VMS Version 5.3 New Features Manual

Order Number: AA–MG29B–TE

October 1989

This manual describes the new features of the VMS Version 5.3 operating system.

Revision/Update Information: This manual supersedes the *VMS Version 5.2 New Features Manual.*

Software Version: VMS Version 5.3

digital equipment corporation
maynard, massachusetts

October 1989

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

ZK5367

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 xiii

Chapter 1 Introduction

1.1 **Overview** 1–1

1.2 **DECwindows** 1–1

1.3 **Distributed Name Service** 1–2

1.4 **VAXTPU** 1–2

1.5 **EVE** 1–4

1.6 **Small Computer System Interface (SCSI)** 1–4

Chapter 2 DECwindows System Features

2.1 **Internationalization** 2–1

2.2 **Monitor Independence** 2–1

2.3 **Multiscreen Support** 2–2

2.4 **Additional Transport Support** 2–2

2.5 **MIT Release 3 Compliance** 2–2

Chapter 3 DECwindows User Information

3.1 **Icon Box** 3–1

3.2 **Session Manager** 3–2

3.3 **FileView** 3–2

3.3.1 Updating the File List 3–3

3.3.2 Keeping Track of Work in Progress 3–3

3.3.3 Working with Files 3–3

3.3.4 Accessing Files and Applications Quickly 3–4

3.3.5 Using Double-Click Commands 3–4

3.3.6 Saving a View 3–4
3.3.7 Changing the Look of Your File List 3–4
3.3.8 Adding Verbs and Building Pull-Down Menus .. 3–5
3.3.9 Customizing FileView 3–6

3.4 Desktop Applications 3–6
3.4.1 Bookreader 3–7
3.4.2 Calculator 3–7
3.4.3 Calendar 3–7
3.4.4 Cardfiler 3–8
3.4.5 Clock 3–8
3.4.6 CDA Viewer 3–8
3.4.7 DECterm 3–8
3.4.8 Mail 3–8
3.4.9 Notepad 3–9
3.4.10 Paint 3–9
3.4.11 Puzzle 3–9

Chapter 4 DCL Commands

4.1 CONVERT/DOCUMENT 4–2

4.2 CREATE/TERMINAL Command 4–6

4.3 SET DISPLAY Command 4–12

4.4 SHOW LICENSE Command 4–12

4.5 VIEW Command 4–14

Chapter 5 System Management Features

5.1 Extension of Lock Manager Limit 5–1

5.2 NCP Executor Command Changes 5–1
5.2.1 Parameter for SET/DEFINE EXECUTOR 5–2
5.2.2 SHOW EXECUTOR CHARACTERISTICS Command 5–2

Chapter 6 Programming Features

6.1 XUI Toolkit New Features 6–1
6.1.1 New Toolkit Routines 6–2
6.1.2 New Attributes 6–5

6.2 New Xlib Routines 6–6

6.3 UIL Compiler Features 6–7
6.3.1 Language Data-Type Functions 6–7
6.3.2 Multiple Callback Procedures Feature 6–8
6.3.3 UIL Constraint Argument 6–8
6.3.4 UIL Version Qualifier 6–8

6.4 CDA Features 6–9
6.4.1 DTIF Support 6–9
6.4.2 CDA Converter Architecture 6–9
6.4.3 CDA Converters 6–9
6.4.4 CDA$CONVERT Routine 6–10
6.4.5 CDA Viewers 6–10
6.4.6 CDA Viewer Routines 6–10

6.5 DECwindows Server $QIO Calls to Driver 6–11

6.6 New DECwindows Driver Support 6–12

6.7 VMS SCSI Third-Party Device Support 6–12

6.8 $QIO Return for Network Name/Object Number ... 6–13

Chapter 7 The VMS Distributed Name Service

7.1 Introduction to the Distributed Name Service 7–2
7.1.1 The DNS Namespace 7–2
7.1.2 Structure of a Namespace 7–4
7.1.3 Creating Objects 7–9
7.1.4 Modifying Objects 7–12
7.1.5 Distributing the Namespace 7–14
7.1.6 Requesting Information from DNS 7–18

7.2 DNS System Services 7–26
$DNS 7–27
$DNSW 7–54

7.3 DNS Run-Time Routines 7–55
DNS$APPEND_SIMPLENAME_TO_RIGHT 7–56
DNS$COMPARE_FULLNAME 7–58
DNS$COMPARE_SIMPLENAME 7–60
DNS$CONCATENATE_NAME 7–62
DNS$COUNT_SIMPLENAMES 7–64
DNS$CVT_DNSADDRESS_TO_BINARY 7–66
DNS$CVT_DNSADDRESS_TO_NODENAME 7–68
DNS$CVT_NODENAME_TO_DNSADDRESS 7–70
DNS$CVT_TO_USERNAME_STRING 7–72

DNS$PARSE_USERNAME_STRING 7–74
DNS$REMOVE_FIRST_SET_VALUE 7–77
DNS$REMOVE_LEFT_SIMPLENAME 7–80
DNS$REMOVE_RIGHT_SIMPLENAME 7–82

7.4 Starting the DNS Clerk 7–84

7.5 DECnet Event Messages 7–84

7.6 System Error Messages 7–85

Chapter 8 VAXTPU

8.1 Associated Documents 8–3

8.2 Changed DECwindows VAXTPU Initialization 8–3

8.3 Initialization Coding 8–4
8.3.1 Enhancements to the MAP Built-In 8–4
8.3.2 Enhancements to the UNMAP Built-In 8–5
8.3.3 Behavior of GET_INFO (widget_variable, "widget_info") Built-In 8–5

8.4 Buffer Change Journaling 8–6
8.4.1 Buffer Change Journal File Naming Algorithm 8–8
8.4.2 Enhancements to the CREATE_BUFFER Built-In 8–9
8.4.3 Enhancements to the DELETE Built-In 8–10
8.4.4 GET_INFO (buffer_variable, "journaling") Built-In 8–10
8.4.5 GET_INFO (buffer_variable, "journal_file") Built-In 8–11
8.4.6 GET_INFO (buffer_variable, "safe_for_journaling") Built-In 8–11
8.4.7 GET_INFO (buffer_variable, "journal_name") Built-In 8–12
8.4.8 GET_INFO (string_variable, "journal") Built-In 8–12
8.4.9 RECOVER_BUFFER Built-In 8–13
8.4.10 SET (JOURNALING) Built-In 8–16
8.4.11 WRITE_FILE Built-In 8–17
8.4.12 TPU$_FILE_RECOVERABLE Item Code 8–17

8.5 Enhancements to VAXTPU's Pattern Support 8–18
8.5.1 New Pattern Keywords 8–18
8.5.2 Search Performance 8–18
8.5.3 The New Reverse Search Algorithm 8–19

8.6 Record Attributes 8–20
8.6.1 Display Value Attribute 8–21
8.6.2 Modifiability Attribute 8–22
8.6.3 New Built-Ins Implementing Record Attribute Support 8–22

8.7 Enhanced Support for DECwindows VAXTPU 8–30
8.7.1 Support for the Watch Cursor 8–30
8.7.2 Support for Resizing Windows and Screens 8–30
8.7.3 Support for Icons 8–32
8.7.4 Support for Sending and Detecting Client Messages 8–35
8.7.5 Other New Built-Ins Extending DECwindows VAXTPU 8–39

8.8 Support for Setting and Fetching the Default Directory 8–49
8.8.1 SET (DEFAULT_DIRECTORY)Built-in 8–49
8.8.2 GET_INFO (SYSTEM, "default_directory") Built-In 8–50
8.8.3 Callable Interface Issues 8–51

8.9 Enhancements to the VAXTPU Compiler 8–52
8.9.1 EQUIVALENCE Statement 8–52
8.9.2 Support for Local Variables in Unbound Code .. 8–53
8.9.3 Support for Conditional Compilation 8–54
8.9.4 Support for Specifying the Radix of Numeric Constants 8–55

8.10 Reserved Keywords 8–56

8.11 Support for Handling Detached Cursor Conditions 8–57
8.11.1 SET (DETACHED_ACTION) Built-In 8–57
8.11.2 GET_INFO (SCREEN, "detached_action") Built-In 8–58
8.11.3 GET_INFO (SCREEN, "detached_reason") Built-In 8–59

8.12 Other Enhanced Built-Ins 8–60
8.12.1 CHANGE_CASE Built-In 8–60
8.12.2 CREATE_RANGE Built-In 8–62

8.12.3 EDIT Built-In 8–63
8.12.4 GET_INFO (buffer_variable) Built-In 8–65
8.12.5 LENGTH Built-In 8–66
8.12.6 MESSAGE Built-In 8–66
8.12.7 MODIFY_RANGE Built-In 8–67
8.12.8 POSITION Built-In 8–68
8.12.9 SUBSTR Built-In 8–68
8.12.10 TRANSLATE Built-In 8–69

8.13 TPU$_FILEIO Item Code 8–71

8.14 TPU$_CHAIN Item Code 8–72

8.15 Enhancements to Keyboard Support 8–72

8.16 Enhancements to the /NODISPLAY Command Qualifier 8–73

Chapter 9 EVE

9.1 Input File Handling 9–2

9.2 Journaling and Recovery 9–3
9.2.1 Buffer Change Journaling 9–3
9.2.2 Keystroke Journaling and Recovery 9–7

9.3 Attribute Saving 9–8
9.3.1 Saving Attributes in a Section File 9–11
9.3.2 Saving Attributes in a Command File 9–13
9.3.3 Saving EVE Default Attributes 9–14

9.4 Menu Entries 9–15

9.5 Case-Exact Search 9–16

9.6 Key Definitions 9–16
9.6.1 DECwindows-Style Function Keys 9–17
9.6.2 Repeat Counts with Gold Key 9–18
9.6.3 WPS Ruler Keys 9–19
9.6.4 Mouse Buttons 9–19
9.6.5 LEARN Sequences and Prompts 9–20

9.7 FILL and Paragraph Boundaries 9–20

9.8 Buffer List 9–20

9.9 Batch Editing 9–21

9.10 Other Changes 9–22
9.10.1 Bound Cursor Motion 9–22

9.10.2 Commands Buffer 9–23
9.10.3 SHOW Command 9–23
9.10.4 RESTORE SELECTION Command with DECwindows Quick Copy 9–24
9.10.5 PREVIOUS BUFFER Command 9–24
9.10.6 Help Topics 9–24

9.11 Program-Level Changes 9–25
9.11.1 Renamed Variable for the MAIN Buffer 9–25
9.11.2 Pre-Key and Post-Key Procedure Sharing 9–25
9.11.3 Detached Cursor Handling 9–26
9.11.4 Status Line Processing 9–26
9.11.5 EVE$INTERNATIONALIZATION Module 9–26
9.11.6 Obsolete Keywords for Message Constants 9–26

Index

Examples

9–1 EVE-Generated Code for Saving Attributes in a Command File 9–13
9–2 EVE Initialization File for Batch Editing 9–22

Figures

7–1 A DNS Namespace 7–5
7–2 Valid Character Codes for DNS Simple Names 7–8
7–3 Additional Character Codes Allowed in Quoted Simple Names 7–8
7–4 A Partitioned Namespace 7–15
7–5 A Namespace with Replicated Directories 7–16

Tables

3–1 FileView Menu Changes 3–2
6–1 New Routines 6–2
6–2 New Attributes 6–5
6–3 New Xlib Routines 6–7
6–4 New UIL Data-Type Functions 6–7
7–1 DNS Item Code Arguments 7–47

8–1	Journaling Behavior Established by EVE	8–8
8–2	Selected Built-In Actions When ERASE_UNMODIFIABLE is Turned Off	8–26
8–3	Detached Cursor Flag Constants	8–60
8–4	CREATE_RANGE Keyword Parameters	8–63
8–5	New GET_INFO Calls for the Editing Point and Their Previous Equivalents	8–65
8–6	MODIFY_RANGE Keyword Parameters	8–68
9–1	EVE Commands for Buffer Change Journaling and Recovery	9–4
9–2	EVE Commands for Saving Default Attributes	9–8
9–3	EVE Commands for Saving Attributes	9–9
9–4	EVE Settings for Saving Attributes	9–12

Preface

Intended Audience

This book is intended for general users, system managers, and programmers who use the VMS operating system.

Structure of This Document

This book is organized as follows:

- Chapter 1 provides an overview of new features developed since VMS Version 5.2.
- Chapter 2 describes the significant DECwindows development and features that affect system performance and usage.
- Chapter 3 summarizes the new DECwindows user features.
- Chapter 4 presents new DCL commands as well as additions made to existing commands, and describes how they are used.
- Chapter 5 summarizes the new features that support system management.
- Chapter 6 summarizes the new programming features. Most of the features are in the DECwindows programming languages.
- Chapter 7 introduces the Distributed Name Service and describes how to program and use the facility.
- Chapter 8 presents the new VAXTPU features and provides the application programmer with information about using these features.
- Chapter 9 presents enhancements made to the EVE editor.

Associated Documents

The *VMS DECwindows Programming Documentation Supplement* contains new DECwindows programming language information. The following books contain additional information about the new features of the VMS Version 5.3 operating system:

- *VMS Version 5.3 Release Notes*
- *CDA Reference Manual*
- *VMS DECwindows Desktop Applications Guide*
- *VMS DECwindows User's Guide*
- *VMS DECwindows Programming Documentation Supplement*
- *VMS DECwindows Transport Manual*
- *VMS Version 5.3 Small Computer System Interface (SCSI) Device Support Manual*

Conventions

The following conventions are used in this manual:

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.)
Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 x	A sequence such as PF1 x indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
Return	A key name is shown enclosed to indicate that you press a key on the keyboard.

. . .	In examples, a horizontal ellipsis indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In format descriptions, brackets indicate that whatever is enclosed is optional; you can select none, one, or all of the choices. Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of substring specification in an assignment statement.
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
red ink	Red ink characters appearing in interactive examples of this book indicate information or commands that you must enter from the keyboard. Characters and output lines in black ink are system prompts and display output. For online versions, user input is shown in **bold**.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason.
italic text	Italic text represents information that can vary in system messages (for example, Internal error *number*).
UPPERCASE TEXT	Uppercase letters indicate that you must enter a command (for example, enter OPEN/READ), or they indicate the name of a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.
-	Hyphens in coding examples indicate that additional arguments to the request are provided on the line that follows.
numbers	Unless otherwise noted, all numbers in the text are assumed to be decimal. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

Chapter 1

Introduction

This chapter provides an overview of the VMS Version 5.3 new features.

1.1 Overview

The VMS Version 5.3 operating system provides new features in the following major areas:

- Improved DECwindows software for workstations
- Distributed Name Service (DNS) for network based systems
- Enhanced VAX Text Processing Utility (TPU) and Extensible VAX Editor (EVE)
- Small Computer System Interface (SCSI) support

1.2 DECwindows

Many of the VMS Version 5.3 new features reflect enhancements to the DECwindows workstation software. DECwindows improvement is noted in the following areas:

- Improved system features and performance

  The improvements to DECwindows system features provide a high level of interoperability, monitor independence, multiscreen support, and compliance with Massachusetts Institute of Technology (MIT) Version 11, X Window standard.

  In addition, DECwindows has extended international features to allow users who have installed support for a language other than English to switch easily to that language.

  The TCP/IP transport introduces support for the the VMS/ULTRIX Connection X layered product. Also, transport layer programming support is now provided to third-party programmers.

See Chapter 2 for more information about these features.

- Expanded user features

  The Window Manager (Icon Box) and Session Manager have been enhanced to make it easier for users to customize their DECwindows sessions.

  FileView now includes additional commands for working with files and provides additional customization features and options. In addition, the desktop applications have been enhanced.

  See Chapter 3 for more details about these user features.

- Expanded programming features

  The X User Interface (XUI) Toolkit, Xlib library, User Interface Lanaguage (UIL), Compound Document Architecture (CDA), and server programming have been expanded. New routines, attributes, arguments, and data-type structures have been added to the DECwindows programming languages. See Chapter 6 for more information about using these program features.

- New Digital Command Language (DCL) commands and additions to exisiting commands have been added to provide more DECwindows support. See Chapter 4 for more information about using these new DCL commands and qualifiers.

1.3 Distributed Name Service

The new VMS Distributed Name Service (DNS) provides for storing the names of resources such as files, disks, nodes, queues, and mailboxes on your network. For the programmer, the DNS clerk provides a means of assigning unique names to resources so that an application or user can easily find those resources. Once an application has named a resource using DNS, the name is available for all users of the application.

If the application requires that the resource be known throughout the network, the DNS Server layered product must be installed on the remote systems. Once the server is installed, multiple users located throughout a network can refer to a common resource by the same name. Resources can be moved within the network. No additional preparation is required, and it is unnecessary to learn a new naming convention. See Chapter 7 for more information about using DNS.

1.4 VAXTPU

VAXTPU is a programmable, text processing utility that aids application and system programmers in developing tools that manipulate text. The following new features have been added to VAXTPU:

- Buffer change journaling

In addition to keystroke journaling, VAXTPU now performs buffer change journaling. File modifications are tracked on a per-buffer basis. Buffer change journaling allows DECwindows VAXTPU sessions to be journaled and recovered. To implement buffer change journaling, VAXTPU also provides six new built-in procedures and enhances four previously existing built-ins.

- DECwindows VAXTPU improvements

  Initialization is faster for applications layered on DECwindows VAXTPU.

  The MAP and UNMAP built-ins now accept a widget parameter. Icon pixmaps are now supported and seven new built-ins are available for finer control over various DECwindows functions.

  Window resizing is easier with a new SET (HEIGHT) and the enhanced SET (WIDTH) built-in procedures.

- Programming tools improved

  Pattern searches are more efficient. New keywords are available for creating patterns. The SCAN, SCANL, SPAN, and SPANL built-in procedures can perform a new kind of reverse search.

  VAXTPU now gives programmers the option to make one or more records in a buffer visible or invisible on a screen and to set or change the left margin of the records. Also, the records can be made unmodifiable, so a user cannot alter the record.

  Function keys F1 through F5 are now supported. Modifiers ALT_MODIFIED and HELP_MODIFIED for main keyboard keys and for control modified keys are also available. In addition, all built-ins can be used when the /NODISPLAY command qualifier is specified.

  VAXTPU supports binary, octal, and hexadecimal constants as well as decimal constants.

  Application programmers or users can now change the current default directory from within VAXTPU and can declare a local variable in code that is not within a procedure. For defining synonyms an EQUIVALENCE statement is now available. Conditional compilation statements enable the programmer to control which portions in a source file are compiled under various conditions.

  You can now designate a program to handle a detached cursor condition (condition in which the cursor position cannot accurately represent the editing point in the current window).

  When using the VAXTPU callable interface, application programmers can allow VAXTPU to supply a default routine instead of specifying the entry TPU$_FILEIO in the item list for the TPU$INITIALIZE routine. The VAXTPU callable interface now supports chaining of item lists, making the VAXTPU callable interface more like VMS system services.

See Chapter 8 for more information about these features.

1.5 EVE

The Extensible VAX Editor (EVE) is an editor written in the VAXTPU programming language. The following new features have been added to EVE:

- Buffer change journaling and recovery

  In addition to keystroke journaling, EVE now performs buffer change journaling. EVE creates a journal file for each user buffer. Recovery is done interactively (that is, by using RECOVER commands within EVE) and restores only your text, not any settings or key definitions. This method of journaling and recovery works on DECwindows as well as on character-cell terminals. You can still use keystroke journaling and recovery when using EVE on a character cell terminal.

- Attribute saving

  You can now save global attribute settings in a section file or TPU command file, either by using the SAVE ATTRIBUTES command or when exiting from EVE.

- Menu customization

  You can now add and delete menu items in DECwindows EVE's pull-down and pop-up menus.

- Search list handling

  If you use a search list or wildcard directory specification (such as [...]) in specifying an input file, EVE gets the first matching file in the search list.

- New commands

  There are several new commands, particularly concerning buffer change journaling and recovery, user attribute saving, and menu customizations.

- Batch editing

  You can now invoke EVE with EDIT/TPU/NODISPLAY, typically for batch editing.

See Chapter 9 for more information about these features.

1.6 Small Computer System Interface (SCSI)

VMS Version 5.3 supports the attachment of third-party Small Computer System Interface (SCSI) peripherals to the SCSI bus of MicroVAX and VAXstation 3100 models or 3520/3540 systems. For more details, see Chapter 6.

Chapter 2

DECwindows System Features

The major new VMS Version 5.3 DECwindows system and performance features are:

- More internationalization support
- Monitor independence
- Multiscreen support
- Additional transport support
- MIT X Window System Release 3 compliance

2.1 Internationalization

The previous release of DECwindows contained numerous international features, including support for 8-bit character sets, European keyboard support, and compound string interfaces in the toolkit programming library.

With VMS Version 5.3, DECwindows extends this support at both the user interface and programming library levels. It adds the ability to switch between multiple natural languages (for example, French or German), if the DECwindows language variant is installed on a system. A new compound string text widget has been added, and numerous enhancements have been made to provide better directional support (for example, right-to-left) for international markets.

2.2 Monitor Independence

The new DECwindows applications have fewer restrictions about the type of workstation monitors and screen sizes on which they can be effectively displayed. All applications can run on monitors with 75 dots-per-inch (dpi) or 100-dpi screens. They also handle screen geometries as small as 512 by 342 pixels using the appropriate fonts and color palettes.

2.3 Multiscreen Support

Significant development and changes in the X server, device drivers, and Session Manager components now support the multiscreen environment. A multiscreen system is a workstation with one keyboard, one mouse, and more than one display screen. It is still a single-user workstation system allowing a specified keyboard or mouse to interact with other monitors of a workstation group. All VMS Version 5.3 DECwindows clients can be run on screens other than screen zero.

2.4 Additional Transport Support

DECwindows now adds TCP/IP transport support for use with the VMS/ULTRIX Connection (UCX) layered product. The programming interface between the DECwindows common-transport module and the transport-specific module is now documented in the *VMS DECwindows Transport Manual.* This enables other Digital groups and third parties to write their own transport layers to extend further the interoperability of DECwindows systems.

2.5 MIT Release 3 Compliance

All of the DECwindows components now meet the published interfaces specified by the Massachusetts Institute of Technology (MIT) X Window System, Version 11, Release 3.

Chapter 3

DECwindows User Information

This chapter describes the latest DECwindows features that are of interest to all VMS DECwindows users. The main sections cover new features for the following topics:

- Window Manager (Icon Box)
- Session Manager
- FileView
- Desktop applications

For more details about using the Icon Box, Session Manager, and FileView, see the *VMS DECwindows User's Guide*. For information about using DECwindows desktop applications, see the *VMS DECwindows Desktop Applications Guide*.

3.1 Icon Box

The DECwindows Icon Box contains the following new features:

- If you press and hold MB2 in the Icon Box's work area, the Window Manager displays a pop-up menu, which contains Save Settings and Customize options.
- When you click on a window for input focus, the text "Press MB2 to customize" replaces the copyright notice in the title bar. When you click again for input focus, the text "Icon Box" replaces "Press MB2 to customize."
- If you point to the title bar of any window and press and hold MB3, you can move the window without setting input focus to that window.

3.2 Session Manager

New Session Manager features allow you to do the following:

- Customize the Applications Menu
- Customize the definitions that start applications
- Select applications to start automatically when you log in
- Customize the default language
- Disable the Print Screen Rotate and Scale Pictures option
- Use color Print Screen options
- Specify a transport when using the Security option
- Customize the Pause text label
- Specify another window manager as executable
- Display a logo other than the Digital logo in the login box

3.3 FileView

Table 3–1 summarizes changes made to FileView menus.

Table 3–1: FileView Menu Changes

Menu Item	Change
Layout...	Moved from Control menu to Customize menu.
Menu Bar...	Moved from Control menu to Customize menu. Allows you to remove or rearrange all the menus in your menu bar. (Previously, the Control, Customize, and Views menus could not be removed or rearranged.)
Privileges...	Moved from Control menu to Customize menu.
Logical Names	Moved from Control menu to Utilities menu.
DDIF	Renamed (on Applications menu) to CDA Viewer.
Read	New command in the Files menu that has the same options as the Edit command, but allows you only to read, not edit, a file.
Type	Removed from the Files menu as well as from all FileView menus. For upward compatibility, the Type command is still defined in the FileView system customization file, but is not referenced. (The Read command includes a Type option as one of the reader choices.)

(continued on next page)

Table 3–1 (Cont.): FileView Menu Changes

Menu Item	Change
Set Protection	New command in the Files menu that lets you change the file protection on selected files by setting and clearing the toggle buttons in the Set Protection dialog box. You can also specify the Log and Confirm options.
Any Verb...	New item on Control menu.

The following sections summarize additional FileView changes and new features.

3.3.1 Updating the File List

FileView now allows you to cancel a long-running view update operation. When you start an update operation, the Apply/Update button becomes a Cancel button. Clicking on the Cancel button aborts the file search, and FileView displays whatever file information was collected up to that point, along with a pop-up message box that indicates some information might be missing.

3.3.2 Keeping Track of Work in Progress

The behavior of the Stop Task buttons in the Work in Progress and Task Output windows have changed. Previously, when you clicked on the Stop Task button, FileView attempted to stop the running task and removed the Work in Progress box entry for the task whether the task actually stopped or not. Now, FileView attempts to stop the task and, rather than removing the entry, shows the status "Stopped" in the Work in Progress box. When the task actually stops, the Work in Progress box entry disappears.

FileView now uses a single Work in Progress dialog box for all FileView tasks, regardless of which FileView window you started them from. When you choose Work in Progress from the Control menu or start a task that causes the Work in Progress dialog box to appear, it is placed relative to the current FileView window. After that, it will reappear in the same position each time it is popped up.

3.3.3 Working with Files

The dialog boxes for commands on the Files menu now contain an Apply button. When you click on the Apply button, the task is started, but the dialog box is not dismissed. Instead, the OK, Apply, and Cancel buttons are disabled until the task completes. When the task completes, FileView reenables the buttons; you can then use the same dialog box to perform another task. For example, you can choose Search, and search repeatedly for different strings on the selected file list without choosing Search again.

3.3.4 Accessing Files and Applications Quickly

When FileView's file list does not entirely fill its window, there is unused blank space after the last file name. In previous versions, you could not double-click or display pop-up menus in this background area. Now you can define a double-click verb and pop-up menu for this unused space by using the new file type **.

