 5–11, 5–14
  controlling errors related to • 7–408
Recovery
  of buffer contents • 1–11, 7–307
  role of source file • 7–308
  using buffer change journaling • 7–307
  using keystroke journal file
    enabling and disabling • 7–408
RECOVER_BUFFER built-in procedure • 7–307 to 7–309
Recursive procedure • 3–19
REFRESH built-in procedure • 6–10, 7–310 to 7–311
  compared with UPDATE (ALL) • 7–538
Relational expression • 3–10
Relational operators • 2–18
REMAIN keyword • 7–312
  with SEARCH • 7–327
  with SEARCH_QUIETLY • 7–332
Removal of key map
  built-in procedures
    REMOVE_KEY_MAP • 7–313
Removal of window • 2–28
REMOVE_KEY_MAP built-in procedure • 7–313 to 7–314
Repetitive statements • 3–21 to 3–22
Reserved word
  built-in procedures • 3–12
  keywords • 3–12
  language elements • 3–13 to 3–14
  predefined constants • 3–13

Resizing
of screen in VAXTPU • 7–391, 7–501
Resource
of widget
fetching class and data type of • 7–215
supported data types for • 4–12
"resources" string constant parameter to GET_INFO • 7–215
Restoring terminal width
example • A–5
Restriction
VAXTPU
virtual address space • 5–1
Restrictions
for subprocess • 2–20
RETURN statement • 3–26, 3–31 to 3–33, 7–315
REVERSE keyword • 7–85, 7–453
with MARK • 7–261
with SEARCH • 7–328
with SEARCH_QUIETLY • 7–333
with SELECT • 7–337
with SET (MESSAGE_ACTION_TYPE) • 7–426
with SET (PROMPT_AREA) • 7–446
with SET (STATUS_LINE) • 7–476
with SET (VIDEO) • 7–492
"Reverse_status" string constant parameter to GET_INFO • 7–224
"Reverse_video" string constant parameter to GET_INFO • 7–224
RIGHT_MARGIN keyword • 7–454
"Right_margin" string constant parameter to GET_INFO • 7–175, 7–186
RIGHT_MARGIN_ACTION keyword • 7–456
"Right_margin_action" string constant parameter to GET_INFO • 7–175
Running VAXTPU from subprocess
example • A–5

S

"safe_for_journaling" string constant parameter GET_INFO built-in • 7–175
Sample procedures using DECwindows VAXTPU built-in procedures • B–1 to B–33
Sample VAXTPU procedures
debugon • 7–365
delete_all_definitions • 7–533
init_help_key_map_list • 7–66
init_sample_key_map • 7–64
line_number_example • 7–417
Sample VAXTPU procedures (Cont.)
mail_sub • 7–343
my_call_user • 7–43
remove_comments • 7–312
SAVE • 7–318
shift_key_handler • 7–257
show_key_maps_in_list • 7–161
show_key_map_lists • 7–160
show_self_insert • 7–161
strip_blanks • 7–124, 7–126, 7–128
strip_eight • 7–528
toggle_self_insert • 7–471
traceback_example • 7–489
user_change_mode • 7–103
user_change_windows • 7–290
user_clear_key • 7–533
user_collect_rnos • 7–145
user_dcl_process • 7–68
user_define_edtkey • 7–240
user_define_key • 7–103
user_delete • 7–89
user_delete_char • 7–29
user_delete_extra • 7–109
user_delete_key • 7–120
user_display_current_character • 7–82
user_display_help • 7–23
user_display_key_map_list • 7–160
user_display_position • 7–522
user_do • 7–131
user_double_parens • 7–265
user_edit_string • 7–114
user_emphasize_message • 7–509
user_end_of_line • 7–251
user_erase_message_buffer • 7–315
user_erase_to_eob • 7–71
user_error_messsage • 7–139
user_fao_conversion • 7–139
user_find_chap • 7–330, 7–335
user_find_mark_twain • 7–514
user_find_parens • 7–320
user_find_procedure • 7–27
user_find_string • 7–315
user_free-cursor_up • 7–98
user_free_cursor_down • 7–98
user_free_cursor_left • 7–95
user_free_cursor_right • 7–95
user_get_info • 7–160
user_get_key_info • 7–256
user_go_down • 7–91
user_go_up • 7–91
user_help • 7–229