3.3.5 Using Double-Click Commands

In FileView's file list, a double click can mean different things depending on the type of file you point at. When you double click on a file, FileView now displays a tiny pop-up menu that shows the verb about to execute.

3.3.6 Saving a View

The Save View dialog box contains five additional view components:

- Window Name
- Icon Name
- Initial State
- Exclude Directories
- Exclude Files

The initial settings in the Save View dialog box have changed. When you choose Save View... from the Customize menu, the dialog box pops up with all its toggle buttons set and with the Name of View field containing Startup (the name of the startup view).

The Save View dialog box contains All and None buttons, which set and clear all the toggle buttons at once. By default, Save View... now behaves like Save Startup View. (In VMS Version 5.1, only the File Filter and Directory toggle buttons were set by default; and the name field was empty.)

3.3.7 Changing the Look of Your File List

The following new features let you change the look of your file list:

- FileView now allows you to exclude VMS directory (DIR) files from your file list. In VMS Version 5.1, DIR files were always shown in the file list, even though they were also visible in the subdirectory (navigation) list box.

  You can control this option from the Layout dialog box (choose Layout... from the Customize menu). You can also save views that will enable or disable the setting by using the Save View dialog box.

  DIR files are now excluded from the file list by default.

- By choosing the new menu item Exclude Files... from the Customize menu, you can specify a file name filter (with wildcards) to remove some files from the file list. This is similar in functionality to the DCL command DIRECTORY/EXCLUDE. You can save these file filters in saved views.

  If the current view has a nonblank exclude file filter, a status line (Excluded: *nn*) appears in FileView's main window. This shows the number of files removed from the file list because of the exclude file filter.

  The Views menu now includes a new saved view (Exclude: None) item. Choosing this saved view clears the current view's exclude file filter.

- The Layout dialog box contains another toggle button (Sort/Filter Key) in the Fields list. By default, the toggle button is enabled and the sort key field (if any) is automatically displayed. When this toggle button is disabled, FileView does not automatically display the sort key field. In VMS Version 5.1, the sort key values were always shown in the file list if you enabled a sorted order for your view. For example, if you sorted your file list by file creation date, there was no way to avoid having the create date displayed.

3.3.8 Adding Verbs and Building Pull-Down Menus

FileView's built-in verbs can now be used in your own custom menus or as double-click verbs. You can also rename any of FileView's built-in verbs.

You can now change the contents of the Control and Customize menus just as you can change any other menu in FileView.

Three built-in verbs have been added (Select All, Select None, and Update View) that correspond to the buttons in FileView's main window. By default, these verbs are on the background pop-up menu.

The command files invoked by verbs on the Files menu, as well as the DCL Command verb command file, now accept the Task Output box size as an optional parameter. For example, the size of DCL Command window can be changed to 48 by changing the verb definition to:

```
@VUE$LIBRARY:VUE$DCL_COMMAND 48
```

A new command file called VUE$ITERATE is included in VUE$LIBRARY. This command file allows you to define verbs that operate on each of the currently selected files without writing a command file. The command file takes a parameter that is the DCL command to execute for each file. The command definition should include the symbol 'VUE$FILE in the position at which the file name is to be substituted. For example, a Show Owner verb can be defined as follows:

```
@VUE$LIBRARY:VUE$ITERATE "DIR/OWNER 'VUE$FILE"
```

The command file allows additional optional control parameters. Parameter 2 is the label to be used in the Task Output box. If no control parameters are specified, Processing is used. The current file name is appended to this label. Parameter 3 is the prompt string to be used to prompt for file names if there are no currently selected files. If parameter 3 is null or not specified, Files: is used. If parameter 4 is present, the Work in Progress box will *not* automatically pop up when you choose the verb. If parameter 4 is null or not specified, the box appears. Parameter 5 is the size of the task output box. The default is 24 lines. The built-in verb Show Files is defined using this command file as follows:

```
@VUE$LIBRARY:VUE$ITERATE "DIR/FULL/NOHEAD/VERSION=1 -
 'VUE$FILE" "Showing "Show File" false
```

3.3.9 Customizing FileView

A new verb, Create Public Profile, makes it easier for application developers to create the FileView verb definition file they need to ship with their products. This verb is not on the menu bar by default, but it can be accessed through the Any Verb... dialog box.

You can now customize these three components of your FileView windows:

- The window name, as displayed in FileView's main window title bar.
- The icon name, as displayed in FileView's icon in the Icon Box.
- FileView's initial state. This allows you to start FileView as a window or an icon.

You can enter labels for the window and icon names, and you can indicate whether the DECnet node name and the current default directory should also be displayed. The window and icon names can be different, and they can be stored as part of a saved view. By default, the DECnet node name and the default directory are included in both the window and icon names.

3.4 Desktop Applications

This section summarizes the new features of the DECwindows Desktop Applications. The following applications have added new features. See the *VMS DECwindows Desktop Applications Guide* for more information.

- Bookreader
- Calculator
- Calendar
- Cardfiler
- Clock
- CDA Viewer

- DECterm
- Mail
- Notepad
- Paint
- Puzzle

3.4.1 Bookreader

The Bookreader now includes:

- Context-sensitive online Help.
- Go Back button in the topic window, which returns the display to the last topic being viewed.
- Menu items on the View menu that let you enable or disable highlighting of hot spots and extensions. (For example, text that describes an extension to a programming language.)
- More button in the topic window, lets you move forward within a topic. (If the end of the topic is reached, the next topic is displayed.)
- Icons to the left of each title in the selection window to show whether the title refers to a shelf or a book.

3.4.2 Calculator

The Calculator now includes the keys deg, X!, 1/X, Rnd, Inv, sin, cos, tan, log, ln, e, and Y^X and allows you to save the screen placement and size of the Calculator if you modified those settings during your session.

3.4.3 Calendar

The following changes and additions have been made to the Calendar:

- Timeslots in the day display now contain a "handle" that enables you to move an entry to a different timeslot.
- You can create daynotes, which are notes that are not associated with a particular time. You can also create daynotes that repeat at regular intervals.
- You can associate icons with entries and daynotes.
- You can create overlapping entries.
- Calendar now allows you much greater range and flexibility in customizing alarm settings.
- You can send and receive Calendar entries through mail.

- You can save the size and position of the Calendar if you modified those settings during your session.
- You can specify a smaller increment by which to move an entry using the handle—for example, you could specify an increment of four minutes, which would enable you to make an entry for an 8:04 meeting.

3.4.4 Cardfiler

The Cardfiler now allows you to cut and paste images from other applications, such as Paint. You can also save the size and placement of the Cardfiler if you modified those settings during your session.

3.4.5 Clock

The Clock now lets you use 24-hour time and to save the size and placement of the Clock if you modified those settings during your session.

3.4.6 CDA Viewer

The Compound Document Architecture (CDA) Viewer (formerly the DDIF Viewer) has two new menu items on the File menu: Diagnostic Information and Document Information, which you can choose to display information about the file you are currently viewing.

The new Options File dialog box enables you to specify the name of a file containing processing options to be applied as your file is loaded for viewing. The new Paper Size dialog box enables you to pick a paper size and to override the existing document format.

There is also a new Page... button at the bottom of the window, which allows you to specify the number of the page you want to display.

3.4.7 DECterm

DECterm now lets you disable operating system control over terminal size and to adjust the transcript size. There is a new Customize Graphics dialog box that lets you modify DECterm graphics features. In addition, you can now specify your own titles for the icon and window.

3.4.8 Mail

The following changes and additions have been made to Mail:

- You can move messages in the outline interface by dragging them.
- You can send an existing file without using an editor by choosing the Use Existing File... menu item on the Create-Send window's File menu.
- You can specify your own display name for new drawers or modify the display name for existing drawers.

- Mail now has keyboard accelerator support for quick ways of performing common functions.
- The Customize menu has the following new menu items: User Attributes..., Send Attributes..., Deliver Attributes..., Print Attributes..., Window Attributes..., Save Attributes, Restore Attributes, and Restore System Attributes.
- The Send button (and the Send menu item on the File Menu) is dimmed after sending a message. It remains dimmed until you change something in the Send window.
- You can specify whether Mail starts in a window or an icon state.

3.4.9 Notepad

The Notepad now lets you split the screen so that you can view and edit two portions of a file at the same time. In addition, there are two new menu items: Select All and Clear, which let you select all or delete all of the text in a buffer. You delete text by choosing Select All and then Clear.

3.4.10 Paint

Paint now has a Picture Size... menu item on the Customize menu, which lets you change the picture size and modify the resolution. The new Full View menu item on the Options menu enables you to display entire figures even if they are too large to fit in the Paint window, and to display or permanently crop only a portion of your picture. The new Scale Picture menu item on the Edit menu lets you scale your entire image.

Paint also now has keyboard accelerators that let you perform common Paint operations without using the mouse.

3.4.11 Puzzle

If you modified the size and placement of the puzzle during a session, you can now save these values.

Chapter 4

DCL Commands

VMS Version 5.3 includes the following new DCL command:

Command	Function
CREATE/TERMINAL	Creates a window, such as a DECterm window, that emulates another terminal type

The following table lists the new qualifiers to the DCL commands SET DISPLAY, SHOW LICENSE, and VIEW:

Qualifier	Function
SET DISPLAY Command	
/DELETE	Deletes the display device and cancels the redirected display
/EXECUTIVE_MODE	Creates an executive mode device
/SUPERVISOR_MODE	Creates a supervisor mode device
/USER_MODE	Creates a user mode device
SHOW LICENSE Command	
/BRIEF	Displays a summary of information about the specified product licenses
/CHARGE_TABLE	Displays information in the current Charge Table for the current VAX computer
/PRODUCER=name	Displays software product licenses active on the current node

Qualifier	Function
VIEW Command	
/INTERFACE	Specifies the type of display device you are using
/HEIGHT	Specifies the height of the page in number of characters
/OVERRIDE_FORMAT	Selects the CDA converter to override your document format
/WIDTH	Specifies the number of characters per line

In addition, the CONVERT/DOCUMENT command now supports the DTIF input and output format that is included in the /FORMAT and /OPTIONS qualifiers. Also, the /OPTIONS qualifier contains new processing options.

4.1 CONVERT/DOCUMENT

The DCL command CONVERT/DOCUMENT converts a revisable format file to another revisable or final format file. You can use this command only if you have DECwindows installed on your system.

The CONVERT/DOCUMENT command has the following format:

CONVERT/DOCUMENT *input-file[/FORMAT=format-name]* -
output-file[/FORMAT=format-name]

Parameter

input-file
Specifies the file to be converted. The default file type is DDIF.

output-file
Specifies the name of the converted file. The default file type is DDIF.

Qualifiers

/FORMAT=format-name
Specifies the encoding format of the input or output file. The default format is DDIF for both input and output.

The DTIF format is now supported with the /FORMAT qualifier. Input formats bundled with the VMS operating system and their default file types are as follows:

Input Format	File Type
DDIF	DDIF
DTIF	DTIF
TEXT	TXT

Output formats bundled with the VMS operating system and their default file types are as follows:

Output Format	File Type
DDIF	DDIF
DTIF	DTIF
TEXT	TXT
PS	PS
ANALYSIS	CDA$ANALYSIS

Digital's CDA Converter Library is a layered product that provides additional input and output formats. Independent software vendors who write DDIF- and DTIF-conforming applications, as well as front- and back-end converters, also provide input and output formats that are layered on the VMS operating system. Contact your system manager for a complete list of input and output formats available on your system.

/OPTIONS=options-filename

Specifies a file that contains processing options for both input and output. The default file type for a VMS options file is CDA$OPTIONS.

You can create an options file that contains all the input and output processing options to be applied during the conversion of the input file to the output file. These processing options affect how your input file is processed and how your output file is created or displayed.

The CDA Converter contains several new processing options available for several of the file formats that are bundled with VMS. The new processing options available with VMS Version 5.3 follow.

Text Back-End Processing Options

The text back-end converter supports the following new options:

- HEIGHT *value*

  Specifies the maximum number of lines per page in your text output file. If you specify 0, the number of lines per page corresponds to the height specified in your document. If you additionally specify OVERRIDE_FORMAT or if the document has no inherent page size, the document is formatted to the height value specified by this option. The default height is 66 lines.

- OVERRIDE_FORMAT [ON,OFF]

  Causes the text back-end converter to ignore the document formatting information included in your document, so that the text is formatted in a single large galley per page that corresponds to the size of the page as specified by the HEIGHT and WIDTH processing options. If this option is not specified, the default is OFF; if this option is specified without a value, the default is ON.

- WIDTH *value*

  Specifies the maximum number of columns of characters per page in your text output file. If you specify 0, the number of columns per page corresponds to the width specified in your document. If you additionally specify OVERRIDE_FORMAT or if the document has no inherent page size, the document is formatted to the value specified by this processing option. If any lines of text exceed this width value, the additional columns are truncated. The default width is 80 characters.

Domain Conversion Processing Options

The new CDA Domain converter provides several new domain conversion processing options. When you are converting any table format to any document format, you can now specify the following processing options using a format name of DTIF_TO_DDIF:

- COLUMN_TITLE

  Displays the column titles as contained in the column attributes centered at the top of the column.

- CURRENT_DATE

  Displays the current date and time in the bottom left corner of the page. The value is formatted according to the document's specification for a default date and time.

- DOCUMENT_DATE

  Displays the document date and time as contained in the document header in the top left corner of the page. The value is formatted according to the document's specification for a default date and time.

- DOCUMENT_TITLE

  Displays the document titles as contained in the document header centered at the top of the page, one string per line.

- PAGE_NUMBER

  Displays the current page number in the top right corner of the page.

- PAPER_SIZE *size*

  Specifies the size of the paper to be used when formatting the PostScript output file. The values are the same as those for the PostScript back end.

- PAPER_HEIGHT *height*

  Specifies a paper size other than one of the predefined values provided. The default paper height is 11 inches.

- PAPER_WIDTH *width*

  Specifies a paper size other than one of the predefined sizes provided. The default paper width is 8.5 inches.

- PAPER_TOP_MARGIN *top-margin*

  Specifies the width of the margin provided at the top of the page. The default value is 0.25 inches.

- PAPER_BOTTOM_MARGIN *bottom-margin*

  Specifies the width of the margin provided at the bottom of the page. The default value is 0.25 inches.

- PAPER_LEFT_MARGIN *left-margin*

  Specifies the width of the margin provided on the left-hand side of the page. The default value is 0.25 inches.

- PAPER_RIGHT_MARGIN *right-margin*

  Specifies the width of the margin provided on the right-hand side of the page. The default value is 0.25 inches.

- PAPER_ORIENTATION *orientation*

  Specifies the paper orientation to be used in the output file. The values are the same as those for the PostScript back end.

Example

```
$ CONVERT/DOCUMENT/OPTIONS=OPTIONS.CDA$OPTIONS -
_$ FOOBAR.DTIF/FORMAT=DTIF MOOMAR.DDIF/FORMAT=DDIF
```

This command converts an input file named FOOBAR.DTIF, which has the DTIF format, to an output file named MOOMAR.DDIF, which has the DDIF format. The specified options file is named OPTIONS.CDA$OPTIONS.

4.2 CREATE/TERMINAL Command

The new DCL command CREATE/TERMINAL creates a window that emulates another terminal type. Currently, only DECterm windows can be created with this command.

Specify CREATE/TERMINAL in the following format:

CREATE/TERMINAL *[command-string]*

Parameter

command-string

Specifies a command string that is to be executed in the context of the created subprocess. This parameter allows CREATE/TERMINAL to be used in much the same way as the SPAWN command.

Description

The CREATE/TERMINAL command creates a subprocess of your current process. When the subprocess is created, the process-permanent open files and any image or procedure context are *not* copied from the parent process. The subprocess is set to command level 0 (DCL level with the current prompt).

If you do not specify the /PROCESS qualifier, the name of this subprocess is composed of the same base name as the parent process and a unique number. For example, if the parent process name is SMITH, the subprocess name can be SMITH_1, SMITH_2, and so on.

The LOGIN.COM file of the parent process is not executed for the subprocess, because the context is copied separately, allowing quicker initialization of the subprocess. When the /WAIT qualifier is in effect, the parent process remains in hibernation until the subprocess terminates and returns control to the parent by using the ATTACH command.

Use the LOGOUT command to terminate the subprocess and return to the parent process. You can also use the ATTACH command to transfer control of the terminal to another process in the subprocess tree, including the parent process. (The SHOW PROCESS/SUBPROCESS command displays the process in the subprocess tree and points to the current process.)

> **NOTE:** Because a tree of subprocesses can be established using the CREATE/TERMINAL command, you must be careful when terminating any process in the tree. When a process is terminated, all the subprocesses below that point in the tree are automatically terminated.

Qualifiers for the CREATE/TERMINAL command must directly follow the command verb. The command string parameter begins after the last qualifier and continues to the end of the command line.

Qualifiers

/APPLICATION_KEYPAD
Sets the APPLICATION_KEYPAD terminal characteristic in the created terminal window. If /APPLICATION_KEYPAD or /NUMERIC_KEYPAD is not specified, the default is to inherit the characteristic from the parent. (See also /NUMERIC_KEYPAD.)

/BIG_FONT
Specifies that the Big Font (as specified in resource files) be selected when the created terminal window is initialized. It is an error to specify /BIG_FONT in combination with /LITTLE_FONT. If you do not specify either /BIG_FONT or /LITTLE_FONT, the initial font is /BIG_FONT.

/BROADCAST
/NOBROADCAST
Determines whether the terminal window is created with broadcast messages enabled. If neither qualifier is specified, the created terminal window inherits the broadcast characteristic of the parent window.

/CARRIAGE_CONTROL
/NOCARRIAGE_CONTROL
Determines whether carriage return and line feed characters are prefixed to the subprocess's prompt string. By default, CREATE/TERMINAL copies the current setting of the parent process. This qualifier is used only with /NODETACH.

/CLI=cli-filespec
/NOCLI
Specifies the name of a Command Language Interpreter (CLI) to be used by the subprocess. The default CLI is the same as that of the parent process (defined in SYSUAF). If you specify /CLI, the attributes of the parent process are copied to the subprocess. The CLI you specify must be located in SYS$SYSTEM and have the file type EXE. This qualifier is used only with /NODETACH.

/CONTROLLER=filename
Specifies the name of the terminal window controller image. This name allows the CREATE/TERMINAL command to create a window on a variant controller, such as for a language not supported by the base product. For a DECterm, the default is SYS$SYSTEM:DECW$TERMINAL.EXE. The device and directory default to SYS$SYSTEM and the file type defaults to EXE.

> **NOTE:** The "name" field of the file name as returned by $PARSE is used to form the mailbox logical name. For example, if the file "name" is DECW$TERMINAL, the mailbox logical name will be DECW$TERMINAL_MAILBOX_node::0.0. For backward compatibility, the controller also defines a logical name DECW$DECTERM_MAILBOX_host::0.0 to point to the same mailbox.

/DEFINE_LOGICAL=({logname, TABLE=tablename} [,...])
Specifies one or more logical names that are set to the name of the created pseudo-terminal device. Each element in the list is either a logical name or else TABLE= followed by the name of a logical name table in which all subsequent logical names will be entered. The default is the process logical name table.

/DETACH
/NODETACH (default)
Determines whether the created terminal process is detached or a subprocess of the current process. The command-string parameter can not be used with the /DETACH qualifier.

/DISPLAY=display-name
Specifies the name of the display on which to create the terminal window. If this parameter is omitted the DECW$DISPLAY logical name is used.

/ESCAPE
/NOESCAPE
Sets or clears the ESCAPE characteristic of the created terminal window. The default is to inherit the characteristic of the parent.

/FALLBACK
/NOFALLBACK
Sets or clears the FALLBACK characteristic of the created terminal window. The default is to inherit the characteristic of the parent.

/HOSTSYNC (default)
/NOHOSTSYNC
Sets or clears the HOSTSYNC characteristic of the created terminal window. The default is to inherit the characteristic of the parent.

/INPUT=filename
Specifies an alternate input file or device to use as SYS$INPUT for the new process. The default is to use the created terminal window for input.

/INSERT
Creates the terminal window with insert mode as the default for line editing. If /INSERT or /OVERSTRIKE is not specified, the default is to inherit the characteristic from the parent. (See also /OVERSTRIKE.)

/KEYPAD (default)
/NOKEYPAD
Determines whether keypad definitions and the current keypad state are copied from the parent process. This qualifier is used only with /NODETACH.

/LINE_EDITING
/NOLINE_EDITING

Determines whether the terminal window is created with line editing enabled. If neither qualifier is specified, the created terminal window inherits the line editing characteristic of the parent.

/LITTLE_FONT

Specifies that the Little Font (as specified in resource files) be selected when the created terminal window is initialized. It is an error to specify /LITTLE_FONT in combination with /BIG_FONT. If you do not specify either /BIG_FONT or /LITTLE_FONT, the initial font is /BIG_FONT.

/LOGGED_IN (default)
/NOLOGGED_IN

Determines whether a prompt for a user name and password are supplied (/NOLOGGED_IN) or the created terminal window is logged in automatically (/LOGGED_IN). This qualifier is used only with /DETACH.

/LOGICAL_NAMES (default)
/NOLOGICAL_NAMES

Determines whether the created terminal window inherits the parent's logical names. This qualifier is used only with /NODETACH.

/NOTIFY
/NONOTIFY (default)

Determines whether a notification message is broadcast to the parent when the created terminal window exits. This qualifier is used only with /NODETACH.

/NUMERIC_KEYPAD

Sets the NUMERIC_KEYPAD terminal characteristic in the created terminal window. If /NUMERIC_KEYPAD or /APPLICATION_KEYPAD is not specified, the default is to inherit the characteristic from the parent. (See also APPLICATION_KEYPAD.)

/OVERSTRIKE

Creates the terminal window with overstrike mode as the default for line editing. If /OVERSTRIKE or /INSERT is not specified, the default is to inherit the characteristic from the parent. (See also /INSERT.)

/PASTHRU
/NOPASTHRU

Sets or clears the PASTHRU characteristic in the created terminal window. The default is to inherit the characteristic of the parent.

/PROCESS (default)
/PROCESS=process-name
/NOPROCESS
Specifies the name of the process or subprocess to be created. /NOPROCESS means that a window is created without a process. If you specify /PROCESS without a process name, a unique process name is assigned with the same base name as the parent process and a unique number. The default process name format is: username_n. If you specify a process name that already exists, an error message is displayed. This qualifier is used with both /DETACH and /NODETACH.

/PROMPT=prompt
Specifies the prompt string of the created terminal window. This qualifier is used only with /NODETACH.

/READSYNC
/NOREADSYNC
Sets or clears the READSYNC terminal characteristic in the created terminal window. The default is to inherit the characteristic from the parent.

/RESOURCE_FILE=filename
Specifies that the created terminal window use the resource file "filename" instead of the default resource file, DECW$USER_FAULTS:DECW$TERMINAL_DEFAULT.DAT.

/SYMBOLS (default)
/NOSYMBOLS
Determines whether the subprocess inherits the parent's DCL symbols. This qualifier is used only with /NODETACH.

/TABLE=command-table
Specifies the name of an alternate command table to be used by the subprocess. This qualifier is used only with /NODETACH.

/TTSYNC
/NOTTSYNC
Sets or clears the TTSYNC terminal characteristic in the created terminal window; the default is to inherit the characteristic of the parent.

/TYPE_AHEAD
/NOTYPE_AHEAD
Sets or clears the TYPE_AHEAD terminal characteristic in the created terminal window. The default is to inherit the characteristic of the parent.

/WAIT
/NOWAIT (default)
Requires that you wait for the subprocess to terminate before you enter another DCL command. The /NOWAIT qualifier allows you to enter new commands while the subprocess is running. This qualifier is used only with /NODETACH.

/WINDOW_ATTRIBUTES=(keyword [,...])
Specifies initial attributes for the created terminal window to override the defaults read from the resource file. These keywords include:

Keyword	Description
BACKGROUND	The background color.
FOREGROUND	The foreground color.
WIDTH	The width, in pixels.
HEIGHT	The height, in pixels.
X_POSITION	The x-position, in pixels.
Y_POSITION	The y-position, in pixels.
ROWS	The number of rows in the window, in character cells. If the Auto Resize Window option is enabled, ROWS and COLUMNS override the size specified by WIDTH and HEIGHT.
COLUMNS	The number of columns in the window, in character cells. If the Auto Resize Window option is enabled, ROWS and COLUMNS override the size specified by WIDTH and HEIGHT.
INITIAL_STATE	The initial state of the window, either ICON or WINDOW.
TITLE	A character string specifying the window title.
ICON_NAME	A character string specifying the window icon name.
FONT	The name of the font to be used in the window. If you specify /LITTLE_FONT, or omit both /LITTLE_FONT and /BIG_FONT, this overrides the name of the Little Font that is set in the resource files; otherwise it overrides the name of the Big Font. The font name can be a logical name, and it can be (but does not have to be) the base font in a complete font set.

Example

```
$ CREATE/TERMINAL=DECTERM/DISPLAY=MYNODE::0 -
_$ /WINDOW_ATTRIBUTES=(ROWS=36,COLUMNS=80, -
_$ TITLE="REMOTE TERMINAL",ICON_NAME="REMOTE TERMINAL")
```

This command creates a detached process in a DECterm window on node MYNODE:: that is 36 rows by 80 columns and has its title and icon name set to "Remote terminal".

4.3 SET DISPLAY Command

The DCL command SET DISPLAY has four new qualifiers that direct the output of a DECwindows application:

/DELETE
Deletes the display device and cancels the redirected display. The /DELETE qualifier replaces the /NOPERMANENT qualifier. The /NOPERMANENT qualifier continues to be supported.

/EXECUTIVE_MODE
Creates an executive mode device and assigns the logical name DECW$DISPLAY to point to it. This qualifier must be used with the /CREATE qualifier. Devices created with the /EXECUTIVE_MODE qualifier are deleted only if:

- They are explicitly deleted with the /DELETE qualifier.
- The system is rebooted.

To create, modify, or delete executive mode devices, you must have SYSMAN privilege.

/SUPERVISOR_MODE
Creates a supervisor mode device and assigns the logical name DECW$DISPLAY to point to it. This qualifier must be used with the /CREATE qualifier. When the user logs out, the device is deleted.

/USER_MODE
Creates a user-mode display device and assigns the logical name DECW$DISPLAY to point to it. This qualifier must be used with the /CREATE qualifier. The lifetime of a user-mode device is one DECwindows image: when the next DECwindows image exits, the device is deleted.

4.4 SHOW LICENSE Command

The DCL command SHOW LICENSE displays software product licenses active on the current node. The command contains new qualifiers that support a charge table and that display licenses according to a specified producer.

The SHOW LICENSE command has the following format:

SHOW LICENSE *[product-name]*

Parameter

product-name

Specifies the name or names of activated software product licenses to display. Wildcard characters (* and %) are allowed. If you do not specify a product name, all active product-name license information is displayed. The **product-name** parameter is incompatible with the /CHARGE_TABLE qualifier.

Qualifiers

/BRIEF

Displays a summary of information about the specified active product licenses. You cannot use the /BRIEF qualifier with the /CHARGE_TABLE qualifier.

/CHARGE_TABLE

Displays information in the current Charge Table, also known as the License Unit Requirement Table (LURT), for the current VAX computer. The /CHARGE_TABLE qualifier is incompatible with the **product-name** parameter, and the /BRIEF and /PRODUCER qualifiers.

/OUTPUT[=filespec]
/NOOUTPUT

By default, the output of the SHOW LICENSE command is sent to the current SYS$OUTPUT device (usually your terminal). To send the output to a file, use the /OUTPUT qualifier followed by a file specification.

You cannot use wildcard characters for the file specification. If you enter a partial file specification (for example, specifying only a directory), SHOW is the default file name and LIS is the default file type.

If you enter the /NOOUTPUT qualifier, output is suppressed.

/PRODUCER=producer-name

Displays software product licenses active on the current node that are supplied by a specified producer. Wildcard characters (* and %) are allowed for the **producer-name** parameter. You cannot use the /PRODUCER qualifier with the /CHARGE_TABLE qualifier.

Example

```
$ SHOW LICENSE

    Active licenses on node WTPOOH:

         DVNETEND
                 Producer: DEC
                 Units: 0
                 Version:  5.3
                 Date:  (none)
                 Termination Date: (none)
                 Availability: E (System Integrated Products)
                 Activity:  0
                 MOD_UNITS
```

```
VAX-VMS
        Producer: DEC
        Units: 0
        Version:  5.3
        Date:  (none)
        Termination Date: (none)
        Availability: A (VMS Capacity)
        Activity:  0
        MOD_UNITS
        NO_SHARE
```

The SHOW LICENSE command in this example displays all the active licenses on the current node named WTPOOH.

```
$ SHOW LICENSE/BRIEF

    Active licenses on node WTPOOH:

------- Product ID --------    ---- Rating ----- -- Version --
Product            Producer    Units Avail Activ Version Date      Expires
DVNETEND           DEC             0  E     0       5.3  (none)    (none)
VAX-VMS            DEC             0  A     0       5.3  (none)    (none)
```

The SHOW LICENSE command in this example displays a summary of all the active licenses on the current node named WTPOOH.

```
$ SHOW LICENSE/CHARGE_TABLE
VMS/LMF Charge Information for node WTPOOH
This is a VAX 8800, hardware model type 18
Type: A, Units Required: 93    (VMS Capacity)
Type: B, * Not Permitted *     (VMS Server)
Type: C, * Not Permitted *     (VMS Concurrent User)
Type: D, * Not Permitted *     (VMS Workstation)
Type: E, Units Required: 400   (System Integrated Products)
Type: F, Units Required: 1200  (Layered Products)
```

The SHOW LICENSE command in this example displays license unit requirements for the current VAX computer with the NODE name WTPOOH. For a description of how to use this information, see the *VMS License Management Utility Manual*, part of the VMS Base Documentation Set.

4.5 VIEW Command

The DCL command VIEW invokes the Compound Document Architecture (CDA) Viewer, which lets you view a compound document file on a character-cell terminal or a DECwindows display. Some of the qualifiers affecting the size of the display have no visible affect if you display a compound document file on a small monitor.

Four new qualifiers have been added to the VIEW command.

The VIEW command has the following format:

VIEW *input-file*

Parameter

input-file

Specifies the name of the file to be viewed. If you do not specify an input file name, you are prompted for one. You cannot use wildcard characters in the file name. The default input file-encoding format is DDIF, and the default file type is DDIF. Valid input file formats are any of those for which there is a CDA converter front end installed on the system.

Qualifiers

/FORMAT[=fmt-name]
/FORMAT=DDIF (default)

Specifies the format of your input file. The input formats that you can use with the CDA Viewer depend on the CDA converters installed on your system. The DTIF format is now supported. The default input format is DDIF. Input formats now bundled with the VMS operating system and their default file types are as follows:

Input Format	File Type
DDIF	DDIF
DTIF	DTIF
TEXT	TXT

Additional input formats are provided in Digital's CDA Converter Library, a layered product. Independent software vendors who write DDIF- and DTIF-conforming applications and front- and back-end CDA converters also provide input formats that are layered on the VMS operating system. Contact your system manager for a complete list of input formats available on your system.

/HEIGHT=nn

Specifies the height of the page in number of characters. If you specify the /OVERRIDE_FORMAT qualifier or if the document being viewed has no inherent format, this page height is used. On the DECwindows display, the default height is 66 lines, which is equivalent to the default page height of 11 inches. On character cell displays, the page height defaults to your terminal's screen height. However, if you use the /OUTPUT qualifier, the page height depends on the page height of your document.

/INTERFACE=DECWINDOWS
/INTERFACE=CHARACTER_CELL (default)

Specifies the type of display you are using.

/OVERRIDE_FORMAT
/NOOVERRIDE_FORMAT (default)

Selects the CDA Viewer to override the format of your document. The NOOVERRIDE option (default) uses the format information stored in your document.

/WIDTH=nn

Specifies the number of characters per line. If you specify the /OVERRIDE_ FORMAT qualifier or if the document being viewed has no inherent format, this page width is used. On the DECwindows display, the default width is 85 characters, which is equivalent to the default page width of 8.5 inches. On character cell displays, the page width defaults to your terminal's screen width. However, if you use the /OUTPUT qualifier, the default is 132 columns.

Example

```
$ VIEW FOOBAR.DTIF/FORMAT=DTIF/OPTIONS=OPTIONS.CDA$OPTIONS -
_$ /NOOUTPUT/NOPAGE/INTERFACE=DECWINDOWS/OVERRIDE_FORMAT -
_$ /WIDTH=80/HEIGHT=66
```

This command invokes the CDA Viewer to view a file named FOOBAR.DTIF, which has the DTIF format. The display interface is DECwindows, and the CDA Viewer will override the document's default format. The display width will be 80 characters, and the display height will be 66 lines.

Chapter 5

System Management Features

This chapter summarizes new features of VMS Version 5.3 that provide system management support. Additions to the following components are described:

- Lock Manager
- NCP Executor Commands

5.1 Extension of Lock Manager Limit

The Lock ID space for the Lock Manager is now extended from 65,535 to 262,144 locks. The SYSGEN parameters listed in the following table are increased to the values indicated:

SYSGEN Parameter	New Maximum Value
LOCKIDTBL	262,144
LOCKIDTBL_MAX	262,144
SRPCOUNT	270,336
SRPCOUNTV	270,336
IRPCOUNT	135,168
IRPCOUNTV	135,168

5.2 NCP Executor Command Changes

The NCP executor commands now include:

- A new parameter to SET/DEFINE EXECUTOR command
- New display characteristics for SHOW EXECUTOR CHARACTERISTICS command

5.2.1 Parameter for SET/DEFINE EXECUTOR

The network control ancillary program (NETACP) manages an index into a properly synchronized table in nonpaged-pool memory. System managers can modify the size of the table using the NCP command SET/DEFINE EXECUTOR with the following new parameter:

Parameter	Description
MAXIMUM DECLARED OBJECTS	Specifies the number of objects that processes can declare. To determine the current number of declared objects on your system, use the NCP SHOW KNOWN OBJECTS command. Each of the objects with a PID listed is one declared object. A single process can declare more than one object. Failure to provide a sufficient number of objects can result in the failure of network servers to be initialized. The default of 31 objects is sufficient for most configurations. The valid range is 8 to 16383. Note that dynamically setting the number lower has no effect.

5.2.2 SHOW EXECUTOR CHARACTERISTICS Command

The SHOW EXECUTOR CHARACTERISTICS command now displays information as shown in the following example. Note that a new entry Maximum Declared Objects is displayed and the Pipeline quota now shows 10000.

```
NCP> SHOW EXECUTOR CHARACTERISTICS

Node Volatile Characteristics as of 16-JUN-1990 10:48:27

Executor node = 2.11 (BOSTON)
```

```
Identification            = DECnet-VAX V5.3,  VMS V5.3
Management version        = V4.0.0
Incoming timer            = 45
Outgoing timer            = 45
Incoming Proxy            = Enabled
Outgoing Proxy            = Enabled
NSP version               = V4.1.0
Maximum links             = 128
Delay factor              = 80
Delay weight              = 5
Inactivity timer          = 60
Retransmit factor         = 10
Routing version           = V2.0.0
Type                      = routing IV
Routing timer             = 600
Broadcast routing timer   = 40
Maximum address           = 1023
Maximum circuits          = 16
Maximum cost              = 1022
Maximum hops              = 15
Maximum visits            = 63
Maximum area              = 63
Max broadcast nonrouters  = 64
Max broadcast routers     = 32
Maximum path splits       = 1
Area maximum cost         = 1022
Area maximum hops         = 30
Maximum buffers           = 100
Buffer size               = 576
Default access            = incoming and outgoing
Pipeline quota            = 10000
Alias incoming            = Enabled
Alias maximum links       = 32
Alias node                =  2.10 (CLUSTR)
Path split policy         = Normal
Maximum Declared Objects  = 31
```

Chapter 6

Programming Features

This chapter summarizes new features of VMS Version 5.3 that provide programming support. The main sections cover new features for the following topics:

- X User Interface (XUI) Toolkit
- VMS DECwindows X Library (Xlib)
- User Interface Language (UIL) Compiler
- Compound Document Architecture (CDA)
- DECwindows Server
- DECwindows Driver
- Small Computer System Interface (SCSI)

With the exception of SCSI, VMS Version 5.3 programming enhancements primarily support the DECwindows environment.

6.1 XUI Toolkit New Features

VMS Version 5.3 of the XUI Toolkit contains two new widgets, one new gadget, and several widget manipulation routines. In addition, there are new attributes supported for existing widgets. Reference information on the new routines and attributes and information on programming with these new features is documented in the *VMS DECwindows Programming Documentation Supplement*. Section 6.1.1 gives an overview of the new routines. Section 6.1.2 lists the new attributes for existing widgets.

6.1.1 New Toolkit Routines

The new XUI Toolkit widgets and gadgets are as follows:

- Color mixing widget—This widget lets you define colors for applications. The widget is a pop-up dialog box containing a default color display subwidget and a default color mixer subwidget. The color display subwidget shows the original color in one window and the new color as you modify it in another window. The color mixer subwidget uses the red, blue, green (RGB) color model. You can either move a slider to specify percentages of each color or enter RGB values for those colors. Applications can replace both the color display subwidget and the color mixer subwidget with custom components.
- Compound string text widget—This widget lets you enter text and edit existing text using various character sets and writing directions.
- Pull-down menu entry gadget—This gadget corresponds to the pull-down menu entry widget.

Table 6–1 lists the new routines supported by VMS Version 5.3 of the XUI Toolkit.

Table 6–1: New Routines

Routine	Function
Intrinsic Routines	
APPLICATION ADD ACTIONS	Declares an action table and registers it with the translation manager.
APPLICATION ADD CONVERTER	Registers a new resource converter.
APPLICATION ERROR	Calls the installed fatal error procedure and passes the message specified.
APPLICATION ERROR MESSAGE	Calls the high-level error handler and passes the information specified.
APPLICATION GET ERROR DATABASE	Obtains the error database.
APPLICATION GET ERROR DATABASE TEXT	Obtains the error database text for an error or a warning.
APPLICATION GET SELECTION TIMEOUT	Returns the current value of the intrinsics selection timeout interval.
APPLICATION SET ERROR HANDLER	Registers an error procedure to be called on a fatal error condition.

(continued on next page)

Table 6–1 (Cont.): New Routines

Routine	Function
Intrinsic Routines	
APPLICATION SET ERROR MESSAGE HANDLER	Registers a procedure called on a fatal error condition.
APPLICATION SET SELECTION TIMEOUT	Sets the intrinsics selection timeout.
APPLICATION SET WARNING HANDLER	Registers a procedure to be called on nonfatal error conditions.
APPLICATION SET WARNING MESSAGE HANDLER	Registers a procedure that is called on a nonfatal error condition.
APPLICATION WARNING	Calls the installed nonfatal error procedure.
APPLICATION WARNING MESSAGE	Calls the installed high-level warning handler.
GET CONSTRAINT RESOURCE LIST	Gets a list of constraint attributes.
Convenience Routines	
ACTIVATE WIDGET	Allows the application to simulate push button activation.
GET USER DATA	Returns the user data associated with the widget.
DRM Routines	
FETCH COLOR LITERAL	Fetches a named color literal from a UID file and converts it to a pixel value.
FETCH ICON LITERAL	Fetches an icon literal from a UID file.
FETCH LITERAL	Fetches the value of a literal in a UID file.
Compound String Routines	
STRING FREE CONTEXT	Frees a compound string context structure.
STRING INIT CONTEXT	Initializes the context needed by GET NEXT SEGMENT.

(continued on next page)

Table 6–1 (Cont.): New Routines

Routine	Function
Cut and Paste Routines	
CLIPBOARD REGISTER FORMAT	Registers the length of the data for formats not specified by the ICCCM conventions.
END COPY FROM CLIPBOARD	Indicates that the application has completed copying data from the clipboard and unlocks the clipboard.
START COPY FROM CLIPBOARD	Indicates that the application is ready to start copying data from the clipboard and locks the clipboard.
START COPY TO CLIPBOARD	Identical to BEGIN COPY TO CLIPBOARD, except that the timestamp of the event that triggered the copy is included as an argument.
High-Level Widget Routines	
COLOR MIX GET NEW COLOR	Returns the red, green, and blue color values to the color mix widget.
COLOR MIX SET NEW COLOR	Sets the red, green, and blue color values in the color mix widget.
CS TEXT	Creates a compound string text widget.
CS TEXT CLEAR SELECTION	Clears the global selection highlighted in the compound string text widget.
CS GET EDITABLE	Obtains the current permission state concerning whether the text in the compound string text widget can be edited by the user.
CS STRING GET MAX LENGTH	Obtains the current maximum allowable length of the text in the compound string text widget.
CS TEXT GET SELECTION	Retrieves the global selection, if any, currently highlighted in the compound string text widget.
CS TEXT GET STRING	Retrieves all the text from the compound text widget.
CS TEXT REPLACE	Replaces a portion of the current text in the compound string text widget or inserts some new text into the current text of the compound string text widget.

(continued on next page)

Table 6–1 (Cont.): New Routines

Routine	Function
High-Level Widget Routines	
CS TEXT SET EDITABLE	Sets the permission state that determines whether the text in the widget can be edited by the user.
CS TEXT SET MAX LENGTH	Sets the maximum allowable length of the text in the compound string text widget.
CS TEXT SET SELECTION	Makes specified text in the compound string text widget the current global selection and highlights it in the compound string text widget.
CS TEXT SET STRING	Sets the text in the compound string text widget.
Low-Level Widget Routines	
COLOR MIX CREATE	Creates a color mix widget, which is a specialized pop-up dialog box, containing a default color display subwidget and a default color mixer tool subwidget.
CS TEXT CREATE	Creates a compound string text widget.
Gadget Creation Routines	
PULL DOWN MENU ENTRY CREATE	Creates a pull-down menu entry gadget.

6.1.2 New Attributes

Several widget creation routines now support additional attributes. Complete descriptions of these attributes are in the *VMS DECwindows Programming Documentation Supplement*. Table 6–2 lists the new attributes for existing widgets.

Table 6–2: New Attributes

Widget	Attribute
Dialog box	auto_unrealize
File selection	file_to_extern_proc
	mask_to_extern_proc
	mask_to_intern_proc

(continued on next page)

Table 6–2 (Cont.): New Attributes

Widget	Attribute
Help	gototopic_label
	gobacktopic_label
	visittopic_label
	close_label
	helphelp_label
	helpontitle_label
	helptitle_label
	help_acknowledge_label
	help_on_help_title
	cache_help_library
	map_callback
Menu	change_vis_atts
	menu_extend_last_row
Message box	second_label
	label_alignment
	second_label_alignment
	icon_pixmap
Push button	insensitive_pixmap
Scroll bar	show_arrows
S text	user_data
Toggle button	insensitive_pixmap_on
	insensitive_pixmap_off

6.2 New Xlib Routines

The DECwindows implementation of Xlib is now equivalent to the Massachusetts Institute of Technology (MIT) Release 3 version. DECwindows now contains five new Xlib routines that make the Display and Visual structures opaque. Table 6–3 lists the routines and their functions. For more detail, see the *VMS DECwindows Programming Documentation Supplement*.

Table 6–3: New Xlib Routines

Routine	Function
DISPLAY KEYCODES	Returns the minimum and maximum number of keycodes supported by the specified display.
DISPLAY MOTION BUFFER SIZE	Returns the size of the motion buffer for the specified display.
MAX REQUEST SIZE	Returns the maximum request size, in 4-byte units, that the server allows.
RESOURCE MANAGER STRING	Returns a pointer to the resource manager string for the specified dsplay.
VISUAL ID FROM VISUAL	Returns the visual identifier for the specified visual type.

6.3 UIL Compiler Features

This section summarizes features of the new UIL compiler in VMS Version 5.3. Changes affect the syntax and usage of the language elements and module components of the compiler. A new compiler qualifier (/VERSION) accommodates version changes to the DECwindows software. For more detail, see the *VMS DECwindows Programming Documentation Supplement.*

6.3.1 Language Data-Type Functions

Table 6–4 describes new and revised data-type functions in the new UIL compiler.

Table 6–4: New UIL Data-Type Functions

Data Type	Function
ASCIZ_STRING_TABLE	An ASCIZ string table is an array of ASCIZ (null-terminated) string values separated by commas. The ASCIZ_STRING_ TABLE function allows you to pass more than one ASCIZ string as a callback tag value.
COMPOUND_STRING_TABLE	A compound string table is an array of compound strings. Objects requiring a list of string values, such as the **items** and **selected_items** arguments for the list box widget, use string table values.
INTEGER_TABLE	An integer table is an array of integer values separated by commas. The INTEGER_TABLE function allows you to pass more than one integer tag value per callback reason.

(continued on next page)

Table 6–4 (Cont.): New UIL Data-Type Functions

Data Type	Function
XBITMAPFILE	The XBITMAPFILE function is similar to the ICON function in that both functions describe a rectangular icon that is x pixels wide and y pixels high. However, XBITMAPFILE allows you to specify an external file containing the definition of an X bitmap, whereas all ICON function definitions are coded directly within the UIL module. The X bitmap file specified as the argument to the XBITMAPFILE function is read at application run time by DRM.

6.3.2 Multiple Callback Procedures Feature

The new UIL allows you to specify multiple callback procedures of a specific callback reason by defining the procedures as a type of list. Just as with any other list types, you can define a procedures list either in-line or in a separate list section that you reference by name.

If you define a reason more than once (for example, when the reason is defined both in a referenced procedures list and in the callbacks list for the object), all previous definitions are overridden by the latest definition.

6.3.3 UIL Constraint Argument

The XUI Toolkit and the X Toolkit (instrinsics) now directly support definitions for constraint arguments. A constraint argument is one that is passed directly to children of an object, beyond those arguments that are normally available. For example, an attached dialog box widget grants a set of constraint arguments to its children. These constraint arguments are used to control the position of the children in the attached dialog box.

6.3.4 UIL Version Qualifier

The UIL version (/VERSION) qualifier now provides upward compatibility between Versions 1.0 and 2.0 of the UIL compiler. Note that VMS Version 5.3 incorporates UIL Version 2.0. The /VERSION qualifier serves two basic functions. First, you can use the /VERSION qualifier as a method for identifying the correct version of the XUI Toolkit for which an application was written. Second, you can use the /VERSION qualifier to continue building interfaces that will run under Version 1.0 of the XUI Toolkit, while still being able to use the new UIL compiler features implemented for Version 2.0.

Allowable values for the /VERSION qualifier are V1 and V2. The default is /VERSION=V2.

6.4 CDA Features

The CDA components (Toolkit, Converters, and Viewers) are significantly enhanced. The Converters and Viewers now render DDIF documents much closer to their intended appearance. Many problems in the Toolkit have been corrected, enabling other applications to process DDIF documents better. The Toolkit also includes new support for DTIF (DIGITAL Table Interchange Format) and a callable interface to the DECwindows and character cell viewers. For more information, see the *CDA Reference Manual*, Part I and Part II.

6.4.1 DTIF Support

DTIF is the standard format for the storage and interchange of table data file formats, such as database information and spreadsheets. DTIF defines the logical structure and layout of a data table, the values and formulas contained within the table, and the presentation attributes to be used when printing or displaying the table.

DTIF aggregate structures are new additions to the CDA Toolkit and are used with the CDA Toolkit routines in the same way as the DDIF aggregate structures.

6.4.2 CDA Converter Architecture

The CDA converter architecture contains a new component called the *domain converter*. The CDA converter kernel invokes the domain converter during the conversion of a table input file format to a document output file format. The domain converter converts the in-memory DTIF representation of a table input file to the in-memory DDIF representation. The converter kernel then converts the DDIF in-memory representation to the document output format for printing or viewing.

The CDA converter architecture also supports the new DTIF_TO_DDIF format name and several new processing options that you can specify in an options file. You use the DTIF_TO_DDIF format name with the new processing options when converting table input file formats to document output file formats from the DCL command line. For a description of these new features, see Chapter 4.

6.4.3 CDA Converters

The CDA Converters that ship with VMS are the Text front and back end, the DDIF and DTIF front- and back-end converters, the PostScript back end, and the Analysis back-end converter. The Text back-end converter has been improved to format documents similar to the format the CDA character-cell Viewer produces.

The Text back-end converter has four new processing options:

Option	Description
HEIGHT *value*	Lets you specify the maximum number of lines per page in your text output file.
OVERRIDE_FORMAT	Causes the Text back-end converter to ignore the document formatting information included in your document.
SOFT_DIRECTIVES	Causes the document to obey the soft directives contained in the document when creating your text output file.
WIDTH *value*	Lets you specify the maximum number of columns of characters per page in your text output file.

6.4.4 CDA$CONVERT Routine

The **standard-item-list** parameter has four new values:

Value	Description
CDA$_INPUT_FRONT_END_DOMAIN	The address and length of a string that specifies the input document domain (either DDIF or DTIF).
CDA$_INPUT_POSITION_PROCEDURE	The address of a procedure that provides position information.
CDA$_OUTPUT_BACK_END_DOMAIN	The address and length of a string that specifies the output document domain (either DDIF or DTIF).
CDA$_OPTIONS_LINE	The address and length of a string that contains options to control processing.

6.4.5 CDA Viewers

The CDA Viewers have been rewritten to significantly improve their formatting capabilities. They process multi-column, multi-font text, native graphics, and images. They support color for text, graphics, and images. The DCL command for the Viewers (VIEW) has been enhanced to give you more control over document formatting. For more details, see Chapter 4.

6.4.6 CDA Viewer Routines

The Toolkit contains several new routines encoded in the portable ULTRIX C format, which you can use to write CDA Viewer applications for use on DECwindows workstations or on character cell terminals. The new DECwindows routines and character cell routines are summarized in the following table:

Routine	Description
Character Cell Routines	
CC DELETE PAGE	Deallocates the page display structure allocated by the routine CC GET PAGE.
CC END	Deallocates all internal structures that were allocated and does general cleanup required for CC viewer shutdown for the current file.
CC GET PAGE	Returns the next sequential formatted page from the CDA document.
CC INITIALIZE	Initializes the character cell CDA Viewer and returns a context block to the caller for use in subsequent character cell CDA Viewer routine calls.
DECwindows Routines	
BOTTOM DOCUMENT	Displays the last content in the file in the widget window.
CLOSE FILE	Closes the file currently being read by the CDA Viewer and clears the window.
DOCUMENT INFO	Returns information from the header aggregate of the document currently opened.
GOTO PAGE	Attempts to move the document to the specified page number.
NEXT PAGE	Displays the next page of a CDA document.
PREVIOUS PAGE	Displays the previous page (if one exists) of a CDA document.
REGISTER CLASS	Indicates that the CDA Viewer widget is registered with DRM.
TOP DOCUMENT	Displays the beginning content of the file in the widget window.
VIEWER	Creates a widget for viewing a CDA file.
VIEWER CREATE	Creates a widget for viewing a CDA file.
VIEWER FILE	Opens a file and begins to view the information content of the file, provided the file can be converted to in-memory DDIF.

6.5 DECwindows Server $QIO Calls to Driver

For DECwindows server programming with a workstation driver, there are three new calls in the common driver interface. The first two calls use the new $QIO function modifier (IO$K_DECW_CURSOR_BOUNDARIES) to set and sense cursor boundaries on a screen. The third call is Write X Event (function code IO$_DECW_WRITEV) that generates an input packet event without the use of hardware.

The GPB Wait $QIO call to an output driver now includes the function modifier (GB$K_LEGGS_WAIT_FOR_PKT) as an option to handle the packet-wait function when no more are free. The $QIO GPB Wait fuction is now renamed to Packet Wait, which now supports two function modifier options. For more details, refer to the *VMS DECwindows Programming Documentation Supplement* and the *VMS DECwindows Device Driver Manual.*

6.6 New DECwindows Driver Support

The new driver, GBBDriver, provides support as an output driver for the VAXstation 3520 and 3540 Low End Graphics Subsystem (LEGGS) color monitors. Existing output drivers GABDriver and GCBDriver also support the VAXstation 3100 models.