Sample VAXTPU procedures (Cont.)
user_help_buffer • 7–62
user_help_on_key • 7–302
user_include_file • 7–38
user_initial_cap • 7–524
user_is_character • 7–231
user_lowercase_line • 7–46
user_make_window • 7–79
user_mark • 7–248
user_message_window • 7–260
user_move_8_lines • 7–283
user_move_by_lines • 7–279
user_move_text • 7–281
user_move_to_mouse • 7–253
user_next_page • 7–286
user_next_screen • 7–93
user_not_quite_working • 7–39
user_one_window_to_two • 7–537
user_on_eol • 7–269
user_paste • 7–116, 7–263
user_print • 7–485
user_prompt_number • 7–233, 7–305
user_quick_parse • 7–137
user_quit • 7–292
user_quote • 7–294
user_remove_blank_lines • 7–514
user_remove_comments • 7–25
user_remove_crlfs • 7–118
user_remove_dsrlines • 7–250
user_remove_non_numbers • 7–323
user_remove_numbers • 7–514
user_remove_odd_characters • 7–321
user_remove_paren_text • 7–531
user_repaint • 7–311
user_replace_prefix • 7–31
user_ring_bell • 7–356
user_runoff_line • 7–87
user_scroll_buffer • 7–326
user_search_for_nonalpha • 7–285
user_search_range • 7–331, 7–336
user_select • 7–341
user_show_direction • 7–85
user_show_first_line • 7–539
user_simple_insert • 7–54
user_slow_down_arrow • 7–354
user_slow_up_arrow • 7–354
user_split_line • 7–84, 7–519
user_start_journal • 7–142
user_start_select • 7–339
user_tab • 7–33
user_test_key • 7–34

Sample VAXTPU procedures (Cont.)
user_toggle_direction • 7–80
user_top • 7–38
user_tpu • 7–132
user_trans_text • 7–528
user_two_window • 7–298
user_upcase_item • 7–46
user_what_is_comment • 7–256
user_write_file • 7–545
SAVE built-in procedure • 7–316 to 7–318
SCAN built-in procedure • 7–319 to 7–321
SCANL built-in procedure • 7–322 to 7–323
Screen
enabling resizing of • 7–372
resizing • 7–391, 7–501
specifying size of • 7–458
updating
controlling support for • 7–460
SCREEN keyword
using with widget-related built-in procedures • 4–16
Screen layout
built-in procedures
ADJUST_WINDOW • 7–19
CREATE_WINDOW • 7–77
MAP • 7–259
REFRESH • 7–310
SHIFT • 7–503
UNMAP • 7–536
UPDATE • 7–538
Screen manager • 2–28, 6–1 to 6–12
automatic update • 6–7
line changes • 6–6
partial update • 6–8
specific window update • 6–8
suppressing updates • 6–6
update all windows • 6–9
update order • 6–7
updates • 6–6
update with ADJUST_WINDOW • 7–22
update with CURSOR_HORIZONTAL • 7–94
update with CURSOR_VERTICAL • 7–97
Screen object
in VAXTPU • 4–14
Screen update
See Screen manager
SCREEN_UPDATE keyword • 7–460
"Screen_update" string constant parameter to GET_INFO • 7–201
Scroll bar
disabling • 7–462
enabling • 7–462

Scroll bar slider
adjusting automatically • 7–224
Scroll bar widget
example of fetching • B–19 to B–22
SCROLL built-in procedure • 6–10, 7–324 to 7–326
Scrolling
effect of on cursor position • 7–324
effect of on editing point • 7–324
with records deleted • 6–5
with records inserted • 6–5
SCROLLING keyword • 7–467
"Scroll" string constant parameter to GET_INFO • 7–201, 7–224
"Scroll_amount" string constant parameter to GET_INFO • 7–224
"Scroll_bottom" string constant parameter to GET_INFO • 7–224
"Scroll_top" string constant parameter to GET_INFO • 7–225
Search
anchored • 7–24
anchoring a pattern • 2–19
for pattern • 2–11
unanchoring pattern elements • 2–19 to 2–20
SEARCH built-in procedure • 7–327 to 7–331
SEARCH_QUIETLY built-in procedure • 7–332 to 7–336
Section files • 5–16
created with EVE$BUILD • G–10 to G–11
creating • 4–23
debugging • 4–34
default • 4–21
definition • 1–10
extending • 4–24
processing • 4–24, 4–25
recommended conventions • 4–28
/SECTION qualifier • 4–25, 5–16
"Section" string constant parameter to GET_INFO • 7–178
"Section_file" string constant parameter to GET_INFO • 7–178, 7–207
Security considerations • 1–12, 7–59, 7–234, 7–235, 7–406
SELECT built-in procedure • 7–337 to 7–339
Selection • 4–16
dynamic • 4–17
found range • 4–18
static • 4–17
using MODIFY_RANGE built-in to alter • 7–273
Select range
in EVE • 4–16
SELECT_RANGE built-in procedure • 7–340 to 7–341
SELF_INSERT keyword • 7–470
"Self_insert" string constant parameter to GET_INFO • 7–204
Semicolon
as statement separator • 1–8, 3–4, 3–15, 3–16, 3–17, 4–3
SEND built-in procedure • 7–342 to 7–343
SEND_CLIENT_MESSAGE built-in procedure • 7–344 to 7–345
SEND_EOF built-in procedure • 7–346
Separator
semicolon used as • 1–8, 3–4, 3–15, 3–16, 3–17, 4–3
SET (ACTIVE_AREA) built-in procedure • 7–350
SET (AUTO_REPEAT) built-in procedure • 7–353 to 7–354
SET (BELL) built-in procedure • 7–355 to 7–356
SET (CLIENT_MESSAGE) built-in procedure • 7–357 to 7–358
SET (COLUMN_MOVE_VERTICAL) built-in procedure • 7–359 to 7–360
SET (CROSS_WINDOW_BOUNDS) built-in procedure • 7–361
SET (DEBUG) built-in procedure • 7–362 to 7–365
SET (DEFAULT_DIRECTORY) built-in procedure • 7–366
SET (DETACHED_ACTION) built-in procedure • 7–367 to 7–369
SET (DISPLAY_VALUE) built-in procedure • 7–370
SET (DRM_HIERARCHY) built-in procedure • 7–371
SET (ENABLE_RESIZE) built-in procedure • 7–372
SET (EOB_TEXT) built-in procedure • 7–374
SET (ERASE_UNMODIFIABLE) built-in procedure • 7–375 to 7–377
SET (FACILITY_NAME) built-in procedure • 7–378
SET (FORWARD) built-in procedure • 7–379
SET (GLOBAL_SELECT) built-in procedure • 7–380
SET (GLOBAL_SELECT_GRAB) built-in procedure • 7–382
SET (GLOBAL_SELECT_READ) built-in procedure • 7–385
SET (GLOBAL_SELECT_TIME) built-in procedure • 7–387
SET (GLOBAL_SELECT_UNGRAB) built-in procedure • 7–389
SET (HEIGHT) built-in procedure • 7–391
SET (ICONIFY_PIXMAP) built-in procedure • 7–395 to 7–396
SET (ICON_NAME) built-in procedure • 7–392
SET (ICON_PIXMAP) built-in procedure • 7–393 to 7–394