6.7 VMS SCSI Third-Party Device Support

VMS Version 5.3 provides the three mechanisms to allow a non-Digital-supplied Small Computer System Interface (SCSI) device to be attached to a MicroVAX or VAXstation system. The implementor of support for a non-Digital-supplied SCSI device can select the most appropriate method based on the capabilities of the device, the needs of its end users, and available programming resources. The three mechanisms provided by VMS Version 5.3 are as follows:

- A SCSI disk or tape drive can function properly using the standard VMS SCSI disk or tape class driver and the VMS SCSI port driver, given certain restrictions and cautions.
- An application program can send commands to, receive status from, and exchange data with a device on the SCSI bus by using the VMS generic SCSI class driver. The VMS operating system defines a special Queue-I/O Request ($QIO) system service interface that allows an application to pass SCSI command packets to the device through the generic SCSI class driver and the VMS SCSI port driver.
- A non-Digital-supplied SCSI class driver, in conjunction with the VMS SCSI port driver, can supply the level of support most closely tailored to the capabilities of the device. By writing a SCSI class driver, a system programmer can implement device-specific error handling and a simple, robust $QIO interface.

  Because the VMS operating system provides a special set of macros that initialize the SCSI port and transfer commands and data to a SCSI device, the programmer of a SCSI class driver can focus on coding details related to device capabilities. The VMS operating system further facilitates the writing of a SCSI class driver by including the online sources of a template SCSI class driver.

Refer to the *VMS Version 5.3 Small Computer System Interface (SCSI) Device Support Manual* for a complete description of VMS support of third-party SCSI devices.

6.8 $QIO Return for Network Name/Object Number

A new $QIO system service return condition (SS$_TOOMUCHDATA) is added to the status list concerning transparent task-to-task network operations. SS$_TOOMUCHDATA is a possible return condition when declaring a network name or object number. For more information about performing network task-to-task operations, see the *VMS Networking Manual.*

Status	Meaning
SS$_TOOMUCHDATA	The QIO has failed because the number of Maximum Declared Objects has been exceeded.

Chapter 7

The VMS Distributed Name Service

The Distributed Name Service (DNS) is a facility for storing the names of resources in your network such as files, disks, nodes, queues, and mailboxes. The Distributed Name Service clerk is the VMS programming interface to DNS that allows an application to register a resource in the name service and then access the resource from any point in the network by a single name. DNS is a layered product and must be installed in your network before you can start the DNS clerk or utilize the name service.

Applications that need the Distributed Name Service must use the $DNS clerk system service and the DNS run-time routines to register, modify, and locate information in the DNS database. A DNS clerk, which is resident on every VMS Version 5.3 system and later, receives application requests through the $DNS system service. The clerk locates a DNS server that can process the request. Once the request is satisfied, the clerk returns the requested information to the client application.

The information in this chapter is intended for VMS programmers who are writing applications that call the Distributed Name Service. It includes the following:

- Conceptual information on DNS
- DNS clerk system services, $DNS and $DNSW
- DNS run-time routines
- Startup information for the DNS clerk
- DECnet event messages from the DNS clerk
- System error messages generated by the DNS clerk

7.1 Introduction to the Distributed Name Service

The VAX Distributed Name Service (DNS) provides a means of assigning unique names to network resources so that a network application or network user can find resources within the network. (Resources are such things as disks, systems, applications, and so on.) Once an application has named a resource using DNS, the name is available for all users of the application. Multiple users located throughout a network can refer to a common resource by the same name. Resources can be moved within the network. No additional preparation is required, and it is unnecessary to learn a new naming convention.

You should consider using DNS applications that need to access such remote resources as printers, files, disks, and nodes. In addition, application databases or servers are good candidates for naming. All of these resources would be commonly named and their locations identified within DNS. With DNS, the resource could be moved without users being aware of the change.

Although it is desirable to name application databases, you should ordinarily use DNS to store only the location of the database, not the database itself. (Most database applications require higher levels of consistency than DNS provides.) If the database is relocated, then only the DNS information has to be modified.

7.1.1 The DNS Namespace

The collection of names in the Distributed Name Service database is called a **namespace**. A namespace is located on VMS nodes where the DNS server software is installed. The collection of databases stored on each server makes up the namespace itself.

DNS refers to the named resources in a namespace as **objects**. Each object name refers to a specific entity. The object name is important because applications use the object name in all DNS operations.

Associated with every object is a set of **attributes** describing properties of an object. An application reads object attributes for information such as an address, class, or version.

Most applications use the address attribute of an object, which allows you to find the node on which a resource resides. When a network resource is relocated, an application has DNS update the object's address attribute. All requests for the object receive the new address. Since the object has the same DNS name, the application user can be unaware that the resource has moved.

7.1.1.1 Planning Namespace Objects

When writing applications that use DNS, it is important to determine ahead of time what resources an application needs and how an application will use each resource. Then you can determine what objects an application needs to create and the kind of information each object needs to store. Once the object is designed, you can decide which object attributes to assign and what their values will be.

7.1.1.2 Restrictions

Because of the high cost of keeping copies of DNS names synchronized, you should use DNS applications that store information that does not change frequently. Frequent updates add traffic to the network, which can degrade overall network performance. Because resources such as files, disks, nodes, queues, and mailboxes remain on one node for a long time, a good example of information to store with DNS is a network address.

Not only should the information stored in DNS be relatively static, it should also be verifiable. When DNS updates its database, it attempts to send the update to all copies of the name within 24 hours. This means that your application can request data from a copy of a name that has not been updated. An application must be able to recognize when data is invalid. For this reason, a network address is a good example of data that can be validated. If you use an address and the resource is not there, the data is obviously outdated.

7.1.1.3 Using the Namespace

An application choosing to use the namespace performs four basic operations:

- Object creation— An application needs to create an object to represent each network resource it requires.
- Object modification— Once an object is created to represent a resource, an application modifies the object to contain the attributes and values the application requires.
- Object deletion— When a resource is no longer useful, an application should delete the object.
- Information retrieval. The most common operation an application performs is requesting the DNS clerk to obtain the values of an attribute so that, for example, the application can locate the resource in the network.

7.1.1.4 Object Names

The name DNS assigns to an object is one that the user supplies. The client application translates the name it receives through the user interface from string format into *opaque* format before passing it to the DNS clerk. DNS works only with opaque format because it is guaranteed to be unique, whereas string format often contains logical names that easily change.

The $DNS system service supplies functions for conversion between string and opaque format. If an application maintains its own databases, then the application must store DNS names in opaque format.

7.1.1.5 Object Attributes

Client applications store information about a resource as object *attributes*. When creating an object, an application needs to assign a class name and a version to a new object. The class name reflects the purpose of the object within an application. The purpose can be specific to an application or it can be shared among a group of applications. For example, a group of user names might be shared. An application uses the class name to search for its objects or list its objects. The class version helps to pair a version of an object with a software version.

In order to store additional information with an object, an application must modify the object.

DNS always assigns certain attributes to an object during creation. It assigns a unique identifier (UID) and an update time-stamping (UTS) indicating when an object was last edited. DNS also assigns a third attribute that specifies access control for the new object. Initially, the owner of the object has read, write, delete, control, and test access. The namespace administrator can modify this access according to site requirements.

An attribute name is limited to 31 characters and its value cannot exceed 4000 bytes. The name service assigns a prefix of DNS$ to the name of each attribute it assigns. An application creates a prefix to assign to attributes it creates. For example, DECnet uses the prefix DNA$ and the Distributed File Service uses the prefix DFS$. Names assigned by Digital all contain the dollar sign ($). User-supplied names should use an underscore (_). To ensure uniqueness, you should register your facility name through Digital's product registration program.

7.1.2 Structure of a Namespace

A DNS namespace is a hierarchical set of directories, as depicted in Figure 7–1. At the top of the hierarchy is the root directory, which is symbolized by a period (.). Below the root directory are levels of subdirectories. The namespace administrator establishes the directory structure of the namespace and, in some cases, assigns names to directories. While the organization of the namespace directories is similar to the VMS directory structure, namespace directories are completely separate from the VMS directory structure.

Directories in a namespace can contain three types of entries:

- Objects
- Directory pointers
- Soft links

An **object** represents a network resource. It consists of a name that is unique within the namespace and its associated attributes.

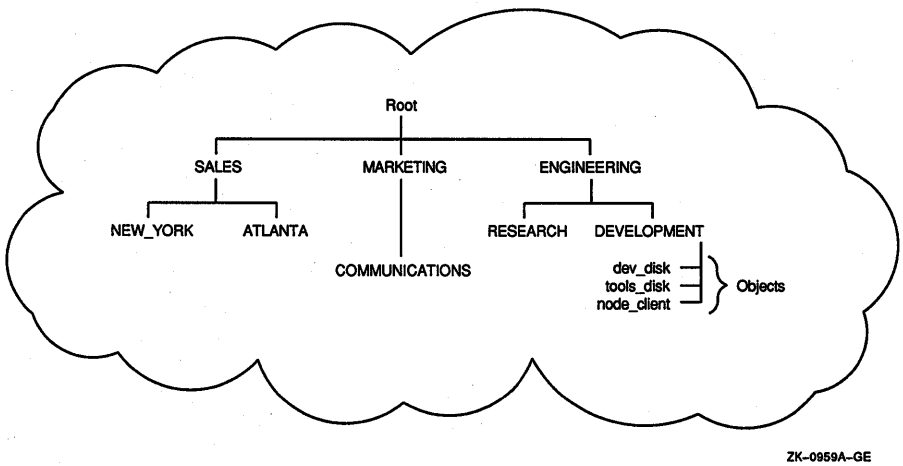
Directory pointers are used internally by DNS to link one level of directories to the next. DNS refers to the hierarchical relationship of directories in terms of *child directories* and *parent directories*.

A **soft link** provides an alternate name for an object, directory, or soft link. For example, a namespace structured with both an organizational and a geographical dimension might access a single object through multiple soft links. A soft link can also be useful in renaming an object. The soft link would point to the original object name so that users could successfully use an outdated name. This kind of soft link would be deleted after sufficient time has passed for applications and users to become aware of the new object name. You create and delete links through the DNS management program.

Although an application requests the creation of an object in order to register a resource, it does not position the object in the namespace. The system administrator determines which directory DNS stores the object in. The structure of a namespace differs for each network, so you should not hard-code names into applications.

Figure 7–1 illustrates a DNS namespace.

Figure 7–1: A DNS Namespace



7.1.2.1 Naming Syntax

The DNS name of an object, directory, or soft link in the namespace is a complete path specification from the root directory to the entity in the directory of interest. For example, the DNS name .ENGINEERING.DEVELOPMENT.TOOLS_DISK identifies an object named TOOLS_DISK in the namespace directory called

.ENGINEERING.DEVELOPMENT. The ENGINEERING directory is in the root directory, and DEVELOPMENT is a child directory of the ENGINEERING directory.

While the full name is a complete path name from the root directory, each element in a full name is called a **simple name**. The last simple name in a full name designates an object, a child directory, or a soft link. In the previous example, TOOLS_DISK is a simple name assigned to a disk object. The maximum length of a simple name is 255 bytes.

You can represent a full name in three ways:

namespacename:.simplename.simplename

or

.simplename.simplename

or

simplename.simplename

If the full name does not start with a namespace name or a period, DNS attempts to translate the first simple name as a logical name. Any equivalence name found is added to the name string in place of the matched simple name. This process is repeated until the first term does not match a logical name or the clerk encounters either a namespace name or a leading period. (A namespace name, assigned during DNS server installation, defaults to *node-name*_NS.)

The following example shows what happens with the name RESEARCH.PROJECT_ DISK:

1. Look up RESEARCH as a logical name.

   RESEARCH translates to ENG.RESEARCH, so the name string expands to ENG.RESEARCH.PROJECT_DISK.

2. Look up ENG as a logical name.

   ENG translates to .ENGINEERING, so the name string becomes .ENGINEERING.RESEARCH.PROJECT_DISK. Because the new name has a leading period, translation stops.

3. The namespace name, INMAX_NS, is added to the front of .ENGINEERING because it is not explicitly specified. (A namespace administrator establishes the namespace name during installation.) The name becomes INMAX_ NS:.ENGINEER- ING.RESEARCH.PROJECT_DISK.

7.1.2.2 Logical Names

When the DNS clerk is started on a VMS operating system (see Section 7.4), the VMS system creates a unique logical name table for DNS to use in translating full names. This logical name table, called DNS$SYSTEM, prevents unintended interaction with other system logical names. The DNS use of logical names in parsing full names is described in Section 7.1.2.1.

To define systemwide logical names for DNS objects, use the DCL command DEFINE. For example, to create the logical RESEARCH.PROJECT_DISK shown in the previous section, you would enter the following DCL command:

```
$ DEFINE/TABLE=DNS$SYSTEM RESEARCH "ENG.RESEARCH"
```

When parsing a name, the $DNS service specifies the logical name DNS$LOGICAL as the table it uses to translate a simple name into a full name. This name ordinarily translates to DNS$SYSTEM in order to access the systemwide DNS logical name table.

In order to define process or job logical names for $DNS, you must create a process or job table and redefine DNS$LOGICAL as a search list, as in the following example (note that elevated privileges are required to create a job table):

```
$ CREATE /NAME_TABLE DNS_PROCESS_TABLE
$ DEFINE /TABLE=LNM$PROCESS_DIRECTORY DNS$LOGICAL -
_$DNS_PROCESS_TABLE,DNS$SYSTEM
```

Once you have created the process or job table and redefined DNS$LOGICAL, you can create job-specific logical names for DNS using the DCL command DEFINE, as follows:

```
$ DEFINE /TABLE=DNS_PROCESS_TABLE RESEARCH "ENG.RESEARCH.MYGROUP"
```

For information about logical names, see *Introduction to VMS System Services*.

7.1.2.3 Valid Characters for DNS Names

DNS namespace names, full names, or simple names can contain letters, numbers, and certain punctuation marks from the ISO Latin 1 character set, as shown in Figure 7–2. Additional characters and punctuation marks can appear as long as the name is enclosed in quotation marks, for example, "project%". See Figure 7–3.

Figure 7–2: Valid Character Codes for DNS Simple Names

Code Range (Decimal)	Character
036	$
045	-
048–057	0 1 2 3 4 5 6 7 8 9
065–077	A B C D E F G H I J K L M
078–090	N O P Q R S T U V W X Y Z
095	_
097–109	a b c d e f g h i j k l m
110–122	n o p q r s t u v w x y z
192–207	À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
208–214	Ð Ñ Ò Ó Ô Õ Ö
216–223	Ø Ù Ú Û Ü Ý Þ ß
224–239	à á â ã ä å æ ç è é ê ë ì í î ï
240–246	ð ñ ò ó ô õ ö
248–255	ø ù ú û ü ý þ ÿ

ZK–0961A–GE

NOTE: All simple names containing the dollar sign ($) are reserved for use by Digital.

Figure 7–3: Additional Character Codes Allowed in Quoted Simple Names

Code Range (Decimal)	Character
032–033	{*space*} !
035	#
037–044	% & ' ( ) * + ,
046–047	. /
058–064	: ; < = > ? @
091–094	[\ ] ^
096	`
123–126	{ | } ~
160–167	{*no–break space*} ¡ ¢ £ ¤ ¥ ¦ §
168–174	¨ © ª « ¬ ®
175–187	¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º »
188–191	¼ ½ ¾ ¿
215	×
247	÷

ZK–0962A–GE

DNS maintains the case of an entity when it registers an object, but it is case insensitive in lookups. For example, the name eng.research would match the name ENG.RESEARCH.

DNS also supports binary simple names. A binary name consists of the leading character pair %x or %X, followed by pairs of hexadecimal digits. A binary simple name does not match any regular or quoted simple name, even if a given name has the same binary value.

DNS makes use of wildcards for identifying groups of objects during search operations. Wildcards consist of the following:

Symbol	Name	Meaning
?	Question mark	Match one character.
*	Asterisk	Match any number of characters.

7.1.3 Creating Objects

Each application that uses DNS must register its resources in the namespace using either the $DNS or the $DNSW system service. Registration involves creating an object in the namespace to represent the resource. You create an object to represent each resource in the network that your application needs to find. At the same time, you should define attributes the object needs and assign their values.

A DNS object consists of a name and its associated attributes. You create the object first, along with some key attributes. Later, you can modify the object to hold additional attributes that are relevant to the application.

To create an object with $DNS:

1. Prompt for a name from the user interface.

   The name that an application assigns to an object should come from a user interface, a configuration file, a system logical, or some other source. The application never assigns an object's name because the namespace structure is uncertain. The name the application receives from the user interface is in string format.

2. Use the $DNS parse function to convert the full name string into the opaque format of DNS.

3. Optionally, reserve an event flag so you can check for completion of the service.

4. Build an item list containing the following elements:
   - The opaque name for the object (resulting from the translation in step 2)
   - The class name given by the application, which should contain the facility code
   - The class version assigned by the application
   - An optional timeout value, specifying when the call expires
5. Optionally, provide the address of the DNS status block to receive status information from the name service.
6. Optionally, provide the address of the asynchronous system trap (AST) service routine. AST routines allow a program to continue execution while waiting for parts of the program to complete.
7. Optionally, supply a parameter to pass to the AST routine.
8. Call the create object function, providing all the parameters supplied in steps 1 through 7.

If a clerk call is not complete when timeout occurs, then the call completes with an error. The error is returned in the DNS status block.

An application should check errors returned; it is not enough to check the return of the $DNS call itself. You need to check the DNS status block to be sure there are no errors at the DNS server.

The following C routine shows how to create an object in the namespace with the synchronous service $DNSW. The routine demonstrates how to construct an item list.

```
#include <dnsdef.h>
#include <dnsmsg.h>
/*
 * Parameters:
 *      class_name = address of the opaque simple name of the class
 *                   to assign to the object
 *      class_len  = length (in bytes) of the class opaque simple name
 *      object_name= address of opaque full name of the object
 *                   to create in the namespace.
 *      object_len = length (in bytes) of the opaque full name of the
 *                   object to create
 */

create_object(class_name, class_len, object_name, object_len)
unsigned char *class_name;
unsigned short class_len;
unsigned char *object_name;
unsigned short object_len;
{
    struct $dnsitmdef createitem[4]; /* Item list used by system service */
    struct $dnscversdef version;     /* Version assigned to the object */
    struct $dnsb iosb;          /* Used to determine DNS server status */
    int status;                 /* Status return from system service */
```

```
    /*
     * Construct the item list that creates the object:
     */
    createitem[0].dns$w_itm_size = class_len;    ❶
    createitem[0].dns$w_itm_code = dns$_class;
    createitem[0].dns$a_itm_address = class_name;

    createitem[1].dns$w_itm_size = object_len;    ❷
    createitem[1].dns$w_itm_code = dns$_objectname;
    createitem[1].dns$a_itm_address = object_name;

    version.dns$b_c_major = 1;    ❸
    version.dns$b_c_minor = 0;

    createitem[2].dns$w_itm_size = sizeof(struct $dnscversdef);    ❹
    createitem[2].dns$w_itm_code = dns$_version;
    createitem[2].dns$a_itm_address = &version;

    *((int *)&createitem[3]) = 0;    ❺

    status = sys$dnsw(0, dns$_create_object, &createitem, &iosb, 0, 0);    ❻

    if(status == SS$_NORMAL)
    {
        status = iosb.dns$l_dnsb_status;    ❼
    }

    return(status);
}
```

❶ The first entry in the item list is the address of the opaque simple name representing the class of the object.

❷ The second entry in the item list is the address of the opaque full name for the object.

❸ The next step is to build a version structure, which will indicate the version of the object. In this case, the object is version 1.0.

❹ The third entry in the item list is the address of the version structure that was just built.

❺ Zero terminates an item list.

❻ Call the system service to create the object.

❼ Check to see that both the system service and DNS were able to perform the operation without error.

7.1.4 Modifying Objects

After applications use DNS to create objects that identify resources, they add attributes to the newly created objects that describe properties of the object.

You modify an object whenever you need to add an attribute, change an attribute value, or delete an attribute. You can add as many attributes as you like. If you add the same attribute to an object twice, the time-stamping on the attribute is updated.

DNS attributes can have a single value or they can have a set of values. For example, an attribute holding the class version number of a resource would have a single value, while an attribute holding the location of a service in the network could have a set of values. The set would hold the addresses of all nodes in the network that offer the service. Depending on the attribute type, DNS performs a slightly different action. DNS adds or deletes a value when there is only one. When there is a set of values, DNS adds or deletes a value from an existing group of values.

To modify an object with $DNS:

1. Build an item list containing the following elements:
   - The opaque name of the object you are modifying
   - The type of entry, as described in Section 7.1.2
   - The operation to perform
   - The type of attribute you are adding: a single value or a set of values
   - The attribute name
   - The value being added to the attribute
2. Supply any of the optional parameters described in Section 7.1.3.
3. Call the modify attribute function, supplying the parameters established in steps 1 and 2.

The following C example shows how to add an attribute and its value to an object:

```
#include <dnsdef.h>
#include <dnsmsg.h>
/*
 * Parameters:
 *      obj_name = address of opaque full name of object
 *      obj_len  = length of opaque full name of object
 *      att_name = address of opaque simple name of attribute to create
 *      att_len  = length of opaque simple name of attribute
 *      att_value= value to associate with the attribute
 *      val_len  = length of added value (in bytes)
 */
```

```
add_attribute(obj_name, obj_len, att_name, att_len, att_value, val_len)
unsigned char *obj_name;
unsigned short obj_len;
unsigned char *att_name;
unsigned short att_len;
unsigned char *att_value;
unsigned short val_len;
{
    struct $dnsitmdef moditem[7];          /* Item list for $DNSW */
    unsigned char objtype = dns$k_object;  /* Using object entries */
    unsigned char opertype = dns$k_present; /* Adding an object */
    unsigned char attype = dns$k_set;      /* Attribute will be type set */
    struct $dnsb iosb;                  /* Used to determine DNS status */
    int status;                         /* Status of system service */

    /*
     * Construct the item list to add an attribute to an object.
     */
    moditem[0].dns$w_itm_size = obj_len;
    moditem[0].dns$w_itm_code = dns$_entry;
    moditem[0].dns$a_itm_address = obj_name;    (1)

    moditem[1].dns$w_itm_size = sizeof(char);
    moditem[1].dns$w_itm_code = dns$_lookingfor;
    moditem[1].dns$a_itm_address = &objtype;    (2)

    moditem[2].dns$w_itm_size = sizeof(char);
    moditem[2].dns$w_itm_code = dns$_modoperation;
    moditem[2].dns$a_itm_address = &opertype;   (3)

    moditem[3].dns$w_itm_size = sizeof(char);
    moditem[3].dns$w_itm_code = dns$_attributetype;
    moditem[3].dns$a_itm_address = &attype;     (4)

    moditem[4].dns$w_itm_size = att_len;
    moditem[4].dns$w_itm_code = dns$_attributename;
    moditem[4].dns$a_itm_address = att_name;    (5)

    moditem[5].dns$w_itm_size = val_len;
    moditem[5].dns$w_itm_code = dns$_modvalue;
    moditem[5].dns$a_itm_address = att_value;   (6)

    *((int *)&moditem[6]) = 0;                  (7)

    /*
     * Call $DNSW to add the attribute to the object.
     */
    status = sys$dnsw(0, dns$_modify_attribute, &moditem, &iosb, 0, 0);

    if(status == SS$_NORMAL)
    {
        status = iosb.dns$l_dnsb_status;
    }

    return(status);
}
```

❶ The first entry in the item list is the address of the opaque full name of the object.

❷ The second entry in the item list shows that the entry is an object—not a soft link or directory pointer.

❸ The third entry in the item list is the operation to perform. The program adds an attribute with its value to the object.

❹ The fourth entry in the item list is the attribute type. The attribute has a set of values rather than a single value.

❺ The fifth entry in the item list is the opaque simple name of the attribute being added.

❻ The sixth entry in the item list is the value associated with the attribute.

❼ Check to see that both the system service and DNS performed the operation without error.

7.1.5 Distributing the Namespace

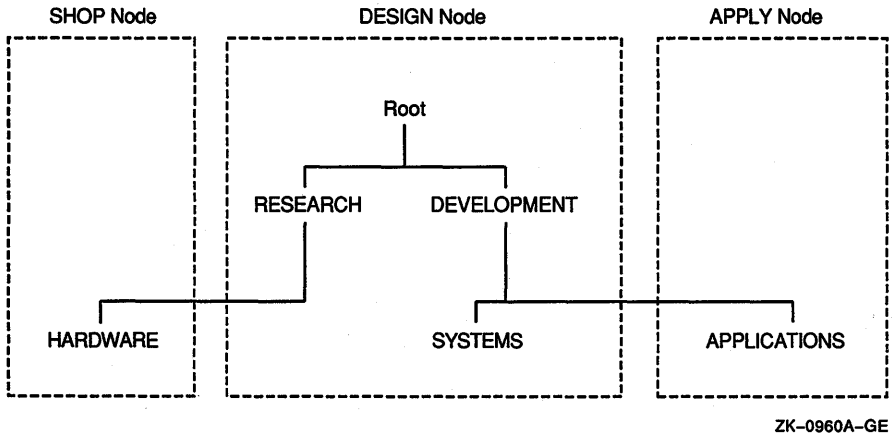
A VMS node running DNS server software can contain the entire namespace. However, performance and reliability are enhanced when several VMS nodes act as DNS servers.

DNS supports the *partitioning* of the namespace across several DNS servers. In this situation, no DNS server contains the entire namespace, but each contains a portion of the namespace, usually the directories frequently accessed by local client applications. Directory pointers connect parts of the the database that are distributed among two or more servers.

Figure 7–4 depicts a namespace with three DNS servers. The DESIGN node contains most of the namespace—the root directory plus the research and development directories. The applications directory resides on the APPLY node, while the hardware directory resides on the SHOP node.

DNS refers to a collection of directories on a server as a **clearinghouse**.

Figure 7–4: A Partitioned Namespace



7.1.5.1 Replicating Directories

In large networks, many applications rely on DNS and names must be available for the application to work. To ensure availability, DNS allows the duplication of data and provides a mechanism to keep all copies of names synchronized. Then, if one server becomes disabled, applications can still access the namespace through another server. Whenever data is duplicated, DNS copies one or more directories with all their contents.

The namespace administrator determines how many copies of each directory should exist and where they should be located. For example, Figure 7–5 shows the same namespace as Figure 7–4. However, in Figure 7–5 the root directory is duplicated so that it exists on all three DNS servers.

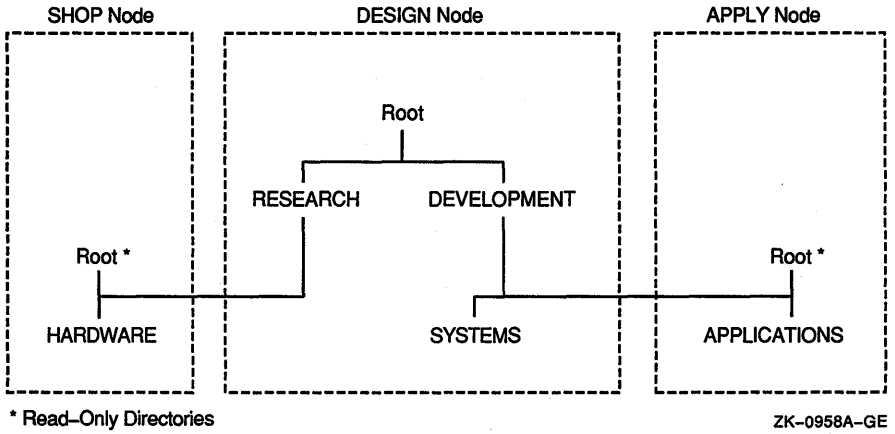
7.1.5.2 Types of Directories

Once you duplicate parts of a namespace, you generate different types of directories. Some are writable, while others are read-only. In a replicated namespace, there are three types of directories:

- Master
- Secondary
- Read-only

For example, in Figure 7–5 there are three copies of the root directory. The master copy resides on node DESIGN. Read-only copies reside on the other two nodes.

Figure 7–5: A Namespace with Replicated Directories



In a master directory an application can create or modify all types of entries: objects, directory pointers, and soft links. In a secondary directory an application can create or modify objects and soft links but not directory pointers. An application can retrieve namespace data from any type of directory.

When an application attempts to create a new object or update an existing one, the DNS clerk sends the request to a DNS server that has a secondary or master directory. The request to create an object succeeds as long as no other entry with the same name exists; the request to modify an object succeeds as long as the object is found in the directory.

7.1.5.3 Setting Confidence

An application can use the confidence argument in a $DNS call to stipulate the type of directory that the DNS clerk should use to service the call. For example, when an application wants to create an object, it can force the DNS clerk to create the object in the master directory by stipulating a high confidence level. Otherwise, DNS creates the object either in the master or in a secondary directory.

In a create or modify call, confidence has the following meaning:

- High confidence—Use the master directory.
- Medium confidence—Use the master or a secondary directory. There can be multiple copies of secondary directories.

An application's expression of confidence has a slightly different meaning in a request to find data. In this operation, there are three levels of confidence:

- High confidence—Use the master directory.
- Medium confidence—Use cached information to find the location of a DNS server but get the information from a DNS server.
- Low confidence—Use cached information.

7.1.5.4 Maintaining Consistency in Data

Whenever a directory is modified, the name service attempts to send the updated information to all directory replicas as long as the *convergence* attribute of the directory is set to high. Sometimes it is impossible to deliver the updates to all directory replicas, however, because a network link may be down or a node may be unreachable.

DNS does have a method of ensuring data consistency—it is called a **skulk**. In a skulk, DNS checks to see if data is consistent. If not, it gathers all updates made to a directory since the last skulk and propagates the updates to all replicas of the directory. If there is any discrepancy between information in a master and a secondary directory during a skulk, then the entry with the most recent time-stamping is used. Once the skulk is completed, DNS informs all directories of the time-stamping of the latest universal update.

When the convergence attribute is high, DNS skulks the namespace every 12 hours. When the convergence is low, the skulk occurs every 24 hours.

Directory replicas can lose their consistency between skulks. Two objects of the same name could be created simultaneously in different directory replicas or updates to the namespace might not be seen by all copies immediately. When DNS detects a conflict in replicas, it preserves the object with the most recent update time-stamping and deletes the older object. There is a chance that an application may get information from the namespace that DNS has not synchronized. In this case, an application has to have a mechanism to deal with the inconsistency.

7.1.6 Requesting Information from DNS

Once an application adds its objects to the namespace and modifies the objects to contain any necessary attributes, the application is ready to use the namespace. An application can request that the DNS clerk read information stored with an object or list all the application's objects that are stored in a particular directory. An application might also need to resolve all soft links in a name in order to identify a target entry.

For example, the VAX Distributed File Service (DFS) is a layered product that provides VMS users with the ability to use remote VMS disks as if they were attached to their local VMS system. The DFS application registers VMS directory structures (a directory and all of its subdirectories) with DNS. Each DFS object registered in the namespace names a particular file access point. DFS creates each object with a class attribute of DFS$ACCESSPOINT and modifies the address attribute (DNS$ADDRESS) of each object to hold the DECnet node address where the directory structures reside. As a final step in registering its resources, DFS creates a database to map DNS names to the appropriate VMS directory structures.

Whenever the DFS application receives the following mount request, DFS sends a request for information to the DNS clerk:

MOUNT ACCESS_POINT dns-name vms-logical-name

To read the address attribute of the access point object, the DFS application performs the following procedures:

1. Translates the DNS name that is supplied through the user interface to opaque format using the $DNS parse function
2. Reads the class attribute of the object with the $DNS read attribute function, indicating that there will be a second call to read other attributes of the object
3. Makes a second call to the $DNS service to read the address attribute of the object
4. Sends the DNS name to the DFS server, which looks up the disk where the access point is located
5. Verifies that the DNS name is valid on the DFS server

Then the DFS client and DFS server communicate to complete the mount function.

7.1.6.1 Reading Objects

When requesting information from DNS, an application always takes an object name from the user interface, translates the name into opaque format, and passes it in an item list to the DNS clerk.

The following C program shows how an application reads an object attribute. The $DNSW service uses an item list to return a set of objects. Then, the application calls a run-time routine to read each value in the set.

```
#include <dnsdef.h>
#include <dnsmsg.h>
/*
 * Parameters:
 *      opaque_objname = address of opaque full name for the object
 *                       containing the attribute to be read
 *      obj_len        = length of opaque full name of the object
 *      opaque_attname = address of the opaque simple name of the
 *                       attribute to be read
 *      attname_len    = length of opaque simple name of attribute
 */

read_attribute(opaque_objname, obj_len, opaque_attname, attname_len)
unsigned char *opaque_objname;
unsigned short obj_len;
unsigned char *opaque_attname;
unsigned short attname_len;
{
    struct $dnsb iosb;              /* Used to determine DNS status */
    char objtype = dns$k_object;    /* Using object entries         */

    struct $dnsitmdef readitem[6]; /* Item list for system service */
    struct dsc$descriptor set_dsc, value_dsc, newset_dsc, uid_dsc;

    unsigned char attvalbuf[dns$k_maxattribute]; /* To hold the attribute */
                            /* values returned from extraction routine. */
    unsigned char attsetbuf[dns$k_maxattribute]; /* To hold the set of    */
                         /* attribute values after the return from $DNSW. */
    unsigned char uidbuf[20];     /* Needed for context of multiple reads */

    int read_status;          /* Status of read attribute routine */
    int set_status;           /* Status of remove value routine */
    int xx;                   /* General variable used by print routine */

    unsigned short setlen;    /* Contains current length of set structure */
    unsigned short val_len; /* Contains length of value extracted from set */
    unsigned short uid_len;   /* Contains length of UID extracted from set */

    /* Construct an item list to read values of the attribute. */ ❶
    readitem[0].dns$w_itm_code = dns$_entry;
    readitem[0].dns$w_itm_size = obj_len;
    readitem[0].dns$a_itm_address = opaque_objname;

    readitem[1].dns$w_itm_code = dns$_lookingfor;
    readitem[1].dns$w_itm_size = sizeof(char);
    readitem[1].dns$a_itm_address = &objtype;

    readitem[2].dns$w_itm_code = dns$_attributename;
    readitem[2].dns$a_itm_address = opaque_attname;
    readitem[2].dns$w_itm_size = attname_len;
```

```
readitem[3].dns$w_itm_code = dns$_outvalset;
readitem[3].dns$a_itm_ret_length = &setlen;
readitem[3].dns$w_itm_size = dns$k_maxattribute;
readitem[3].dns$a_itm_address = attsetbuf;

*((int *)&readitem[4]) = 0;

do  ❷
{
    read_status = sys$dnsw(0, dns$_read_attribute, &readitem, &iosb, 0, 0);

    if(read_status == SS$_NORMAL)
    {
        read_status = iosb.dns$l_dnsb_status;
    }

    if((read_status == SS$_NORMAL) || (read_status == DNS$_MOREDATA))
    {
        setlen--;
        do  ❸
        {
            set_dsc.dsc$w_length = setlen;
            set_dsc.dsc$a_pointer = &attsetbuf[0]; /* Address of set */

            value_dsc.dsc$w_length = dns$k_simplenamemax;
            value_dsc.dsc$a_pointer = attvalbuf;  /* Buffer to hold  */
                                                  /* attribute value */

            uid_dsc.dsc$w_length = 20;
            uid_dsc.dsc$a_pointer = uidbuf; /* Buffer to hold value's UID*/

            newset_dsc.dsc$w_length = dns$k_maxattribute;
            newset_dsc.dsc$a_pointer = &attsetbuf[0]; /* Same buffer for */
                                                      /* each call      */

            set_status = dns$remove_first_set_value(&set_dsc, &value_dsc,
                         ❹                          &val_len, &uid_dsc,
                                                    &uid_len, &newset_dsc,
                                                    &setlen);

            if(set_status == SS$_NORMAL)
            {  ❺
                readitem[4].dns$w_itm_code = dns$_contextvartime;
                readitem[4].dns$w_itm_size = uid_len;
                readitem[4].dns$a_itm_address = uidbuf;

                *((int *)&readitem[5]) = 0;
```

```
                printf("\tValue: ");    ❻
                for(xx = 0; xx < val_len; xx++)
                    printf("%x ", attvalbuf[xx]);
                printf("\n");
            }
            else if (set_status != 0)
            {
                printf("Error %d returned when removing value from set\n",
                       set_status);
                exit(set_status);
            }
        } while(set_status == SS$_NORMAL);
    }
    else
    {
        printf("Error reading attribute = %d\n", read_status);
        exit(read_status);
    }
} while(read_status == DNS$_MOREDATA);
}
```

❶ The item list contains five entries:

- The opaque full name of the object with the attribute the program wants to read
- The type of namespace entry to access
- The opaque simple name of the attribute to read
- The address of the buffer containing the set of values returned by the read operation
- A zero to terminate the item list

❷ The loop repeatedly calls the $DNSW service to read the values of the attribute because the first call might not return all the values. The loop executes until $DNSW returns something other than DNS$_MOREDATA.

❸ This loop extracts all values from the set returned by $DNSW, one value at a time. The routine takes the address of the second byte in the structure because the first byte indicates the attribute type. This routine sets up descriptors for buffers that are used by the DNS$REMOVE_FIRST_SET_VALUE routine to extract values from the set. The loop executes until all values are extracted from the set or it encounters an error.

❹ The DNS$REMOVE_FIRST_SET_VALUE routine extracts a value from the set.

❺ This attribute name might be the context the routine uses to read additional attributes. The attribute's UID, not its value, provides the context.

❻ Finally, display the value in hexadecimal format. (You could also take the attribute name and convert it to a printable format before displaying the result.)

7.1.6.2 Listing Information

The list functions of $DNS allow applications to list the objects, subdirectories, or soft links in a specific directory. Either the asterisk (*) or question mark (?) wildcard, described in Section 7.1.2.3, allows an application to screen on the basis of its facility name.

The values DNS returns from read or enumerate functions are in different structures. For example, an enumeration of objects returns different structures than an enumeration of directories.

The following C program shows how an application can read the objects in a directory with the $DNS system service. It demonstrates how you parse any set that the enumerate objects function returns with a run-time routine in order to remove the first entry from the set. The example also demonstrates how the program takes each value from the set.

```
#include <dnsdef.h>
#include <dnsmsg.h>
/*
 * Parameters:
 *      fname_p   : opaque full name of the directory to enumerate
 *      fname_len : length of full name of the directory
 */

struct $dnsitmdef enumitem[4];            /* Item list for enumeration */
unsigned char setbuf[100];                /* Values from enumeration */
struct $dnsb enum_iosb;       /* DNS status information */
int synch_event;              /* Used for synchronous AST threads */
unsigned short setlen;        /* Length of output in setbuf */

enumerate_objects(fname_p, fname_len)
unsigned char *fname_p;
unsigned short fname_len;
{
    int enumerate_objects_ast();

    int status;              /* General routine status */
    int enum_status;         /* Status of enumeration routine */

    /* Set up item list */

    enumitem[0].dns$w_itm_code = dns$_directory; /* Opaque directory name */
    enumitem[0].dns$w_itm_size = fname_len;
    enumitem[0].dns$a_itm_address = fname_p;

    enumitem[1].dns$w_itm_code = dns$_outobjects;  /* output buffer */
    enumitem[1].dns$a_itm_ret_length = &setlen;
    enumitem[1].dns$w_itm_size = 100;
    enumitem[1].dns$a_itm_address = setbuf;

    *((int *)&enumitem[2]) = 0; /* Zero terminate item list */

    status = lib$get_ef(&synch_event); ❶
```

```
    if(status != SS$_NORMAL)
    {
        printf("Could not get event flag to synch AST threads\n");
        exit(status);
    }

    enum_status = sys$dns(0, dns$_enumerate_objects, &enumitem,
                ❷    &enum_iosb, enumerate_objects_ast, setbuf);

    if(enum_status != SS$_NORMAL)    ❸
    {
        printf("Error enumerating objects = %d\n", enum_status);
        exit(enum_status);
    }
    status = sys$synch(synch_event, &enum_iosb);    ❹

    if(status != SS$_NORMAL)
    {
        printf("Synchronization with AST threads failed\n");
        exit(status);
    }
}

/* AST routine parameter:                          */
/*      outbuf : address of buffer that contains enumerated names. */
                                                    ❺
unsigned char objnamebuf[dns$k_simplenamemax]; /* Opaque object name */

enumerate_objects_ast(outbuf)
unsigned char *outbuf;
{
    struct $dnsitmdef cvtitem[3];              /* Item list for class name */
    struct $dnsb iosb;        /* Used for name service status information */
    struct dsc$descriptor set_dsc, value_dsc, newset_dsc;

    unsigned char simplebuf[dns$k_simplestrmax];   /* Object name string */

    int enum_status;   /* The status of the enumeration itself */
    int status;        /* Used for checking immediate status returns */
    int set_status;    /* Status of remove value routine */

    unsigned short val_len;     /* Length of set value */
    unsigned short sname_len;   /* Length of object name */

    enum_status = enum_iosb.dns$l_dnsb_status;  /* Check status */
    if((enum_status != SS$_NORMAL) && (enum_status != DNS$_MOREDATA))
    {
        printf("Error enumerating objects = %d\n", enum_status);
        sys$setef(synch_event);
        exit(enum_status);
    }

    do
    {
        /*
         * Extract object names from output buffer one
         * value at a time.  Set up descriptors for the extraction.
         */
        set_dsc.dsc$w_length = setlen;     /* Contains address of */
        set_dsc.dsc$a_pointer = setbuf;    /* the set whose values */
                                           /* are to be extracted */
```

```
    value_dsc.dsc$w_length = dns$k_simplenamemax;
    value_dsc.dsc$a_pointer = objnamebuf; /* To contain the */
                                          /* name of an object */
                                          /* after the extraction */

    newset_dsc.dsc$w_length = 100;       /* To contain a new */
    newset_dsc.dsc$a_pointer = setbuf;   /* set structure after */
                                         /* the extraction. */

    /* Call RTL routine to extract the value from the set */
    set_status = dns$remove_first_set_value(&set_dsc, &value_dsc, &val_len,
                                            0, 0, &newset_dsc, &setlen);

    if(set_status == SS$_NORMAL)
    {                                                    6
        cvtitem[0].dns$w_itm_code = dns$_fromsimplename;
        cvtitem[0].dns$w_itm_size = val_len;
        cvtitem[0].dns$a_itm_address = objnamebuf;

        cvtitem[1].dns$w_itm_code = dns$_tostringname;
        cvtitem[1].dns$w_itm_size = dns$k_simplestrmax;
        cvtitem[1].dns$a_itm_address = simplebuf;
        cvtitem[1].dns$a_itm_ret_length = &sname_len;

        *((int *)&cvtitem[2]) = 0;

        status = sys$dnsw(0, dns$_simple_opaque_to_string, &cvtitem,
                          &iosb, 0, 0);

        if(status == SS$_NORMAL)
            status = iosb.dns$l_dnsb_status;  /* Check for errors */

        if(status != SS$_NORMAL) /* If error, terminate processing */
        {
            printf("Converting object name to string returned %d\n",
                   status);
            exit(status);
        }
        else
        {
            simplebuf[sname_len] = 0;   /* Null terminate for printing */
            printf("%s\n", simplebuf);
        }

        enumitem[2].dns$w_itm_code = dns$_contextvarname;    7
        enumitem[2].dns$w_itm_size = val_len;
        enumitem[2].dns$a_itm_address = objnamebuf;

        *((int *)&enumitem[3]) = 0;
    }
    else if (set_status != 0)
    {
        printf("Error %d returned when removing value from set\n",
               set_status);
        exit(set_status);
    }
} while(set_status == SS$_NORMAL);
```

```
    if(enum_status == DNS$_MOREDATA)
    {                                                    ❽
        enum_status = sys$dns(0, dns$_enumerate_objects, &enumitem,
                              &enum_iosb, enumerate_objects_ast, setbuf);

        if(enum_status != SS$_NORMAL)  /* Check status of $DNS */
        {
            printf("Error enumerating objects = %d\n", enum_status);
            sys$setef(synch_event);
        }
    }
    else
    {                                                    ❾
        sys$setef(synch_event);
    }
}
```

❶ Get an event flag to synchronize the execution of AST threads.

❷ Use the system service to enumerate the object names.

❸ Check the status of system service itself before waiting for threads.

❹ Use the $SYNCH call to make sure the DNS clerk has completed and that all threads have finished executing.

❺ After enumerating objects, $DNS calls an AST routine. The routine shows how DNS$REMOVE_FIRST_SET_VALUE extracts object names from the set returned by the DNS$_ENUMERATE_OBJECTS function.

❻ Use an item list to convert the opaque simple name to a string name so you can display it to the user. The item list contains the following entries:

- The address of the opaque simple name to be converted
- The address of the buffer that will hold the string name
- A zero to terminate the item list

❼ This object name could provide the context for continuing the enumeration. Append the context variable to the item list so the enumeration can continue from this name if there is more data.

❽ Use the system service to enumerate the object names as long as there is more data.

❾ Set the event flag to indicate that all AST threads have completed and the program can terminate.

7.1.6.3 How the Clerk Locates Data

When the DNS clerk receives an application's call for service, it tries to find a DNS server that can process the request.

Often, the DNS clerk does not know which DNS server holds the object information. To find an unknown server, the clerk looks in its own cache first. The clerk cache holds namespace information gathered from servicing earlier application requests. If the clerk cache does not list the needed server, then the DNS clerk requests information from a local DNS server in its cache. (A clerk always knows about at least one DNS server because this information is loaded at system startup.)

The clerk's last recourse is to trace directory pointers through the namespace. Any DNS server is capable of telling the clerk about another DNS server holding other directories in the namespace hierarchy. The clerk follows directory pointers until it finds a DNS server holding the specified directory. If the clerk cannot find the specified directory, then it follows directory pointers up to the root directory. Once the root directory is found, the clerk traces directory pointers away from the root, until it finds a DNS server that has the directory holding the requested object.

Once the clerk finds a directory that holds the required information, it delivers the request to the DNS server. As soon as the clerk receives a response, it transmits the result to the application.

7.2 DNS System Services

The Distributed Name Service Clerk system services are the programming interface to the VAX Distributed Name Service facility. The DNS Clerk system services allow an application to register a resource in a distributed database and then access the resource from any point in the network by a single name. There are two system service calls to the clerk that are described in this section.

- $DNS (Distributed Name Service Clerk)
- $DNSW (Distributed Name Service Clerk and Wait)

The $DNS system service is the asynchronous client interface for applications using the Distributed Name Service. The $DNSW system service is the synchronous client interface.

$DNS—Distributed Name Service Clerk

The Distributed Name Service Clerk service registers a resource in a distributed database. The $DNS service completes asynchronously; that is, it returns to the client immediately after making a name service call. The status returned to the client call indicates whether a request was successfully queued to the name service.

Note that the Distributed Name Service Clerk and Wait ($DNSW) call is the synchronous equivalent of $DNS. $DNSW is identical to $DNS in every way except that $DNSW returns to the caller after the operation completes.

format

SYS$DNS *[efn] ,func ,itmlst ,[dnsb] ,[astadr] ,[astprm]*

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return by immediate value a condition value in R0. Condition values returned by this call are listed in the section Condition Values Returned. Errors returned here are from the DNS clerk. Refer to the **dnsb** argument for errors returned by the name service.

arguments

efn

VMS Usage: **ef_number**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of the event flag to be set when $DNS completes. The **efn** argument is a longword containing this number. The **efn** argument is optional; if not specified, event flag 0 is set.

When $DNS begins execution, it clears the event flag. Even if the service encounters an error and completes without queuing a name service request, the specified event flag is set.

func

VMS Usage: **function_code**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Function code specifying the action that $DNS is to perform. The **func** argument is a longword containing this function code.

A single call to $DNS can specify one function code. Most function codes require or allow for additional information to be passed in the call with the **itmlst** argument.

$DNS Function Codes

DNS$_CREATE_OBJECT

This request creates an object in the namespace. Initially, the entry has the attributes of DNSUID, DNSUTS, DNS$CLASS, DNS$ACS, and DNS$CLASSVERSION. The name service creates the DNSUID, DNSUTS, and DNS$ACS attributes. The client application supplies the DNS$CLASS and DNS$CLASSVERSION attributes. You can add additional attributes using the DNS$_MODIFY_ATTRIBUTE function.

The DNS clerk cannot guarantee that an object has been created. Another DNS$_CREATE_OBJECT request could supersede the object created by your call. To verify an object creation, wait until the directory is skulked and then check to see if the requested object entry is present. If the value of the directory's DNS$ALLUPTO attribute is greater than the UID of the object entry, your object entry has been successfully created.

Creating an object in the namespace requires write access to the directory in which the object will reside.

If specified, DNS$_OUTUID holds the UID of the created object.

You must specify the following item codes:

DNS$_CLASS (Class_Name)
DNS$_OBJECTNAME (Opaque_Full_Name)
DNS$_VERSION (Class_Version)

You can specify the following input item codes:

DNS$_CONF
DNS$_WAIT

You can specify the following output item code:

DNS$OUTUID (UID)

$DNS returns the following:

SS$_NORMAL
DNS$_ENTRYEXISTS
DNS$_INVALID_OBJECTNAME
DNS$_INVALID_CLASSNAME
Any condition listed in the section Condition Values Returned.

$DNS returns the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_DELETE_OBJECT

This request removes the specified object from the namespace. The function requires delete access to the object in question.

You must specify the following input item code:

DNS$_OBJECTNAME (Opaque_Full_Name)

You can specify the following input item codes:

DNS$_CONF
DNS$_WAIT

$DNS returns the following:

SS$_NORMAL
DNS$_INVALID_OBJECTNAME
Any condition listed in the section Condition Values Returned.

$DNS returns the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_ENUMERATE_ATTRIBUTES

This request returns a set of attributes in DNS$_OUTATTRIBUTESET that is associated with the entry. The entry type is specified in the DNS$_LOOKINGFOR entry.

To manipulate the values returned by this call, use the DNS$REMOVE_FIRST_SET_VALUE run-time routine. The values returned are the Enum_Att_Name structure, which is described in Table 7–1.

You must have read access to the entry to enumerate its attributes.

The DNS clerk enumerates attributes in alphabetical order. A return status of DNS$_MOREDATA implies that not all attributes have been enumerated. You should make further calls, setting DNS$_CONTEXTVARNAME to the last attribute in the set returned, until the procedure returns SS$_NORMAL.

You must specify the following input item codes:

DNS$_ENTRY (Opaque_Full_Name)
DNS$_LOOKINGFOR (Entry_Type)

You must specify the following output item code:

DNS$_OUTATTRIBUTESET (set of Enum_Att_Name)

You can specify any of the following input item codes:

DNS$_CONF
DNS$_CONTEXTVARNAME (Opaque_Simple_Name)
DNS$_WAIT

$DNS can return the following:

SS$_NORMAL
DNS$_MOREDATA
DNS$_INVALID_ENTRYNAME
DNS$_INVALID_CONTEXTNAME
Any condition listed in the section Condition Values Returned.

$DNS returns the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_ENUMERATE_CHILDREN

This request takes as input a directory name with an optional simple name that uses a wildcard. The DNS clerk matches the input against child directory entries in the specified directory.

The DNS clerk returns a set of simple names of child directories in the target directory that match the name with the wildcard. A null set is returned when there is no match or when the directory has no children.

To manipulate the values returned by this call, use the DNS$REMOVE_FIRST_SET_VALUE run-time routine. The value returned is a simple name.

The function requires read access to the parent directory.

The child directories are enumerated in alphabetical order. If the call returns DNS$_MOREDATA, not all children have been enumerated and the client should make further calls, setting DNS$_CONTEXTVARNAME to the last child directory in the set returned, until the procedure returns SS$_NORMAL. Subsequent calls return the child directories, starting with the directory specified in DNS$_CONTEXTVARNAME and continuing in alphabetical order.

You must specify the following input item code:

DNS$_DIRECTORY (Opaque_Full_Name)

You must specify the following output item code:

DNS$_OUTCHILDREN (set of Opaque_Simple_Name)

You can specify the following input item codes:

DNS$_CONF
DNS$_CONTEXTVARNAME (Opaque_Simple_Name)

DNS$_WAIT
DNS$_WILDCARD (Opaque_Simple_Name)

$DNS returns the following:

SS$_NORMAL
DNS$_MOREDATA
DNS$_INVALID_DIRECTORYNAME
DNS$_INVALID_CONTEXTNAME
DNS$_INVALID_WILDCARDNAME

You might receive the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_ENUMERATE_OBJECTS

This request takes as input the directory name, a simple name that uses a wildcard, and a class name that uses a wildcard. The DNS clerk matches these against objects in the directory. If a wildcard and class filter are not specified, then all objects in the directory are returned.

The function returns (in DNS$_OUTOBJECTS) a set of simple names of objects in the directory that match the name with the wildcard. If no objects match the wildcard or the directory contains no objects, a null set is returned. The DNS clerk returns DNS$V_DNSB_OUTLINKED qualifying status if it encounters one or more soft links in resolving the names of object entries to be enumerated.