SET (INFORMATIONAL) built-in procedure • 7–397
SET (INPUT_FOCUS) built-in procedure • 7–398
SET (INPUT_FOCUS_GRAB) built-in procedure • 7–400
SET (INPUT_FOCUS_UNGRAB) built-in procedure • 7–402
SET (INSERT) built-in procedure • 7–404
SET (JOURNALING) built-in procedure • 7–405 to 7–407
SET (KEYSTROKE_RECOVERY) built-in procedure • 7–408 to 7–409
SET (KEY_MAP_LIST) built-in procedure • 7–410 to 7–411
SET (LEFT_MARGIN) built-in procedure • 7–412 to 7–413
SET (LEFT_MARGIN_ACTION) built-in procedure • 7–414 to 7–415
SET (LINE_NUMBER) built-in procedure • 7–416 to 7–417
SET (MAPPED_WHEN_MANAGED) built-in procedure • 7–418
SET (MARGINS) built-in procedure • 7–419 to 7–420
SET (MAX_LINES) built-in procedure • 7–421
SET (MENU_POSITION) built-in procedure • 7–422 to 7–423
SET (MESSAGE_ACTION_LEVEL) built-in procedure • 7–424 to 7–425
SET (MESSAGE_ACTION_TYPE) built-in procedure • 7–426
SET (MESSAGE_FLAGS) built-in procedure • 7–427 to 7–428
SET (MODIFIABLE) built-in procedure • 7–429 to 7–430
SET (MODIFIED) built-in procedure • 7–431
SET (MOUSE) built-in procedure • 7–432 to 7–433
SET (NO_WRITE) built-in procedure • 7–434
SET (OUTPUT) built-in procedure • 7–203
SET (OUTPUT_FILE) built-in procedure • 7–435
SET (OVERSTRIKE) built-in procedure • 7–436
SET (PAD) built-in procedure • 7–437 to 7–438
SET (PAD_OVERSTRUCK_TABS) built-in procedure • 7–439 to 7–440
SET (PERMANENT) built-in procedure • 7–441
SET (POST_KEY_PROCEDURE) built-in procedure • 7–442 to 7–443
SET (PRE_KEY_PROCEDURE) built-in procedure • 7–444 to 7–445
SET (PROMPT_AREA) built-in procedure • 7–446 to 7–447
SET (RECORD_ATTRIBUTE) built-in procedure • 7–448 to 7–450
SET (RESIZE_ACTION) built-in procedure • 7–451
SET (REVERSE) built-in procedure • 7–453
SET (RIGHT_MARGIN) built-in procedure • 7–454 to 7–455
SET (RIGHT_MARGIN_ACTION) built-in procedure • 7–456 to 7–457
SET (SCREEN_LIMITS) built-in procedure • 7–458
SET (SCREEN_UPDATE) built-in procedure • 7–460 to 7–461
SET (SCROLLING) built-in procedure • 7–467 to 7–469
SET (SCROLL_BAR) built-in procedure • 7–462
  example of use • B–22 to B–25
SET (SCROLL_BAR_AUTO_THUMB) built-in procedure • 7–465
  example of use • B–22 to B–25
SET (SELF_INSERT) built-in procedure • 7–470 to 7–471
SET (SHIFT_KEY) built-in procedure • 7–472 to 7–473
SET (SPECIAL_ERROR_SYMBOL) built-in procedure • 7–474 to 7–475
SET (STATUS_LINE) built-in procedure • 7–476 to 7–478
SET (SUCCESS) built-in procedure • 7–479
SET (SYSTEM) built-in procedure • 7–480
SET (TAB_STOPS) built-in procedure • 7–481 to 7–482
SET (TEXT) built-in procedure • 7–483 to 7–485
SET (TIMER) built-in procedure • 7–486 to 7–487
SET (TRACEBACK) built-in procedure • 7–488 to 7–489
SET (UNDEFINED_KEY) built-in procedure • 7–490 to 7–491
SET (VIDEO) built-in procedure • 7–492 to 7–493
SET (WIDGET) built-in procedure • 7–494
  example of use • B–22 to B–25, B–25 to B–27
  using to specify resource values • 4–12
SET (WIDGET_CALLBACK) built-in procedure • 7–499
  example of use • B–22 to B–25
  using to specify callback routine • 4–9
SET (WIDGET_CALL_DATA) built-in procedure • 7–496 to 7–498
SET (WIDTH) built-in procedure • 7–501 to 7–502
SET built-in procedure • 7–347 to 7–349
  WIDGET • 4–10
SHIFT built-in procedure • 7–503 to 7–504
SHIFT key
  restriction on defining in EVE • 7–472
"Shift_amount" string constant parameter to GET_INFO • 7–225
SHIFT_KEY keyword • 7–472