To manipulate the values returned by this call, use the DNS$REMOVE_FIRST_SET_VALUE run-time routine. The value returned is a simple name structure.

This function requires read access to the parent directory.

The objects are enumerated in alphabetical order. If the call returns DNS$_MOREDATA, not all objects have been enumerated and the client should make further calls, setting DNS$_CONTEXTVARNAME to the last object in the set returned, until the procedure returns SS$_NORMAL. If the class filter is specified, only those objects of the specified classes are returned.

You must specify the following input item code:

DNS$_DIRECTORY (Opaque_Full_Name)

You must specify the following output item code:

DNS$_OUTOBJECTS (set of Opaque_Simple_Names)

You can specify any of the following input item codes:

DNS$_WILDCARD (Opaque_Simple_Name)
DNS$_CLASSFILTER (Opaque_Simple_Name)

DNS$_CONTEXTVARNAME (Opaque_Simple_Name)
DNS$_CONF
DNS$_WAIT

$DNS returns the following:

SS$_NORMAL
DNS$_MOREDATA
DNS$_INVALID_DIRECTORYNAME
DNS$_INVALID_CONTEXTNAME
DNS$_INVALID_WILDCARDNAME
DNS$_INVALID_CLASSNAME

You might receive the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_ENUMERATE_SOFTLINKS

This request takes as input the name of a directory and a wildcarded simple name. The DNS clerk matches these against soft links in the directory. It returns (in DNS$_OUTSOFTLINKS) a set consisting of simple names of soft links in the directory that match the wildcarded name. If no soft link entries match the wildcard or the directory contains no soft links, a null set is returned.

If no wildcard is specified, then all soft links in the directory are returned.

To manipulate the values returned by this call, use the DNS$REMOVE_FIRST_SET_VALUE run-time routine. The value returned is a simple name.

This function requires read access to the parent directory.

The soft links are enumerated in alphabetical order. If the call returns DNS$_MOREDATA, not all matching soft links have been enumerated and the client should make further calls, setting DNS$_CONTEXTVARNAME to the last soft link in the set returned, until the procedure returns SS$_NORMAL.

You must specify the following input item code:

DNS$_DIRECTORY (Opaque_Full_Name)

You must specify the following output item code:

DNS$_OUTSOFTLINKS (set of Opaque_Simple_Name)

You can specify the following input item codes:

DNS$_WILDCARD (Opaque_Simple_Name)
DNS$_CONTEXTVARNAME (Opaque_Simple_Name)
DNS$_CONF

DNS$_WAIT

$DNS returns the following:

SS$_NORMAL
DNS$_INVALID_DIRECTORYNAME
DNS$_INVALID_CONTEXTNAME
DNS$_INVALID_WILDCARDNAME

You might receive the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_FULL_OPAQUE_TO_STRING

This request converts a full name in opaque format to its equivalent in string format, as described in Section 7.1.1.4. Setting the byte referred to by DNS$_SUPPRESS_NSNAME to 1 prevents the namespace name from being included in the string name.

You must specify the following item codes:

DNS$_FROMFULLNAME (Opaque_Full_Name)
DNS$_TOSTRINGNAME (Full_Name_Str)

You can specify the following input item code:

DNS$_SUPPRESS_NSNAME (byte)

$DNS returns the following:

SS$_NORMAL
DNS$_INVALIDNAME

You do not receive qualifying status.

DNS$_MODIFY_ATTRIBUTE

This request applies one update to the specified entry in the namespace. You can add or remove an attribute; you can add or remove a value from either a single-value attribute or a set-valued attribute.

This operation requires write or delete access to the entry whose attribute is being modified, depending on whether the operation adds or removes the attribute.

When adding a value to a single-value attribute, include a value in DNS$_MODVALUE or you will receive the error DNS$_INVALIDUPDATE. The item code DNS$_MODVALUE is not required when writing to an attribute set because the name service creates the attribute if no value is provided.

In a delete operation, include the DNS$_MODVALUE item code to remove a certain value from an attribute set. Unless you specify the item code, the name service removes the attribute and all its values from the entry.

You must specify the following item codes:

DNS$_ENTRY (Opaque_Full_Name)
DNS$_LOOKINGFOR (Entry_Type)
DNS$_MODOPERATION (DNS$K_PRESENT or DNS$K_ABSENT)
DNS$_ATTRIBUTETYPE (DNS$K_SET or DNS$K_SINGLE)
DNS$_ATTRIBUTENAME (Opaque_Simple_Name)

You can specify the following input item codes:

DNS$_CONF
DNS$_MODVALUE
DNS$_WAIT

$DNS returns the following:

SS$_NORMAL
DNS$_WRONGATTRIBUTETYPE
DNS$_INVALIDUPDATE
DNS$_INVALID_ENTRYNAME
DNS$_INVALID_ATTRIBUTENAME

You might receive the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_PARSE_FULLNAME_STRING

This request takes a full name in string format and converts it to its equivalent in opaque format. If DNS$_NEXTCHAR_PTR is used, the longword referenced by this entry contains the address of the character immediately following the DNS name given in DNS$_FROMSTRINGNAME.

You must specify the following item codes:

DNS$_FROMSTRINGNAME (Full_Name_Str)
DNS$_TOFULLNAME (Opaque_Full_Name)

You can specify the following input item code:

DNS$_NEXTCHAR_PTR

$DNS can return the following:

SS$_NORMAL
DNS$_INVALIDNAME

You do not receive qualifying status.

DNS$_PARSE_SIMPLENAME_STRING

This request takes a simple name in string format and converts it to its equivalent in opaque format. If DNS$_NEXTCHAR_PTR is used, the longword referenced by this entry contains the address of

the character immediately following the DNS name given in DNS$_FROMSTRINGNAME.

You must specify the following item codes:

DNS$_FROMSTRINGNAME (Simple_Name_Str)
DNS$_TOFULLNAME (Opaque_Simple_Name)

You can specify the following input item code:

DNS$_NEXTCHAR_PTR

$DNS can return the following:

SS$_NORMAL
DNS$_INVALIDNAME

You do not receive qualifying status.

DNS$_READ_ATTRIBUTE

This request returns (in DNS$_OUTVALSET) a set whose members are the values of the specified attribute. When the request completes successfully, the qualifying status indicates whether soft links were followed in resolving the name.

This function requires read access to the object whose attribute is to be read.

To manipulate the values returned by this call, use the DNS$REMOVE_FIRST_SET_VALUE run-time routine. The contents of DNS$_OUTVALSET are passed to DNS$REMOVE_FIRST_SET_VALUE, and the routine returns the value of the attribute.

The attribute values are returned in the order they were received. If the call returns DNS$_MOREDATA, not all values have been returned. The client application can make further calls, setting DNS$_CONTEXTVARTIME to the time-stamping of the last attribute in the set returned, until the procedure returns SS$_NORMAL. If the client sets the DNS$_MAYBEMORE argument to 1, the name service attempts to make subsequent DNS$_READ_ATTRIBUTE calls for the same entry more efficient. The client may set this argument to true on any call, but performance improves only if the client accesses no other entry before making a read attribute call for the previous entry.

You must include the following input item codes:

DNS$_ENTRY (Opaque_Full_Name)
DNS$_LOOKINGFOR (Entry_Type)
DNS$_ATTRIBUTENAME (Opaque_Simple_Name)

You must include the following output item code:

DNS$_OUTVALSET (set of values)

You can include the following input item codes:

DNS$_MAYBEMORE (Boolean)
DNS$_CONTEXTVARTIME (UID)
DNS$_CONF
DNS$_WAIT

$DNS returns the following:

SS$_NORMAL
DNS$_MOREDATA
DNS$_INVALID_ENTRYNAME
DNS$_INVALID_ATTRIBUTENAME

You might receive the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_RESOLVE_NAME

This request follows a chain of soft links to its destination, returning the full name of that entry so that future calls by the client application can use the entry name without incurring the overhead of following the link.

This function requires read access to each of the soft links in the chain.

Applications that maintain their own databases of opaque DNS names should use DNS$_RESOLVE_NAME any time they receive the qualifying status DNS$V_DNSB_OUTLINKED. This status indicates a need to update the current name, using the soft link facility of DNS. Use the original name with DNS$_RESOLVE_NAME and store the result in the application database.

If the application provides a name that does not contain any soft links, DNS$_NOTLINKED status is returned. If the target of any of the chain of soft links followed does not exist, the DNS$_DANGLINGLINK status is returned. To obtain the target of any particular soft link, use the DNS$_READ_ATTRIBUTE function with DNS$_LOOKINGFOR set to DNS$K_SOFTLINK and request the attribute DNS$LINKTARGET. This can be useful in discovering which link in a chain is "broken." If the DNS clerk detects a loop, it returns DNS$_POSSIBLECYCLE status.

You must specify the following input item code:

DNS$_LINKNAME (Opaque_Full_Name)

You must specify the following output item code:

DNS$_OUTNAME (Opaque_Full_Name)

You can specify the following input item codes:

DNS$_CONF
DNS$_WAIT

$DNS returns the following:

SS$_NORMAL
DNS$_INVALID_LINKNAME
DNS$_NOTLINKED

You might receive the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_SIMPLE_OPAQUE_TO_STRING

This request takes a simple name in opaque format and converts it to its equivalent in string format, as described in Section 7.1.1.4.

You must specify the following item codes:

DNS$_FROMSIMPLENAME (Opaque_Simple_Name)
DNS$_TOSTRINGNAME (Simple_Name_Str)

$DNS returns the following:

SS$_NORMAL
DNS$_INVALIDNAME

You do not receive qualifying status.

DNS$_SIMPLENAME_STRING_TO_OPAQUE

This request converts a simple name in string format to its equivalent in opaque format. If DNS$_NEXTCHAR_PTR is used, the longword referenced by this entry contains the address of the character immediately following the DNS name given in DNS$_FROMSTRINGNAME.

You must specify the following item codes:

DNS$_FROMSTRINGNAME (Simple_Name_Str)
DNS$_TOSIMPLENAME (Opaque_Simple_Name)

You can specify the following input item code:

DNS$_NEXTCHAR_PTR

$DNS can return the following:

DNS$_INVALIDNAME

You do not receive qualifying status.

DNS$_TEST_ATTRIBUTE

This request returns DNS$_TRUE if the specified attribute has one of the following characteristics:

- It is a single-value attribute and its value matches the client-specified value.
- It is a set-valued attribute and the attribute contains the client-specified value as one of its members.

On successful completion of the function, DNS$V_DNSB_OUTLINKED indicates whether soft links were followed in resolving the name.

This function requires test or read access to the entry whose attribute is to be tested.

If the attribute is not present in the entry, the function returns DNS$_FALSE.

You must specify the following item codes:

DNS$_ENTRY (Opaque_Full_Name)
DNS$_LOOKINGFOR (Entry_Type)
DNS$_ATTRIBUTENAME (Opaque_Simple_Name)
DNS$_VALUE (value)

You can specify the following input item codes:

DNS$_CONF
DNS$_WAIT

$DNS returns the following when the call is successful:

DNS$_TRUE
DNS$_FALSE

$DNS returns the following when the call is unsuccessful:

DNS$_INVALID_ENTRYNAME
DNS$_INVALID_ATTRIBUTENAME

You might receive the following qualifying status:

DNS$V_DNSB_OUTLINKED

DNS$_TEST_GROUP

This request tests for group membership. It returns DNS$_TRUE if the specified member is a member of the specified group (or a subgroup thereof), and DNS$_FALSE otherwise. If a recursive search is required and one or more of the subgroups is unavailable, the status encountered in trying to access that group is returned.

The DNS$_INOUTDIRECT argument, on input, controls the scope of the search. If set to true, the only group considered is the top level group specified by the group argument. If set to false, recursive evaluation is performed. On output, the DNS$_INOUTDIRECT argument is set to 1 if the member was found in the top level group; otherwise it is set to 0.

You must specify the following item codes:

DNS$_GROUP (Opaque_Full_Name)
DNS$_MEMBER (Opaque_Full_Name)

You can specify the following input item codes:

DNS$_CONF
DNS$_INOUTDIRECT (Boolean)
DNS$_WAIT

$DNS returns the following:

SS$_NORMAL
DNS$_NOTAGROUP
DNS$_INVALID_GROUPNAME
DNS$_INVALID_MEMBERNAME

You might receive the following qualifying status:

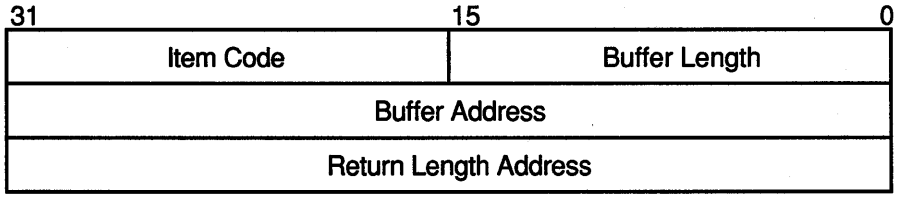
DNS$V_DNSB_INOUTDIRECT

itmlst

VMS Usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list supplying information to be used in performing the function specified by the **func** argument. The **itmlst** argument is the address of the item list. The item list consists of one or more item descriptors, each of which is three longwords. The descriptors can be in any order in the item list. Each item descriptor specifies an item code. Each item code either describes the specific information to be returned by $DNS or otherwise affects the action designated by the function code. The item list is terminated by a longword of zero.

The item list is a standard VMS format item list. The following figure depicts the general structure of an item descriptor:



ZK–1705–GE

$DNS Item Descriptor Fields

item code
A word containing a symbolic code describing the nature of the information currently in the buffer or to be returned in the buffer. The location of the buffer is pointed to by the buffer address field. Each item code has a symbolic name; these symbolic names are defined by the $DNS macro and have the format DNS$_code.

buffer length
A word specifying the length of the buffer; the buffer either supplies information to be used by $DNS or receives information from $DNS. The required length of the buffer varies depending on the item code specified; each item code description specifies the required length.

buffer address
A longword containing the address of the buffer that specifies or receives the information.

return length address
A longword containing the address of a word specifying the actual length in bytes of the information returned by $DNS. The information resides in a buffer identified by the buffer address field. The field applies to output item list entries only and is ignored for input entries. If the return length address is 0, it is ignored.

$DNS Item Codes

DNS$_ATTRIBUTETYPE
The DNS$_ATTRIBUTETYPE item code specifies whether an attribute is set valued (DNS$K_SET) with a value of 3 or single valued (DNS$K_ SINGLE) with a value of 2.

DNS$_ATTRIBUTENAME

The DNS$_ATTRIBUTENAME item code specifies the opaque simple name of an attribute. An attribute name cannot be longer than 31 characters.

DNS$_CLASS

The DNS$_CLASS item code specifies the class of an object for the $DNS function DNS$_CREATE_OBJECT. DNS$_CLASS is an opaque simple name.

DNS$_CLASSFILTER

DNS$_CLASSFILTER is used by the $DNS function DNS$_ENUMERATE_OBJECTS to limit the scope of the enumeration to those objects belonging to a certain class (or, if a wildcard name is used, a group of classes). DNS$_CLASSFILTER is an opaque simple name, which can use a wildcard.

DNS$_CLASSFILTER is optional. A wildcard simple name of * is used by default, meaning that objects of all classes will be enumerated.

DNS$_CONF

DNS$_CONF specifies for $DNS the level of importance in returning up-to-date information. DNS$_CONF is 1 byte long and can take one of the following values:

Confidence Level	Value	Description
DNS$K_LOW	1	Service the DNS clerk request at the lowest cost, usually from cached information.
DNS$K_MEDIUM	2	Bypass any cached information and obtain the data directly from a name server.
DNS$K_HIGH	3	Service the request from a master directory.

The entry is optional; if it is not specified, the DNS clerk assumes a value of DNS$K_LOW.

DNS$_CONTEXTVARNAME

DNS$_CONTEXTVARNAME is used by the enumeration functions of $DNS to specify a context from which the enumeration is to begin. The item is an opaque simple name.

DNS$_CONTEXTVARNAME is optional. If not given, the enumeration begins with the first element.

DNS$_DIRECTORY

DNS$_DIRECTORY is used by most of the enumeration functions of $DNS to specify the namespace directory in which the elements of the enumeration are to be found. DNS$_DIRECTORY is an opaque full name.

DNS$_ENTRY
DNS$_ENTRY specifies for $DNS the opaque full name of a namespace entry (object, soft link, directory, clearinghouse).

DNS$_FROMFULLNAME
DNS$_FROMFULLNAME specifies for the DNS$_FULL_OPAQUE_TO_STRING function the opaque full name that is to be converted into string format.

DNS$_FROMSIMPLENAME
DNS$_FROMSIMPLENAME specifies for the DNS$_SIMPLE_OPAQUE_TO_STRING function the opaque simple name that is to be converted into string format.

DNS$_FROMSTRINGNAME
DNS$_FROMSTRINGNAME specifies a name in string format for the parse functions DNS$_PARSE_FULLNAME_STRING and DNS$_PARSE_SIMPLENAME_STRING that is to be converted to opaque format.

DNS$_GROUP
DNS$_GROUP specifies for the DNS$_TEST_GROUP function the opaque full name of the group that is to be tested. DNS$_GROUP must be the name of a group object.

DNS$_INOUTDIRECT
DNS$_INOUTDIRECT is a Boolean value that serves two different purposes for the DNS$_TEST_GROUP function. On input, DNS$_INOUTDIRECT controls the scope of the search for the test, as follows:

Value	Definition
1	The only group to be tested is the top level group specified by the DNS$_GROUP item.
0	All subgroups of the group named in DNS$_GROUP are tested for inclusion. A subgroup is any member that is a group in itself.

On output, DNS$_INOUTDIRECT is set to indicate whether the members were found in the top level group or were found as members of one of the subgroups, as follows:

Value	Definition
1	The member was found in the top level group.
0	The member was found in one of the subgroups of the top level group.

DNS$_INOUTDIRECT is a single-byte value.

DNS$_LINKNAME
DNS$_LINKNAME specifies the opaque full name of a soft link.

DNS$_LOOKINGFOR
DNS$_LOOKINGFOR specifies the type of entry on which the call is to operate. DNS$_LOOKINGFOR, which is encoded as a byte, can take one of the following values:

Type of Entry	Value
DNS$K_DIRECTORY	1
DNS$K_OBJECT	2
DNS$K_CHILDDIRECTORY	3
DNS$K_SOFTLINK	4
DNS$K_CLEARINGHOUSE	5

DNS$_MAYBEMORE
DNS$_MAYBEMORE is used with the DNS$_READ_ATTRIBUTE function to indicate that the results of the read operation are to be cached. This is a single-byte item.

When this item is set to 1, the name service returns as much information about the attributes for the entry as it is able to fit in the return buffer. All of this information is cached to make later lookups of attribute information for the entry quicker and more efficient.

If this item is not supplied, then only the requested information for the entry is returned.

DNS$_MEMBER
DNS$_MEMBER specifies for the DNS$_TEST_GROUP function of $DNS the opaque full name of a member that is to be tested for inclusion within a given group.

DNS$_MODOPERATION
DNS$_MODOPERATION specifies for the DNS$_MODIFY_ATTRIBUTE function the type of operation that is to take place. There are two types of modifications: adding an attribute (DNS$K_PRESENT), which has a value of 1, or deleting an attribute (DNS$K_ABSENT), which has a value of 0.

The name service adds an attribute in the following way:

- For an existing attribute where an attribute value is given, the value is added to a set-valued attribute and all other values for the set are unaffected. The value replaces any previous value in a single-value attribute.

- For an existing attribute where an attribute value is not given, all previous values for the attribute are unaffected.
- For a new attribute
  - — Where an attribute is given, the attribute is created and given the attribute type of DNS$K_SET or DNS$K_SINGLE as supplied with the DNS$K_ATTRIBUTETYPE item. The value is assigned to the attribute.
  - — Where an attribute value is not given, a set-valued attribute is created without a value assignment, but a single-value attribute is *not* created.

The name service deletes an attribute in the following way:

- If the attribute exists and an attribute value is given, the attribute value is removed from a set-valued attribute. All other values are unaffected. For a single-value attribute, the attribute (along with its value) is removed from the entry.
- If an attribute value is not given, then the attribute and all values of the attribute are removed. This is true for both set-valued attributes and single-value attributes.

DNS$_MODVALUE

DNS$_MODVALUE specifies for the DNS$_MODIFY_ATTRIBUTE function the value that is to be added to or deleted from an attribute. The structure of this value is dependent on the application.

DNS$_MODVALUE is an optional argument that affects the overall operation of the DNS$_MODIFY_ATTRIBUTE function. (See the DNS$_ MODOPERATION item code description for more information.)

DNS$_NEXTCHAR_PTR

DNS$_NEXTCHAR_PTR is an optional item code that can be used with the parse functions DNS$_PARSE_FULLNAME_STRING and DNS$_ PARSE_SIMPLENAME_STRING to return the address of the character that immediately follows a valid DNS name. This option is most useful when applications are parsing command line strings.

Without this item code, the parse functions return an error if any portion of the name string is invalid.

DNS$_OBJECTNAME

DNS$_OBJECTNAME specifies the opaque full name of an object.

DNS$_OUTATTRIBUTESET

DNS$_OUTATTRIBUTESET specifies to the DNS$_ENUMERATE_ ATTRIBUTES function the address of a buffer that is to contain the set of enumerated attribute names.

The names returned in this set can be extracted from the buffer with the DNS$REMOVE_FIRST_SET_VALUE routine. The resulting values are contained in the $DNSATTRSPECDEF structure, a byte indicating whether an attribute is set-value or single-value followed by an opaque simple name.

DNS$_OUTNAME

DNS$_OUTNAME specifies for the DNS$_RESOLVE_NAME function the address of a buffer that is to contain the opaque full name of the namespace entry that is pointed to by a soft link.

DNS$_OUTOBJECTS

DNS$_OUTOBJECTS specifies for the DNS$_ENUMERATE_OBJECTS function the address of a buffer that is to contain the set of opaque simple names returned by the enumeration.

The values resulting from the enumeration can be extracted using the DNS$REMOVE_FIRST_SET_VALUE routine. The resulting values are the opaque simple names of the objects found in the directory.

DNS$_OUTCHILDREN

DNS$_OUTCHILDREN specifies for the DNS$_ENUMERATE_CHILDREN function the address of a buffer that is to contain the set of opaque simple names returned by the enumeration.

The values resulting from the enumeration can be extracted using the DNS$REMOVE_FIRST_SET_VALUE routine. These values are the opaque simple names of the child directories found in the parent directory.

DNS$_OUTSOFTLINKS

DNS$_OUTSOFTLINKS specifies for the DNS$_ENUMERATE_SOFTLINKS function the address of a buffer that is to contain the set of opaque simple names returned by the enumeration.

The values resulting from the enumeration can be extracted using the DNS$REMOVE_FIRST_SET_VALUE routine. The resulting values are the opaque simple names of the soft links found in the directory.

DNS$_OUTVALSET

DNS$_OUTVALSET specifies for the DNS$_READ_ATTRIBUTE function the address of a buffer that is to contain the set of values for the given attribute.

The values of the set placed in this buffer can be extracted using the DNS$REMOVE_FIRST_SET_VALUE routine. The extracted values are the values of the attribute.

DNS$_OUTUID
DNS$_OUTUID is an optional item code that contains the address of a buffer used by the create functions of $DNS to return the unique identifier (UID). The UID is the time-stamping the entry received at creation.

DNS$_SUPPRESS_NSNAME
DNS$_SUPPRESS_NSNAME is an optional item for the DNS$_FULL_OPAQUE_TO_STRING function that is used to indicate that the leading namespace name should not be returned in the converted full name string. This is a single-byte value.

A value of 1 suppresses the leading namespace name in the resulting full name string.

DNS$_TOFULLNAME
DNS$_TOFULLNAME specifies for the DNS$_PARSE_FULLNAME_STRING function the address of a buffer that will contain the resulting opaque full name.

DNS$_TOSIMPLENAME
DNS$_TOSIMPLENAME specifies for the DNS$_PARSE_SIMPLENAME_STRING function the address of a buffer that will contain the resulting opaque simple name.

DNS$_TOSTRINGNAME
DNS$_TOSTRINGNAME specifies the address of a buffer that is to contain the string name resulting from one of the conversion functions: DNS$_FULL_OPAQUE_TO_STRING or DNS$_SIMPLE_OPAQUE_TO_STRING.

DNS$_VALUE
DNS$_VALUE specifies for the DNS$_TEST_ATTRIBUTE function the value that is to be tested. This item contains the address of a buffer holding the value.

DNS$_VERSION
DNS$_VERSION specifies for the DNS$_CREATE_OBJECT function the version associated with an object. This item contains the address of a $DNSCVERSDEF (CLASSVERSION) structure. This is a 2-byte structure: the first byte contains the major version number; the second contains the minor version number.

DNS$_WAIT
DNS$_WAIT enables the client to specify a timeout value to wait for a call to complete. If the timeout expires, the call returns either DNS$K_TIMEOUTNOTDONE or DNS$K_TIMEOUTMAYBEDONE, depending on whether the name space was updated by the incomplete operation.

The $BINTIM service converts an ASCII string time value to the quadword time value required by $DNS.

The parameter is optional; if it is not specified, a system-defined default timeout value of 10 minutes is assumed.

DNS$_WILDCARD

DNS$_WILDCARD is an optional item code that specifies to the enumeration functions of $DNS the opaque simple name used to limit the scope of the enumeration. (The simple name does not have to use a wildcard.) Only those simple names that match the wildcard are returned by the enumeration.

Item Code Identifiers

The identifiers shown in Table 7–1 are data structures that are used in item code arguments. Each data structure defines the encoding of an item list element.

Table 7–1: DNS Item Code Arguments

Item Code Identifier	Description
Attribute_Name	The structure of an opaque simple name, limited to 31 ISO Latin 1 characters.
Attribute_Name_Str	An attribute name string with the structure of a simple name string but limited to 31 ISO Latin 1 characters.
Boolean	A 1-byte field with the value 0 if false and 1 if true.
Class_Name	An opaque simple name, limited to 31 ISO Latin 1 characters.
Class_Name_Str	A simple name string, limited to 31 ISO Latin 1 characters.
Class_Version	A 2-byte field specifying major and minor version numbers associated with the object class.
Confidence	A 1-byte field with the value: DNSK_LOW, DNSK_MEDIUM, or DNS$K_HIGH.
Entry_Type	A 1-byte field with the value DNSK_OBJECT, DNSK_SOFTLINK, DNS$K_DIRECTORY, or DNS$K_CLEARINGHOUSE.
Enum_Att_Name	A structure consisting of a single byte, indicating whether the attribute is a set (DNSK$_SET) or a single value (DNS$K_SINGLE), followed by an opaque simple name.

(continued on next page)

Table 7–1 (Cont.): DNS Item Code Arguments

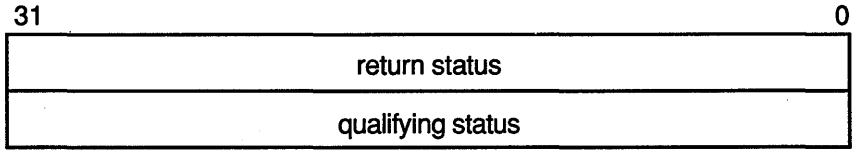
Item Code Identifier	Description
Full_Name_String	A full name string with the following structure: [NS_name:] [.] Namestring [.Namestring] NS_name:, if present, is a local system representation of the NSUID, the unique identifier of the name server. The DNS clerk supplies a namespace name (*node-name*_NS) if the value is omitted. Namestring represents a simple name component. Multiple simple names are separated by periods. You can include the asterisk wildcard (*) and simple name strings within quotation marks.
Group_Member	A structure consisting of a single byte, indicating whether the entry is a principal (DNS$K_GRPMEM_NOT_GROUP) or another group (DNS$K_GRPMEM_IS_GROUP), followed by the opaque full name of the member.
Opaque_Full_Name	The internal format of the complete name identifier for an object. The maximum output of DNS$PARSE_FULLNAME_STRING is 402 bytes.
Opaque_Simple_Name	A simple name specifies the internal format of one component of an Opaque_Full_Name. The Opaque_Simple_Name is the output of the DNS$PARSE_SIMPLENAME_STRING routine. It can be no longer than 257 bytes.
Simple_Name_Str	One term consisting of a string of ASCII characters with its length stored separately in an item list.

dnsb

VMS Usage: **dns_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

Status block to receive the final completion status of the $DNS operation. The **dnsb** argument is the address of the quadword $DNS status block.

The following figure depicts the structure of a $DNS status block:



ZK–1080A–GE

Status Block Fields

return status

Set on completion of a DNS clerk request to indicate the success or failure of the operation. Check the qualifying status word for additional information about a request marked as successful. Wherever possible, each function code description includes return status values.

qualifying status

This field consists of a set of flags that provide additional information about a successful name service operation. Wherever possible, each function code description includes qualifying status values.

The qualifying status values are defined as follows:

- DNS$V_DNSB_INOUTDIRECT—If true, indicates only the top level group was seached for a member.
- DNS$V_DNSB_OUTLINKED—If set, indicates that one or more soft links were encountered while resolving the object of the call.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

Asynchronous system trap (AST) routine to be executed when I/O completes. The **astadr** argument, which is the address of a longword value, is the entry mask to the AST routine.

The AST routine executes in the access mode of the caller of $DNS.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Asynchronous system trap (AST) parameter passed to the AST service routine. The **astprm** argument is a longword value containing the AST parameter.

description

The VMS Distributed Name Service Clerk system service provides a low-level interface between an application (client) and the VAX Distributed Name Service. The DNS clerk interface is used to create, delete, modify, and retrieve objects or soft links in a namespace.

A single system service call supports the DNS clerk. It has two main parameters:

- A function code identifying the particular service to perform
- An item list specifying all the parameters for the required function

The use of this item list is similar to that of other system services that use a single item list for both input and output operations.

Item list entries must be specified in opaque format. You can convert any one of the name strings to opaque format with a conversion function. If applications need to store names, they must store them in opaque format. The opaque format guarantees the uniqueness of a name over time, whereas a string format does not.

Many of the functions return results as a set. In some cases, the specified output buffer might not be large enough to contain the complete set. In this case, the return status indicates this condition with the success status $DNS_MOREDATA. To obtain the remaining data from the set, the client should make repeated calls, each time specifying the last attribute received in the context variable item until the call returns SS$_NORMAL.

The context variable item can take one of two forms depending on the function:

- DNS$CONTEXTVARNAME—If the returned data is a set of names, then the item is a simple name.
- DNS$CONTEXTVARTIME—If the returned data is a set of values, then the item is a time-stamping.

If the context variable item is not specified or is null, then the results are returned from the beginning of the set.

All functions return the SS$_NORMAL status for success except DNS$_TEST_ATTRIBUTE, which returns DNS$_TRUE or DNS$_FALSE. The functions return linked information in the $DNS status block. The DNS$V_DNSB_OUTLINKED bit in the status block indicates whether any soft links are encountered in an information search.

Condition Values Returned

SS$_NORMAL	Normal completion of the request.
DNS$_ACCESSDENIED	Caller does not have required access to the entry in question. This error is returned only if the client has some access to the entry. Otherwise, the unknown entry status is returned.
DNS$_BADCLOCK	The clock at the name server has a value outside the permissible range.
DNS$_BADEPOCH	Copies of directories are not synchronized.
DNS$_BADITEMBUFFER	Invalid output item buffer detected. (This normally indicates that the buffer has been modified during the call.)
DNS$_CACHELOCKED	Global client cache locked.
DNS$_CLEARINGHOUSEDOWN	Clearinghouse is not available.
DNS$_CLERKBUG	Internal clerk error detected.
DNS$_CONFLICTINGARGUMENTS	Two or more optional arguments conflict; they cannot be specified in the same function call.
DNS$_DANGLINGLINK	Soft link points to nonexistent entry.
DNS$_DATACORRUPTION	An error occurred in accessing the data stored at a clearinghouse. The clearinghouse may be corrupted.
DNS$_ENTRYEXISTS	An entry with the same full name already exists in the namespace.
DNS$_FALSE	Unsuccessful test operation.
DNS$_INVALIDARGUMENT	A syntactically incorrect, out of range, or otherwise inappropriate argument was specified in the call.
DNS$_INVALID_ATTRIBUTENAME	The name given for function is not a valid DNS attribute name.
DNS$_INVALID_CLASSNAME	The name given for function is not a valid DNS class name.
DNS$_INVALID_CLEARINGHOUSENAME	The name given for function is not a valid DNS clearinghouse name.
DNS$_INVALID_CONTEXTNAME	The name given for function is not a valid DNS name.
DNS$_INVALID_DIRECTORYNAME	The name given for function is not a valid DNS directory name.

DNS$_INVALID_ENTRYNAME	The name given for function is not a valid DNS entry name.
DNS$_INVALIDFUNCTION	Invalid function specified.
DNS$_INVALID_GROUPNAME	The name given for function is not a valid DNS group name.
DNS$_INVALIDITEM	Invalid item list entry specified.
DNS$_INVALID_LINKNAME	The name given for function is not a valid DNS link name.
DNS$_INVALID_MEMBERNAME	The name given for function is not a valid DNS name.
DNS$_INVALIDNAME	A badly formed name was supplied to the call.
DNS$_INVALID_NSNAME	Namespace name given in name string is not a valid DNS name.
DNS$_INVALID_OBJECTNAME	The name given for function is not a valid DNS object name.
DNS$_INVALID_TARGETNAME	The name given for function is not a valid DNS name.
DNS$_INVALIDUPDATE	An update was attempted to an attribute that cannot be directly modified by the client.
DNS$_INVALID_WILDCARDNAME	The name given for function is not a valid DNS name.
DNS$_LOGICAL_ERROR	Error translating logical name in given string.
DNS$_MISSINGITEM	Required item list entry is missing.
DNS$_MOREDATA	More output data to be returned.
DNS$_NAMESERVERBUG	A name server encountered an implementation bug. Please submit an SPR.
DNS$_NOCACHE	Client cache file not initialized.
DNS$_NOCOMMUNICATION	No communication was possible with any name server capable of processing the request. Check NCP event 353.5 for the DECnet error.
DNS$_NONSRESOURCES	The call could not be performed due to lack of memory or communication resources at the local node to process the request.
DNS$_NONSNAME	Unknown namespace name specified.
DNS$_NOTAGROUP	The full name given is not the name of a group.
DNS$_NOTIMPLEMENTED	This function is defined by the architecture as optional and is not available in this implementation.
DNS$_NOTLINKED	A link is not contained in the name.

DNS$_NOTNAMESERVER	The node contacted by the clerk does not have a DNS server running. This can happen when the application supplies the clerk with inaccurate replica information.
DNS$_NOTSUPPORTED	This version of the architecture does not support the requested function.
DNS$_POSSIBLECYCLE	Loop detected in link or group entry.
DNS$_RESOURCEERROR	Failure to obtain system resource.
DNS$_TIMEOUTNOTDONE	The operation did not complete in the time allotted. No modifications have been performed even if the operation requested them.
DNS$_TIMEOUTMAYBEDONE	The operation did not complete in the time allotted. Modifications may or may not have been made to the namespace.
DNS$_TRUE	Successful test operation.
DNS$_UNKNOWNCLEARINGHOUSE	The clearinghouse does not exist.
DNS$_UNKNOWNENTRY	Either the requested entry does not exist or the client does not have access to the entry.
DNS$_UNTRUSTEDCH	A DNS server is not included in the object's access control set.
DNS$_WRONGATTRIBUTETYPE	The caller specified an attribute type that did not match the actual type of the attribute.

$DNSW—Distributed Name Service Clerk and Wait

The Distributed Name Service Clerk and Wait service registers a resource in a distributed database; same as $DNS. However, the $DNSW service completes synchronously; that is, it returns to the caller after the operation completes.

For asynchronous completion, use the $DNS service, which returns to the caller immediately after making a name service call. The return status to the client call indicates whether a request was successfully queued to the name service.

In all other respects, $DNSW is identical to $DNS. Refer to the $DNS description for complete information about the $DNSW service.

format

SYS$DNSW *[efn] ,func ,itmlst ,[dnsb] ,[astadr] ,[astprm]*

7.3 DNS Run-Time Routines

All applications designed to take advantage of the Distributed Name Service (DNS) use the DNS clerk system services and the DNS run-time routines to register a resource in the DNS namespace and to modify and find information within the namespace. This section describes the run-time routines.

DNS$APPEND_SIMPLENAME_TO_RIGHT—Append a Simple Name to the End of a Full Name

The Append a Simple Name to the End of a Full Name routine adds an opaque simple name to the end of an opaque full name to create a new full name.

format

DNS$APPEND_SIMPLENAME_TO_RIGHT
fullname ,simplename ,resulting-fullname ,resulting-length

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

fullname
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The opaque full name gaining a new simple name. The **fullname** argument is the address of a descriptor pointing to the opaque full name that is to be extended.

simplename
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The opaque simple name that is appended. The **simplename** argument is the address of a descriptor pointing to an opaque simple name that is to be appended to the full name, thus creating a new full name.

resulting-fullname
VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The new full name. The **resulting-fullname** argument is the address of a descriptor that points to the buffer that receives the new full name.

This buffer can be the same as the buffer referred to by the **fullname** argument; however, the descriptors must be separate.

resulting-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the new full name. The **resulting-length** argument is the address of a word that receives the length of the new full name found in **resulting-fullname**.

description

DNS$APPEND_SIMPLENAME_TO_RIGHT adds an opaque simple name to the end of an opaque full name to create a new full name.

condition values returned

SS$_NORMAL	Normal successful completion.
DNS$_INVALIDNAME	The name to be converted is not a valid DNS name.
0	Error appending name.

DNS$COMPARE_FULLNAME—Compare Full Names

The Compare Full Names routine compares two opaque full names and returns the result.

format

DNS$COMPARE_FULLNAME *fullname1 ,fullname2*

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

fullname1

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

One opaque full name. The **fullname1** argument is the address of a descriptor pointing to an opaque full name.

fullname2

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

One opaque full name. The **fullname2** argument is the address of a descriptor pointing to an opaque full name.

description

DNS$COMPARE_FULLNAME compares two opaque full names and returns the result. First, the procedure checks the namespace UIDs of the full names as numbers. If they are unequal, the routine returns the result. If they are equal, it compares each simple name in the full name until it finds an inequality or determines that both names are the same.

condition values returned

–1	**fullname1** is less than **fullname2.**
0	**fullname1** equals **fullname2.**
1	**fullname1** is greater than **fullname2.**

DNS$COMPARE_SIMPLENAME—Compare Two Simple Names

The Compare Two Simple Names routine compares two simple names, without considering case.

format

DNS$COMPARE_SIMPLENAME *simplename1 ,simplename2*

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

simplename1

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

An opaque simple name. The **simplename1** argument is the address of a descriptor pointing to the first simple name.

simplename2

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

An opaque simple name. The **simplename2** argument is the address of a descriptor pointing to the second simple name.

description

DNS$COMPARE_SIMPLENAME compares two simple names, without considering case. The routine determines the relationship between two opaque simple names to see if they are equal.

condition values returned

SS$_NORMAL	Normal successful completion.
–1	**simplename1** is less than **simplename2.**
0	**simplename1** equals **simplename2.**
1	**simplename1** is greater than **simplename2.**

DNS$CONCATENATE_NAME—Join Two Names

The Join Two Names routine joins two opaque full names to form a new full name.

format

DNS$CONCATENATE_NAME
fullname1 ,fullname2 ,resulting-fullname ,resulting-length

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

fullname1
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The opaque full name to be joined. The **fullname1** argument is the address of a descriptor pointing to the opaque full name.

fullname2
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The opaque full name appended to **fullname1**. The **fullname2** argument is the address of a descriptor pointing to the full name to be appended.

resulting-fullname
VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The buffer where the new full name will be written. The **resulting-fullname** argument is the address of a descriptor pointing to the buffer. This buffer can be the same as the buffer referred to by the **fullname1** argument; however, the descriptors must be separate.

resulting-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the new full name. The **resulting-length** argument is the address of a word that receives the length of the new full name found in **resulting-fullname**.

description

DNS$CONCATENATE_NAME joins two opaque full names to form a new opaque full name, which is placed in the buffer named by the **resulting-fullname** argument. The new full name receives the namespace name of the first opaque full name. For example, appending the full name TEST:.POP.DIR1 (**fullname2**) to DEC:.ENG.NAC (**fullname1**) results in a full name of DEC:.ENG.NAC.POP.DIR1.

condition values returned

SS$_NORMAL	Normal successful completion.
DNS$_INVALIDNAME	The name to be converted is not a valid DNS name.
0	Error performing concatenation.

DNS$COUNT_SIMPLENAMES—Count the Simple Names in a Full Name

The Count the Simple Names in a Full Name routine counts the number of simple names contained in an opaque full name.

format

DNS$COUNT_SIMPLENAMES *fullname ,count*

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

fullname

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The full name to be counted. The **fullname** argument is the address of a descriptor pointing to the full name that is to be examined for the simple names it contains.

count

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The number of simple names found in the full name. The **count** argument is the address of a word that receives the number of simple names.

description

DNS$COUNT_SIMPLENAMES counts the number of simple names—but not the namespace name—found in an opaque full name. The number of simple names counted is returned in the word referenced by the **count** argument. The routine is meant to be used with DNS$REMOVE_RIGHT_SIMPLENAME and DNS$REMOVE_LEFT_SIMPLENAME.

condition values returned

SS$_NORMAL	Normal successful completion.
DNS$_INVALIDNAME	The name to be converted is not a valid DNS name.

DNS$CVT_DNSADDRESS_TO_BINARY—Convert a DNS Address to a Phase IV Binary Address

The Convert a DNS Address to a Phase IV Binary Address routine takes a DNS address and returns the DECnet Phase IV node address.

format

DNS$CVT_DNSADDRESS_TO_BINARY *dnsaddress ,binary*

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

dnsaddress

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The DNS address. The **dnsaddress** argument is the address of a descriptor pointing to the DNS address.

binary

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The DECnet Phase IV address found in the DNS address structure. The **binary** argument is the address of a word containing the 16-bit Phase IV address of the node.

description

DNS$CVT_DNSADDRESS_TO_BINARY takes a DNS address and returns the DECnet Phase IV node address. The Phase IV address is returned in a word. If no Phase IV address is found in the DNS address, then the value 0 is returned as an error.

condition values returned

SS$_NORMAL	Normal successful completion.
0	No DECnet Phase IV address found.

DNS$CVT_DNSADDRESS_TO_NODENAME—Convert a DNS Address to a Node Name

The Convert a DNS Address to a Node Name routine takes a DNS address and returns a DECnet Phase IV node name.

format

DNS$CVT_DNSADDRESS_TO_NODENAME
dnsaddress ,nodename ,resulting-length

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

dnsaddress

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The DNS address. The **dnsaddress** argument is the address of a descriptor pointing to the DNS address.

nodename

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The DECnet Phase IV node name. The **nodename** argument is the address of a descriptor pointing to the Phase IV node name. The memory buffer referenced by the DSC$A_POINTER portion of this descriptor must be large enough to contain the entire Phase IV node name string, which can be up to six bytes long.

resulting-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the node name (in bytes) after conversion. The **resulting-length** argument is a word containing the length of the node name (in bytes) after conversion.

description

DNS$CVT_DNSADDRESS_TO_NODENAME takes a DNS address and returns a DECnet Phase IV node name. If no Phase IV address is found, then the value 0 is returned.

Because DNS$CVT_DNSADDRESS_TO_NODENAME calls both $ASSIGN and $QIOW, it can return condition values from either of these system services. The routine also returns errors detected through NETACP.

condition values returned

SS$_NORMAL	Normal successful completion.
0	No DECnet Phase IV address found.

DNS$CVT_NODENAME_TO_DNSADDRESS—Convert a Node Name to an Address

The Convert a Node Name to a DNS Address routine takes a DECnet Phase IV node name and returns a DNS address.

format

DNS$CVT_NODENAME_TO_DNSADDRESS
nodename ,dnsaddress ,resulting-length

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

nodename
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The DECnet Phase IV node name. The **nodename** argument is the address of a descriptor pointing to the node name. This routine creates a DNS address containing the node address of the given Phase IV node name.

dnsaddress
VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The address of a buffer containing the DNS address. The **dnsaddress** argument is the address of a descriptor pointing to the buffer address.

resulting-length
VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the DNS address after conversion. The **resulting-length** argument is a word containing the length of the address.

description

DNS$CVT_NODENAME_TO_DNSADDRESS takes a DECnet Phase IV node name and returns a DNS address. The routine creates the DNS address for a given Phase IV node name.

DNS$CVT_NODENAME_TO_DNSADDRESS calls $ASSIGN and $QIOW so it can return condition values from either of these system services. The routine also returns errors detected through NETACP.

condition values returned

SS$_NORMAL	Normal successful completion.

DNS$CVT_TO_USERNAME_STRING—Convert an Opaque User Name to a String

The Convert an Opaque User Name to a String routine converts an opaque DECnet Phase IV user name into a username string.

format

DNS$CVT_TO_USERNAME_STRING
fullname ,username ,resulting-length

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

fullname
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The opaque full name for the DECnet Phase IV user name. The **fullname** argument is the address of a descriptor pointing to the name.

username
VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The name converted to DECnet Phase IV format (node::user). The **username** argument is the address of a descriptor pointing to a buffer containing the converted name.

resulting-length
VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the converted user name. The **resulting-length** argument is the address of a word containing the length.

description

DNS$CVT_TO_USERNAME_STRING converts a DNS representation of a Phase IV user name into a Phase IV username string.

If any full name other than a DNS representation of a Phase IV user name is given, the routine returns a DNS$_INVALIDNAME error.

condition values returned

SS$_NORMAL	Procedure successfully completed.
DNS$_ACCESSVIOLATION	Memory or other resource access violation.
DNS$_CACHELOCKED	Global client cache locked by another process.
DNS$_INVALIDARGUMENT	One of the arguments was incorrect, out of range, or otherwise inappropriate.
DNS$_INVALIDNAME	The name to be converted is not a valid DNS name.
DNS$_NOCACHE	Client cache file not initialized.
DNS$_RESOURCEERROR	Insufficient resources on local system to process request.

DNS$PARSE_USERNAME_STRING—Convert a User Name from String to Opaque

The Convert a User Name from String to Opaque routine converts a DECnet Phase IV user name to an opaque full name.

format

DNS$PARSE_USERNAME_STRING
user-string ,phase4-name ,resulting-length [,next-character-pointer]

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

user-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The name string to convert. The **user-string** argument is the address of a descriptor pointing to the DECnet Phase IV username string, which is in the format *node::user*.

phase4-name

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The opaque full name resulting from conversion. The **phase4-name** argument is the address of a descriptor pointing to the buffer that is to contain an opaque full name representing a user name on a Phase IV node.

resulting-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the opaque full name. The **resulting-length** argument is the address of a word holding the length of the name returned in **phase4-name**.

next-character-pointer
VMS Usage: **address**
type: **address**
access: **write only**
mechanism: **by reference**

The character following the DNS name extracted from **user-string**. The **next-character-pointer** argument is the address of the character following the DNS name. When you use this argument, DNS$PARSE_USERNAME_STRING returns DNS$_INVALIDNAME when it encounters an invalid name. In such a case, **next-character-pointer** points to the first character in the name that is invalid.

description

DNS$PARSE_USERNAME_STRING converts a DECnet Phase IV user name to an opaque full name that represents the user name.

The **next-character-pointer** argument affects how the routine parses the string:

- When **next-character-pointer** is zero or absent, the full name string given in **user-string** must contain valid DNS characters with DNS naming syntax. If any part of the string violates this rule, the routine returns DNS$_INVALIDNAME and the output should not be used.
- When the **next-character-pointer** argument has a nonzero value, the parsing begins at the first character referenced by **user-string** and parsing continues until one of the following occurs:
  - — An invalid DNS character is found.
  - — An exception to DNS syntax rules occurs.
  - — All characters have been parsed.

  Then the address given by **next-character-pointer** is set to the address of the character or group of characters that is invalid. It returns DNS$_INVALIDNAME if the colons (::) separating the node name from the user name of the Phase IV name are missing.

If any part of the node portion of the DECnet Phase IV username string is not a proper DNS name, the routine returns DNS$_INVALIDNAME regardless of the value and whether or not the **next-character-pointer** argument is supplied.

Error conditions can result from the parse routine. You can test for error conditions in any of the following ways:

- When all parts of the name are invalid, test whether **next-character-pointer** contains the same address as **user-string**. Alternatively, test whether the resulting length is zero.
- When **user-string** contains a valid DNS name, test whether **next-character-pointer** contains the address immediately following the given buffer. Alternatively, test whether the address in **next-character-pointer** minus the address of **user-string** is equal to or larger than the size of the given buffer.
- When parsing a user name that has been extracted from a command line, test whether the character given at the address of **next-character-pointer** is a valid separator for the command line, for example, a space. If you find an invalid character, then part of the DNS name is invalid.

condition values returned

SS$_NORMAL	Normal successful completion.
DNS$_ACCESSVIOLATION	Memory or other resource access violation.
DNS$_CACHELOCKED	Global client cache locked by another process.
DNS$_INVALIDARGUMENT	One of the arguments was incorrect, out of range, or otherwise inappropriate.
DNS$_INVALIDNAME	The name to be converted is not a valid DNS name.
DNS$_NOCACHE	Client cache file not initialized.
DNS$_RESOURCEERROR	Insufficient resources on local system to process request.
0	Error creating opaque name.

DNS$REMOVE_FIRST_SET_VALUE—Remove a Value from a Set

The Remove a Value from a Set routine extracts the first value from a set and returns the value with its creation time-stamping UID.

format

DNS$REMOVE_FIRST_SET_VALUE
set [,value] [,value-length] [,uid] [,uid-length] [,newset] [,newset-length]

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

set
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The set from which the value is extracted. The **set** argument is the address of a descriptor pointing to the set.

value
VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The value extracted from the set. The **value** argument is the address of a descriptor pointing to a buffer containing the value.

value-length
VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the value. The **value-length** argument is the address of a word holding the length of the value placed in **value**.

uid

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The buffer holding the creation time-stamping UID of the extracted value. The **uid** argument is the address of a descriptor pointing to the buffer.

uid-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the UID placed in **uid**. The **uid-length** argument is the address of a word holding the length.

newset

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The opaque **set** without the first value. The **newset** argument is the address of a descriptor pointing to a buffer containing that set. The buffer can be the same as the one given in the **set** argument.

newset-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the new set copied to the **newset** buffer. The **newset-length** argument is the address of a word that receives the length.

description

DNS$REMOVE_FIRST_SET_VALUE extracts a value from a set and returns the value with its creation time-stamping UID. Use the routine to extract values from the sets returned by $DNS and $DNSW.

A set can contain any number of values that are relevant to a given call. The routine extracts values one at a time from the opaque set for use by a program. In order to extract all values from the set, you must use an execution loop.

condition values returned

SS$_NORMAL	Normal successful completion.
DNS$_INVALIDARGUMENT	The set argument was incorrect, out of range, or otherwise inappropriate.
0	Set buffer is empty.
–1	Length of **value**, **uid**, or **newset** buffers too small.

DNS$REMOVE_LEFT_SIMPLENAME—Strip the Simple Name on the Left from the Full Name

The Remove the Simple Name on the Left from the Full Name routine removes the leftmost simple name from an opaque full name. It returns both the simple name stripped and the new full name that results from the operation.

format

DNS$REMOVE_LEFT_SIMPLENAME
fullname [,resulting-fullname] [,resulting-fullname-length]
[,resulting-simplename] [,resulting-simplename-length]

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

fullname

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The opaque full name to strip. The **fullname** argument is the address of a descriptor pointing to the opaque full name to strip. If the full name does not contain any simple names, the routine returns a value of 0 in **cond_value.**

resulting-fullname

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The opaque full name resulting from the operation. The **resulting-fullname** argument is the address of a descriptor pointing to the buffer containing the resulting opaque full name. This buffer can be the same as the buffer referred to by the **fullname** argument; however, the descriptors must be separate.

resulting-fullname-length
VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the full name that is returned. The **resulting-fullname-length** argument is the address of a word receiving the length of the full name returned in **resulting-fullname**.

resulting-simplename
VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The simple name stripped from **fullname**. The **resulting-simplename** argument is the address of a descriptor pointing to the buffer containing the opaque simple name that was stripped.

resulting-simplename-length
VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the simple name. The **resulting-simplename-length** argument is the address of a word that receives the length of the simple name returned in **resulting-simplename**.

description

DNS$REMOVE_LEFT_SIMPLENAME removes the leftmost simple name from an opaque full name. When **resulting-fullname** is nonzero, the full name resulting from the removal of the leftmost simple name is returned in that buffer. When **resulting-simplename** is nonzero, the simple name that was stripped from **fullname** is returned in that buffer. The namespace name is not stripped from the full name; only simple names are affected.

condition values returned

SS$_NORMAL	Normal successful completion.
DNS$_INVALIDNAME	The name to be converted is not a valid DNS name.
–1	Error stripping name.
0	No simple name.