"Shift_key" string constant parameter to GET_INFO • 7–204, 7–207
SHOW (KEYWORDS) built-in procedure • 2–5
SHOW built-in procedure • 7–505 to 7–507
SHOW DEFAULTS BUFFER command • 4–32
Showing version number • 4–2
SHOW_BUFFER identifier • 7–506
SHOW_BUFFER variable • 4–29
SLEEP built-in procedure • 7–508 to 7–509
Slider • 7–224
  example of fetching • B–19 to B–22
Source file
  defined • 7–308
Source files for EVE • 1–11
SPAN built-in procedure • 7–510 to 7–511
SPANL built-in procedure • 7–512 to 7–514
SPAWN built-in procedure • 7–515 to 7–517
SPECIAL_GRAPHICS keyword
  with SET (STATUS_LINE) • 7–476
"Special_graphics_status" string constant parameter to GET_INFO • 7–225
SPLIT_LINE built-in procedure • 7–518 to 7–519
Startup files • 1–10 to 1–11, 4–21 to 4–33
  command file • 1–10
  definition • 1–10
  initialization file • 1–10
  order of execution • 4–22
  section file • 1–10
"Start_character" string constant parameter to GET_INFO • 7–178
/START_POSITION qualifier • 5–17
"Start_record" string constant parameter to GET_INFO • 7–178
Statement
  separator for • 4–3
Static selection • 4–17
Status line
  default information • 7–77
  fields added with EVE$BUILD • G–7 to G–8
  video attributes • 7–476
STATUS_LINE keyword • 7–476
"Status_line" string constant parameter to GET_INFO • 7–225
"Status_video" string constant parameter to GET_INFO • 7–225
STR built-in procedure • 7–520 to 7–522
String
  concatenating • 3–4
  converting contents of buffer to using STR • 7–520
  converting contents of range to using STR • 7–520
  to insert with FAO • 7–138
  to insert with MESSAGE • 7–268
String (Cont.)
  to insert with MESSAGE_TEXT • 7–271
String constants • 3–5
STRING data type • 2–23 to 2–24
STUFF_SELECTION client message • 7–344
Subclass
  finding out if a widget is a member of • 7–214
Subprocess
  at DCL level • 7–67
  built-in procedures
    ATTACH • 7–35
    CREATE_PROCESS • 7–67
    RECOVER_BUFFER • 7–307
    SEND • 7–342
    SEND_EOF • 7–346
  built-in procedures for defining
    SPAWN • 7–515
  deleting • 7–67
  restrictions • 2–20
  running VAXTPU from • A–5
  within VAXTPU • 7–67
SUBSTR built-in procedure • 7–523 to 7–525
SUCCESS keyword • 7–479
"Success" string constant parameter to GET_INFO • 7–207
Supported terminals • 1–8
Symbols • 3–3 to 3–4
Synonyms for commands • G–5 to G–7
Syntax • 4–3
SYSTEM keyword • 7–480
"System" string constant parameter to GET_INFO • 7–175

T

TAB_STOPS keyword
  used with SET • 7–481
"Tab_stops" string constant parameter to GET_INFO • 7–175
Terminal
  behavior • C–1
  DEC_CRT2 • C–3
  restoring width • A–5
  setting • C–1 to C–3
    AUTO_REPEAT • C–2
    auxiliary keypad • C–2
    132 columns • C–2
    control sequence introducer • C–2
    CSI • C–2
    cursor • C–2

Terminal
  setting (Cont.)
    DEC_CRT • C–2
    edit mode • C–2
    eightbit characters • C–2
    scrolling • C–3
    video attributes • C–3
    wrap • C–4
  support • C–1
  width
    restoring • A–5
Terminal emulator • 6–4
Terminal support • 1–8
TEXT keyword • 7–483
Text manipulation
  built-in procedures
    APPEND_LINE • 7–28
    BEGINNING_OF • 7–37
    CHANGE_CASE • 7–44
    COPY_TEXT • 7–53
    CREATE_BUFFER • 7–58
    EDIT • 7–111
    END_OF • 7–115
    ERASE • 7–117
    ERASE_CHARACTER • 7–119
    ERASE_LINE • 7–121
    FILE_PARSE • 7–140
    FILE_SEARCH • 7–143
    FILL • 7–146
    MOVE_TEXT • 7–280
    READ_FILE • 7–297
    SEARCH • 7–327
    SEARCH_QUIETLY • 7–332
    SELECT • 7–337
    SELECT_RANGE • 7–340
    SPLIT_LINE • 7–518
    TRANSLATE • 7–526
    WRITE_FILE • 7–543
"Text" string constant parameter to GET_INFO • 7–225
%THEN lexical keyword • 3–36
Time
  inserting with FAO • 7–138
  inserting with MESSAGE • 7–268
  inserting with MESSAGE_TEXT • 7–271
"Timed_message" string constant parameter to GET_INFO • 7–207
TIMER keyword • 7–486
Title bar widget • 4–16
TPU$COMMAND logical name • 4–21, 5–6
TPU$DEBUG logical name • 5–8
TPU$INIT_PROCEDURE procedure • 4–22, 4–28
TPU$K_DISJOINT constant • 7–198, 7–368
TPU$K_INVISIBLE constant • 7–198, 7–368
TPU$K_OFF_LEFT constant • 7–198, 7–368
TPU$K_OFF_RIGHT constant • 7–198, 7–368
TPU$K_UNMAPPED constant • 7–198, 7–368
TPU$LOCAL_INIT procedure • 4–29
TPU$LOCAL_INIT_PROCEDURE procedure • 4–23
TPU$SECTION logical name • 4–21, 4–27, 5–16
TPU$STACKOVER status
  correcting • 4–2
TPU$WIDGET_INTEGER_CALLBACK callback routine • 4–9, 4–10
TPU$WIDGET_STRING_CALLBACK callback routine • 4–9, 4–10
TPU$X_MESSAGE_BUFFER variable • 4–29
TPU$X_SHOW_BUFFER variable • 4–29
TPU$X_SHOW_WINDOW variable • 4–29
TPU$_UNKLEXICAL error message • 3–38
TPU command • 4–19
TPU debugger • 4–33 to 4–37
  ATTACH command • 4–36
  CANCEL BREAKPOINT command • 4–36
  DEBUGON procedure • 4–35
  DEPOSIT command • 4–36
  DISPLAY SOURCE command • 4–36
  EXAMINE command • 4–36
  GO command • 4–34, 4–36
  HELP command • 4–36
  invoking • 4–33
  QUIT command • 4–36
  SCROLL command • 4–37
  SET BREAKPOINT command • 4–34, 4–37
  SET WINDOW command • 4–37
  SHIFT command • 4–37
  SHOW BREAKPOINTS command • 4–37
  SPAWN command • 4–37
  STEP command • 4–35, 4–37
  TPU command • 4–37
TRACEBACK keyword • 7–488
"Traceback" string constant parameter to GET_INFO • 7–207
TRANSLATE built-in procedure • 7–526 to 7–529
"Type" GET_INFO request_string • 7–165
TYPE keyword
  with FILE_PARSE • 7–141
  with FILE_SEARCH • 7–144

U

UNANCHOR keyword • 7–530 to 7–531

UNANCHOR keyword (Cont.)
with SEARCH_QUIETLY • 7–333
Unbound code
use of local variables in • 3–34
UNDEFINED_KEY keyword • 7–490
"Undefined_key" string constant parameter to GET_INFO • 7–204
UNDEFINE_KEY built-in procedure • 7–532 to 7–533
UNDERLINE keyword
with MARK • 7–261
with SELECT • 7–337
with SET (PROMPT_AREA) • 7–446
with SET (STATUS_LINE) • 7–476
with SET (VIDEO) • 7–492
"Underline_status" string constant parameter to GET_INFO • 7–225
"Underline_video" string constant parameter to GET_INFO • 7–225
Ungrab routine
global selection
fetching • 7–202
specifying • 7–389
input focus
fetching • 7–202
specifying • 7–402
UNMANAGE_WIDGET built-in procedure • 7–534
UNMAP built-in procedure • 7–536 to 7–537
Unmodifiable record • 7–448
determining if present • 7–175, 7–186, 7–193
preventing or allowing erasing of • 7–375
sensing erasable state • 7–169
"Unmodifiable_records" string constant parameter to GET_INFO • 7–175, 7–186, 7–193
UNSPECIFIED data type • 2–24
Unsupported terminals • 2–29
UPDATE built-in procedure • 6–9, 7–538 to 7–539
compared with REFRESH • 7–538
"Update" string constant parameter to GET_INFO • 7–208
Updating windows • 2–29
User window
in EVE • 4–16
Utility routines
forming the VAXTPU callable interface • 4–1, 7–41

V

Value(s)
assigning to widget resources • 4–10, 7–494
Variable
buffer • 2–4
global • 3–4
initializing • 2–24
local • 3–4, 3–20, 3–34
VARIABLE declaration • 3–36
Variables
recommended naming conventions • 4–31
VARIABLES keyword
with EXPAND_NAME • 7–135
VAXTPU
built-in procedures • 1–2
DECwindows • 1–2
journaling methods • 1–11
relationship with DECwindows features • 1–2
used with UIL • 1–4
VERSION keyword • 7–141
with FILE_SEARCH • 7–144
Version number • 4–2
"Version" string constant parameter to GET_INFO • 7–208
Video attribute
marker • 2–9, 7–261
PROMPT_AREA • 7–446
range • 2–22
SET (VIDEO) built-in procedure • 7–492
with STATUS_LINE • 7–476
VIDEO keyword • 7–492
"Video" string constant parameter to GET_INFO • 7–187, 7–193, 7–226
Virtual address space
VAXTPU restriction concerning • 5–1
Visibility
fetching display value of record or window • 7–186, 7–222
of record
using display value to determine • 7–370
setting record • 7–448
"Visible" string constant parameter to GET_INFO • 7–226
"Visible_bottom" string constant parameter to GET_INFO • 7–226
"Visible_length" string constant parameter to GET_INFO • 7–202, 7–226
"Visible_top" string constant parameter to GET_INFO • 7–226
"Vk100" string constant parameter to GET_INFO • 7–202
"Vt100" string constant parameter to GET_INFO • 7–202
"Vt200" string constant parameter to GET_INFO • 7–202

"Vt300" string constant parameter to GET_INFO • 7–202

W

Widget
  callback_parameters • 7–209
  case sensitivity of name • 7–74
  creating • 7–72
  defining a class of • 7–105
  deleting • 7–108
  fetching callback routine for • 7–214
  fetching children of in VAXTPU • 7–210
  fetching class of in VAXTPU • 7–214
  fetching name of • 7–215
  finding out if managed in VAXTPU • 7–214
  getting information about • 7–216
  listing of • 4–5
  main window • 4–16
  managing • 7–258
  mapped status
    controlling in VAXTPU • 7–418
  membership in subclass
    finding out in VAXTPU • 7–214
  menu bar
    in VAXTPU • 4–16
  menu position of in VAXTPU • 7–210
  parent of
    fetching in VAXTPU • 7–215
  realizing in VAXTPU • 7–306
  resource
    fetching class and data type of in VAXTPU • 7–215
  scroll bar • 7–224, 7–462
  scroll bar slider • 7–224
  setting resource values of • 7–494
  title bar • 4–16
  unmanaging • 7–534
  using callback data structure in VAXTPU • 7–496
  widget_id • 7–209
Widget children
  managing • 7–258
  unmanaging • 7–534
WIDGET data type • 2–24 to 2–25
Widget resources
  data types of • 4–12
  specifying • 4–12
WIDGET_CALL_DATA parameter to SET built-in procedure • 7–496
WIDTH parameter to SET built-in procedure • 7–501
"Width" string constant parameter to GET_INFO • 7–202
Wildcard characters
  in file names • 5–20
Window
  adjusting size • 7–19
  attributes • 7–78
  bottom
    example of fetching • B–16 to B–19
  changing position • 7–20
  command
    in EVE • 4–16
  creating • 2–26
  current • 2–27, 7–77
  definition • 2–25
  deleting • 6–4, 7–108
  determining bottom of • 7–222
  determining boundaries and size of • 7–222
  determining last column of • 7–224
  determining leftmost column of • 7–222
  determining length of • 7–223
  determining top of • 7–225
  determining width of • 7–226
  dimensions • 2–25
  enlarging • 7–19
  fetching display value of • 7–222
  function of
    in VAXTPU compared with DECwindows • 4–16
  getting information • 2–29
  key map list
    example of fetching • B–19 to B–22
  length • 2–26
    example of fetching • B–16 to B–19
  making current • 6–2
  mapping • 2–27, 6–3
  message
    in EVE • 4–16
  reducing • 7–20
  removing • 2–28
  screen management • 6–2 to 6–4
  screen updates • 6–7
  scroll bar in • 7–224, 7–462
  scroll bar slider in • 7–224
  setting display value of • 7–370
  size
    with terminal display • 6–4
    with terminal emulator • 6–4
  top
    example of fetching • B–16 to B–19
  unmapping • 2–28

Window (Cont.)
unsupported terminals • 2–29
updating • 2–29
user
in EVE • 4–16
values • 2–27
width • 2–26
example of fetching • B–19 to B–22
window width • 6–4
WINDOW data type • 2–25 to 2–29
"Within_range" string constant parameter to GET_INFO • 7–187
Word separators • 7–146
/WRITE qualifier • 5–17
"Write" string constant parameter to GET_INFO • 7–178
WRITE_CLIPBOARD built-in procedure • 7–540
example of use • B–11 to B–13
WRITE_FILE built-in procedure • 7–543 to 7–545
WRITE_GLOBAL_SELECT built-in procedure • 7–546
example of use • B–31 to B–33

X

XOR operator • 3–7
X resource
fetching value of • 7–151

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 Text Processing Utility
Manual: Part II
AA–PBTNA–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

--- Do Not Tear - Fold Here and Tape ---

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

--- Do Not Tear - Fold Here ---

Cut Along Dotted Line

Reader's Comments

VAX Text Processing Utility
Manual: Part II
AA–PBTNA–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

--- Do Not Tear - Fold Here and Tape ---

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

--- Do Not Tear - Fold Here ---

Cut Along Dotted Line