DNS$REMOVE_RIGHT_SIMPLENAME—Strip the Simple Name on the Right from the Full Name

The Remove the Simple Name on the Right from the Full Name routine removes the rightmost simple name from an opaque full name. It returns both the simple name stripped and the new full name that results from the operation.

format

DNS$REMOVE_RIGHT_SIMPLENAME
fullname [,resulting-fullname] [,resulting-fullname-length]
[,resulting-simplename] [,resulting-simplename-length]

returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

arguments

fullname

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The opaque full name to strip. The **fullname** argument is the address of a descriptor pointing to the opaque full name to strip. When the opaque full name does not contain any simple names, the routine returns a value of 0 in **cond_value**.

resulting-fullname

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The opaque full name resulting from the operation. The **resulting-fullname** argument is the address of a descriptor pointing to a buffer containing the resulting opaque full name. This buffer can be the same as the buffer referred to by the **fullname** argument; however, the descriptors must be separate.

resulting-fullname-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the full name returned in **resulting-fullname**. The **resulting-fullname-length** argument is the address of a word that receives the length of the full name returned in **resulting-fullname**.

resulting-simplename

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

A buffer containing the opaque simple name stripped from **fullname**. The **resulting-simplename** argument is the address of a descriptor pointing to the buffer.

resulting-simplename-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

The length of the simple name. The **resulting-simplename-length** argument is the address of a word receiving the length of the simple name returned in **resulting-simplename**.

description

DNS$REMOVE_RIGHT_SIMPLENAME removes the rightmost simple name from an opaque full name. When **resulting-fullname** is nonzero, the full name resulting from the removal of the rightmost simple name is returned in that buffer. When **resulting-simplename** is nonzero, the simple name that was stripped from **fullname** is returned in that buffer. The namespace name is not stripped from the full name; only simple names are affected.

condition values returned

SS$_NORMAL	Normal successful completion.
DNS$_INVALIDNAME	The name to be converted is not a valid DNS name.
–1	Error stripping name.

7.4 Starting the DNS Clerk

The VAX Distributed Name Service (DNS) is a product consisting of two modules: a clerk and a server. The DNS clerk is an integral part of the VMS operating system. The server is a layered product. As long as a DNS server is installed in your network, you can start the DNS clerk on your local VMS system. Then, applications can take advantage of the DNS name service.

You start the DNS clerk once DECnet is running. The DNS startup procedure defines the default DNS server, installs necessary libraries, and creates an advertiser process. Startup involves two steps:

1. Obtain the name of the default DNS server from your network administrator.
2. Execute the command procedure DNS$CHANGE_DEF_FILE. It runs the command procedure DNS$CLERK_STARTUP, which installs the shareable libraries and creates the advertiser process DNS$ADVER.

   To run the command procedure, enter the following command:

   ```
   $ @SYS$STARTUP:DNS$CHANGE_DEF_FILE.COM
   ```

   When executed, SYS$STARTUP:DNS$CHANGE_DEF_FILE.COM prompts for the name of the node where the DNS server is located.

   ```
   Name of DNS server node?
   ```

   Enter a node name, identifying the node that has the DNS server installed.

Once you have run DNS$CHANGE_DEF_FILE.COM, you do not need to run it again unless you want to change the default DNS server or the default namespace. DNS$CHANGE_DEF_FILE.COM copies a configuration file to SYS$SYSTEM that is called DNS$DEFAULT_FILE.DAT. It lists the name of the namespace currently being used as a default.

You must add the file SYS$STARTUP:DNS$CLERK_STARTUP.COM to SYS$MANAGER:SYSTARTUP_V5.COM after the line that starts DECnet so that the DNS clerk images are installed and the advertiser is started after a system boot.

7.5 DECnet Event Messages

The following are DECnet event messages sent by the Distributed Name Service clerk. For a complete list of DECnet event messages, see the *VMS Network Control Program Manual*.

353.5 DNS Clerk Unable to Communicate with Server

The DNS clerk was unable to communicate with a DNS server. This message displays the name of the clearinghouse where the communication failed, the node on which the DNS server servicing the clearinghouse exists, and the reason why the communication failed, which might be any of the following:

- Unknown clearinghouse

  The requested clearinghouse is not serviced by the DNS server that was contacted. This can happen when the cache maintained by the local DNS clerk contains outdated information for a directory.

- Clearinghouse down

  A DNS server is unable to service a request because the clearinghouse is not operational (stopped state).

- Wrong state

  A DNS server is unable to service a request because the clearinghouse is currently starting up or shutting down.

- Data corruption

  A DNS server is unable to service the request because the clearinghouse file has been corrupted.

- No communication

  A network error occurred on the local system or on the system containing the DNS server. The local VMS error is displayed as a part of this message.

353.20 Local DNS Advertiser Error

This event communicates errors that are local to the DNS Advertiser (DNS$ADVER). All these errors have the prefix ADV and are generated when the DNS Advertiser has encounters an error that prevents proper operation of the process. Exact errors are listed in Section 7.6.

7.6 System Error Messages

The following are the system error messages that can be generated by the Distributed Name Service clerk.

ADVADVCOMP, device error while advertising

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred on the network device while advertising clearinghouse information.

User Action: The device in which the error occurred is included in the error text, along with the VMS status from the device. Take corrective action based on the accompanying VMS error message.

ADVADVERSEND, multicast error while advertising clearinghouse

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process was unable to build an advertisement protocol or send the protocol by means of multicasting.

User Action: This error should be seen only on systems that contain a DNS server. Stop the server and restart it. If the problem persists, contact your Digital Customer Service Group representative.

ADVALLOC, failed to allocate space for cache entry

Facility: DNS, Distributed Name Service Clerk Service

Explanation: There is not enough room in the DNS clerk cache for a namespace nickname. The namespace will not be added to the clerk cache.

User Action: This is an informational message; no overt user action is required.

ADVBADFORMAT, bad advertisement message received from device

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The message that the DNS$ADVER process read from the network device was an invalid clearinghouse advertisement.

User Action: Make sure no other process is using multicast ID numbers 09-00-2B-02-01-00 and 09-00-2B-02-01-01. If not, check that the multicasting device is operating correctly.

ADVBTIM, error converting time string in $BINTIM

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An internal error occurred in the DNS$ADVER process. The image might be corrupt.

User Action: Submit a Software Performance Report (SPR).

ADVCACHEINIT, could not initialize DNS clerk cache

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process was unable to initialize the DNS caches. This message is usually accompanied by a message indicating the reason for the failure.

User Action: Make sure no other DNS$ADVER process is running on the system. Also, make sure that the DNS$CLIENT or DNS$SHARE executables are not installed as sharable images. Take corrective action based on the accompanying VMS error message.

ADVCANTIM, error canceling multicast timer

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process was unable to cancel a local timer (used to determine when a multicast of local DNS server information is performed).

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVCHUID, failed to add clearinghouse to cache

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while trying to add DNS clearinghouse information to the DNS global cache. The actual DNS error that caused this error accompanies this error message. The clearinghouse will not be entered in the global cache.

User Action: This is an informational message; no overt user action is required.

ADVCLILOADFAIL, error while loading client clearinghouses

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while the DNS$ADVER process attempted to load clearinghouse information into the client cache.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVDEASGN, error deassigning device

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process was unable to deassign a device that was no longer needed.

User Action: The device in which the error occurred is included in the error message, along with the VMS status from the device. Take corrective action based on the accompanying VMS error message.

ADVDEFCONN, error connecting DNS$DEFAULT_FILE.DAT, no default namespace defined

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process was able to open the configuration file but was unable to connect to the file and read it.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVDEFLOAD, error closing DNS$DEFAULT_FILE.DAT

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process was unable to close the configuration file.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVDEFOPEN, error opening DNS$DEFAULT_FILE.DAT, no default namespace defined

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process was unable to open the file containing configuration information for this system. The DNS clerk can still execute, but names that use default namespaces will be marked invalid.

User Action: Make sure the file DNS$DEFAULT_FILE.DAT exists in the SYS$SYSTEM directory. If it does not, then use the command file SYS$STARTUP:DNS$CHANGE:DEF_FILE.COM to configure default information for the local system. If the file does exist, make sure it has world-readable protection.

ADVDEFPARSE, error parsing DNS$DEFAULT_FILE.DAT, no default namespace defined

Facility: DNS, Distributed Name Service Clerk Service

Explanation: One or more of the entries in the DNS configuration file was invalid. The advertiser was unable to determine the default namespace.

User Action: You might need to edit the configuration file DNS$DEFAULT_FILE.DAT in SYS$SYSTEM so it only contains configuration information for the DNS clerk. You can reconfigure the system using the SYS$STARTUP:DNS$CHANGE_DEF_FILE.COM command file. After making corrections, stop the DNS clerk (using SYS$STARTUP:DNS$CLERK_STOP.COM) and then restart it (using SYS$STARTUP:DNS$CLERK_STARTUP.COM).

ADVDEVCHECK, error checking Ethernet devices

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while checking the status of a network device.

User Action: The device in which the error occurred is included in the error text, along with the VMS status from the device. Take corrective action based on the accompanying VMS error message.

ADVDUMPFAIL, error while dumping DNS server clearinghouses

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while the DNS$ADVER process attempted to create a configuration file on the local system.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVETHERMEM, no memory for write to device

Facility: DNS, Distributed Name Service Clerk Service

Explanation: There is not enough memory available for the DNS$ADVER process to allocate space to build a protocol.

User Action: Check process and system memory quotas.

ADVMBX, could not create mailbox DNS$SOLICIT_MBX

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred on the local system that prevented the DNS$ADVER process from creating the DNS$SOLICIT_MBX mailbox. This mailbox is used by the DNS clerk to log errors.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVNICKUID, failed to add NICKNS entry to cache

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while trying to associate a UID with a namespace nickname in the DNS global cache. The actual DNS error that caused the error is included in the error message. An entry will not be made in the global cache for this namespace.

User Action: This is an informational message; no overt user action is required.

ADVPRIV, not enough privs to run DNS$ADVER

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The process executing the DNS$ADVER process does not have the necessary privileges.

User Action: The DNS$ADVER process needs SHMEM, PFNMAP, EXQUOTA, PRMGBL, SYSGBL, SYSLCK, SYSPRV, LOG_IO, NETMBX, TMPMBX, SYSNAM and CMKRNL privileges. Restart DNS$ADVER using these privileges.

ADVRANDOM, error generating random number

Facility: DNS, Distributed Name Service Clerk Service

Explanation: Random numbers are used by the DNS$ADVER process to help prevent congestion of multicasts on the network.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVRECBUF, device error while receiving message

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process encountered an error while receiving a message through the network.

User Action: The device in which the error occurred is included in the error text, along with the VMS status from the device. Take corrective action based on the accompanying VMS error message.

ADVRECREATE, recreation of mailbox DNS$SOLICIT_MBX failed

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while receiving a message on the mailbox. Another error occurred when the DNS$ADVER process tried to remedy the error by recreating the mailbox.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVREREAD, re-read of DNS$SOLICIT_MBX failed

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while receiving a message on the mailbox. After a new mailbox had been successfully created, another error occurred while trying to reread the mailbox.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVSETIMR, error setting timer for multicasts

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process was unable to set a local timer. This timer is used to determine when a multicast of local DNS server information is performed.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVSOLCOMP, device error while soliciting DNS servers

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process encountered an error while multicasting a solicitation message to DNS servers on the network.

User Action: The device in which the error occurred is included in the error text, along with the VMS status from the device. Take corrective action based on the accompanying VMS error message.

ADVTIMERR, cannot create timeout value

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS$ADVER process attempted to create a DNS timeout value but received an error from the $GETTIM service routine.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

ADVUIDNICK, failed to add NSNAME entry to cache

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while trying to associate a UID with a namespace nickname in the DNS global cache. The actual DNS error that caused this error is included in this error text. No entry will be made in the global cache.

User Action: This is an informational message; no overt user action is required.

ADVWRITETHER, error writing on device

Facility: DNS, Distributed Name Service Clerk Service

Explanation: An error occurred while DNS$ADVER was trying to write to an ethernet device.

User Action: The device in which the error occurred is included in the error text, along with the VMS status from the device. Take corrective action based on the accompanying VMS error message.

NOCOMMUNICATION, unable to communicate with DNS server

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS clerk was unable to communicate with a clearinghouse. The error text includes VMS error information that might have caused the communication problem.

User Action: A specific VMS error immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

SOLCONNECT, error connecting to file SYS$SYSTEM:DNS$DEFAULT_FILE.DAT

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The solicitor attempted to check the configuration file for errors, but was unable to connect to the file.

User Action: A specific VMS error message immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

SOLOPEN, error opening file SYS$SYSTEM:DNS$DEFAULT_FILE.DAT

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The solicitor attempted to check the configuration file for errors but was unable to open the file.

User Action: A specific VMS error message immediately follows the error text of this message. Take corrective action based on the accompanying VMS error message.

SOLREADLINE, error reading line in file SYS$SYSTEM:DNS$DEFAULT_FILE.DAT

Facility: DNS, Distributed Name Service Clerk Service

Explanation: Invalid configuration information was found in the DNS configuration file.

User Action: You might need to edit the configuration file DNS$DEFAULT_FILE.DAT in SYS$SYSTEM. Make sure that the file contains configuration information for the DNS clerk only. You can reconfigure the system using the SYS$STARTUP:DNS$CHANGE_DEF_FILE.COM command file. After making corrections, you must stop the DNS clerk (using SYS$STARTUP:DNS$CLERK_STOP.COM) and restart it (using SYS$STARTUP:DNS$CLERK_STARTUP.COM).

SOLTIMEOUT, timed out waiting for DNS$ADVER to read mailbox message from DNS$SOLICIT_MBX

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS solicit process was unable to communicate with the advertiser process. This can happen when the advertiser process (DNS$ADVER) is unable to start properly or has terminated abnormally.

User Action: Make sure the DNS$ADVER process is running. If it is not, then restart it using the SYS$STARTUP:DNS$CLERK_STARTUP.COM command file. If it is running, stop it using SYS$STARTUP.DNS$CLERK_STOP.COM and then restart it. If this does not work, call your Digital Customer Service Group representative.

SOLWAITING, waiting for DNS$ADVER to read mailbox message from DNS$SOLICIT_MBX

Facility: DNS, Distributed Name Service Clerk Service

Explanation: During the startup phase of the DNS clerk, you get this message if your system is heavily loaded. It indicates that startup has taken longer than normally anticipated but has not yet failed.

User Action: This is an informational message; no overt user action is required.

UNKNOWNCLEARINGHOUSE, the clearinghouse does not exist

Facility: DNS, Distributed Name Service Clerk Service

Explanation: The DNS clerk attempted to contact a clearinghouse that does not exist in the namespace. This can occur when the DNS clerk caches are out of date or when the replica set of a given directory is out of date.

User Action: None. The DNS clerk will try to contact other clearinghouses to satisfy the request.

Chapter 8

VAXTPU

This chapter provides information on new and changed features of VAXTPU for programmers who write applications in VAXTPU or who extend or layer applications on the Extensible VAX Editor (EVE).

The portions of this chapter that cover DECwindows VAXTPU are intended for readers familiar with DECwindows programming concepts and terminology. For more information on programming in DECwindows, see the *VMS DECwindows Guide to Application Programming* and the *VMS DECwindows Toolkit Routines Reference Manual.*

The major new VAXTPU features for VMS Version 5.3 are as follows:

- Initialization is faster in DECwindows VAXTPU. Note that some application recoding might be required.
- The MAP and UNMAP built-ins accept a widget parameter.
- New buffer change journaling keeps track of modifications on a per-buffer basis. Buffer change journaling allows DECwindows VAXTPU sessions to be journaled and recovered. To implement buffer change journaling, VAXTPU provides six new built-ins and enhances four previously existing built-ins. Keystroke journaling is still provided.
- Pattern searches are more efficient. The new keywords BUFFER_BEGIN and BUFFER_END are available for creating patterns. The SCAN, SCANL, SPAN, and SPANL built-ins can perform a new kind of reverse search in addition to the previously supported reverse search.
- Programmers now have the option to make one or more records in a buffer visible or invisible on a screen. The programmer can also set or change the left margin of the records. Finally, the records can be made unmodifiable, so a user cannot alter the record.
- A watch cursor replaces the pointer cursor when DECwindows VAXTPU is busy for more than one second.

- DECwindows VAXTPU supports the use of icon pixmaps and the addition of the SET (HEIGHT) built-in and enhancement of the SET (WIDTH) built-in make window resizing easier.
- Seven new built-ins provide finer control over various DECwindows functions.
- Application programmers or users can change the current default directory from within VAXTPU.
- For defining synonyms, an EQUIVALENCE statement is available and programmers can declare a local variable in code that is not within a procedure. Binary, octal, and hexadecimal constants are supported, as well as decimal constants.
- Conditional statements are now available to control which portions of a source file are compiled under various conditions.
- You can designate a program to handle detached cursor conditions (conditions in which the cursor position cannot accurately represent the editing point in the current window).
- Miscellaneous enhancements have been added to the following previously existing built-ins:
  - CHANGE_CASE
  - CREATE_RANGE
  - EDIT
  - GET_INFO (buffer_variable)
  - LENGTH
  - MESSAGE
  - MODIFY_RANGE
  - POSITION
  - SUBSTR
  - TRANSLATE
- Seven keywords are reserved for future use.
- When using the VAXTPU callable interface, the application programmer can allow VAXTPU to supply a default routine instead of specifying the entry TPU$_FILEIO in the item list for the TPU$INITIALIZE routine.
- The VAXTPU callable interface supports chaining of item lists, making the callable interface more like VMS system services.

- Function keys F1 through F5 are now supported. Modifiers ALT_MODIFIED and HELP_MODIFIED are available for main keyboard keys and for control modified keys.
- All built-ins are available when you specify the /NODISPLAY command qualifier.

These features are described in more detail in the following sections.

8.1 Associated Documents

For complete reference information on the features of VAXTPU through VMS Version 5.2, see the *VAX Text Processing Utility Manual.*

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 Volume* 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.

8.2 Changed DECwindows VAXTPU Initialization

This section discusses new VAXTPU initialization processing features that are incompatible with previous versions of DECwindows VAXTPU.

All code that compiled and executed properly on the versions of VAXTPU released with VMS Version 5.0, Version 5.1, and Version 5.2 will continue to compile and execute properly, unless the code depends on displaying information or prompting during startup of DECwindows VAXTPU. To display information or prompts, you must modify the initialization code to include a statement containing any of the following built-ins:

- READ_KEY built-in procedure
- READ_CHAR built-in procedure
- READ_LINE built-in procedure
- MAP built-in procedure

Also note that the use of new or incompatible EVE features might make rebuilding existing section files desirable.

8.3 Initialization Coding

Unlike previous versions of DECwindows VAXTPU, the new version does not map any prompts or display information to the screen until the initialization program completes. This results in a faster initialization process.

Because user prompts and display information are no longer automatically displayed during the application's initialization phase, your program code for previous versions of VAXTPU might not work as intended under the new version. The READ_CHAR, READ_KEY, and READ_LINE built-ins have been enhanced to map the user visible portion of your application to the DECwindows screen. For information about resizing windows/screens and the READ_CHAR, READ_KEY, and READ_LINE built-in procedures, refer to Section 8.7.2.

If you want an application to display information during initialization but do not want to use READ_CHAR, READ_KEY, or READ_LINE, you can use the newly enhanced MAP built-in. MAP allows a VAXTPU application to map the window associated with a widget to the screen. The UNMAP built-in has been similarly enhanced to allow an application to unmap the window associated with a widget.

To map or unmap VAXTPU's top-level window and all its subwindows, your application can call GET_INFO (widget, "parent") in a loop. The loop starts at some known child of the VAXTPU top-level widget and proceeds up the widget hierarchy until GET_INFO (widget, "parent") returns 0, indicating that a widget with no parent has been found. The application then calls MAP or UNMAP on the parentless widget. The application can use the widget returned by GET_INFO (WIDGET, "widget_id", SCREEN, "tpu$mainwindow") as the starting point in such a loop.

8.3.1 Enhancements to the MAP Built-In

The MAP built-in can now take a widget parameter. When you use MAP this way, the built-in calls the Xlib routine MAP WINDOW to map the window associated with the specified widget.

Syntax

MAP (widget)

Parameter

widget

The widget instance you want to make visible.

Example

```
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 SET (MAPPED_WHEN_MANAGED) in this chapter.

8.3.2 Enhancements to the UNMAP Built-In

The UNMAP built-in can now take a parameter that is a variable containing a widget instance. When you use UNMAP this way, the built-in calls the Xlib routine UNMAP WINDOW to unmap the window associated with the specified widget. If the unmapped window is VAXTPU's top-level window, VAXTPU automatically maps the top-level window again if a READ_CHAR, READ_KEY or READ_LINE built-in is executed.

Syntax

UNMAP (widget)

Parameter

widget
The widget instance you want to make invisible.

Example

```
UNMAP (example_widget);
```

This statement causes the widget assigned to the variable *example_widget* to become invisible.

8.3.3 Behavior of GET_INFO (widget_variable, "widget_info") Built-In

The GET_INFO (widget_variable, "widget_info") built-in has been modified to handle the case where the requested resource is a list of items and the list contains no entries. In this case, the GET_INFO call uses either the element of the array parameter or uses the value parameter to return an array containing no elements.

Syntax

```
{0 | 1} := GET_INFO (widget_variable, "widget_info",
                     {array | arg_pair} [, array | arg_pair...])
```

Parameters

widget_variable
The variable containing the widget instance whose resource values you want to fetch.

"widget_info"
A string indicating that you want the current value for one or more resources of the specified widget.

array
An array used to return resource values. For complete information on using this array, see the description of GET_INFO (widget_variable) in the *VAX Text Processing Utility Manual.*

arg_pair
A string naming a valid resource for the specified widget followed by a variable to store the value of the resource. For complete information on using this pair of parameters, see the description of GET_INFO (widget_variable) in the *VAX Text Processing Utility Manual.*

Example

```
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.

8.4 Buffer Change Journaling

In previous versions of VAXTPU, the only form of journaling available was keystroke journaling. In keystroke journaling, VAXTPU keeps track of each keystroke made by the user during a session, regardless of which buffer is in use when the user presses the key. If a system interruption occurs during a session, the user can use the /JOURNAL and /RECOVER qualifiers to reconstruct the work done during the session. For more information on recovery using a keystroke journal file, see the *VMS EVE Reference Manual.*

Keystroke journaling did not work in the DECwindows environment in previous versions of VAXTPU, which made journaling and recovery impossible.

In VMS Version 5.3, VAXTPU supports both keystroke journaling and buffer change journaling. Buffer change journaling involves maintaining journal files on a per-buffer basis. The application can use the enhanced SET (JOURNALING) built-in to direct VAXTPU to establish and maintain a separate journal file for any buffer or buffers created during the session. The application programmer or user can also use the enhanced SET (JOURNALING) built-in to turn buffer

change journaling off or on for a given buffer during a session. For more information on SET (JOURNALING), see Section 8.4.10. For more information on using buffer change journaling files to recover from a system interrupt, see Section 8.4.9.

In a buffer's journal file, VAXTPU keeps track of the following attributes for each record in the buffer:

- Left margin setting
- Modifiability or unmodifiability
- Display value

For more information on record attributes and display values, see Section 8.6.

In the buffer's journal file, VAXTPU also keeps track of the following changes to records and record attributes:

- Characters inserted in and deleted from a record (including the location where the change took place)
- Records inserted in and deleted from a buffer (including the location where the change took place)
- Changes to record attributes

Buffer change journaling does not involve keeping a record of all keystrokes performed by the user while the user is editing a given buffer.

Now you can use both keystroke and buffer change journaling at the same time. To turn on keystroke journaling, the application uses the JOURNAL_OPEN built-in. For more information on JOURNAL_OPEN, see the *VAX Text Processing Utility Manual.*

The application layered on VAXTPU, not the VAXTPU engine, determines what kind of journaling is turned on and under what conditions. Table 8–1 shows the journaling behavior established by EVE, which is VAXTPU's default editor.

Table 8–1: Journaling Behavior Established by EVE

User or Programmer Action	Effect on Keystroke Journaling	Effect on Buffer Change Journaling
No DCL command qualifier related to journaling is specified.	Keystroke journaling is turned off.	Buffer change journaling is turned on.
/NOJOURNAL command qualifier is specified.	Keystroke journaling is turned off.	Buffer change journaling is turned off. Note, however, that /NOJOURNAL does not disable buffer change journaling. Even when /NOJOURNAL has been specified, it is possible to use SET (JOURNALING) to turn on buffer change journaling.
/JOURNAL = <filespec> command qualifier is specified.	Keystroke journaling is turned on.	Buffer change journaling is turned off.
/JOURNAL command qualifier is specified without a file specification.	Keystroke journaling is turned off.	Buffer change journaling is turned on.

To determine whether buffer change journaling is turned on, use a statement similar to the following:

```
status := GET_INFO (buffer_name, "journaling");
```

To determine the name of the keystroke journal file, use a statement similar to the following:

```
filename := GET_INFO (SYSTEM, "journal_file");
```

8.4.1 Buffer Change Journal File Naming Algorithm

By default, VAXTPU creates the buffer change journal file name by using the following algorithm:

1. Convert all characters in the buffer name that are not alphanumeric, dollar sign, underscore, or hyphen to underscores.
2. Truncate the resulting file name to 39 characters.
3. Add the file extension .TPU$JOURNAL.

For example, a buffer named FOO.BAR has a default journal file name of FOO_BAR.TPU$JOURNAL.

VAXTPU puts all journal files in the directory defined by the logical name TPU$JOURNAL. By default, this logical is defined as SYS$SCRATCH. You can reassign this logical name. For example, if you want journal files written to the current default directory, define TPU$JOURNAL as [].

8.4.2 Enhancements to the CREATE_BUFFER Built-In

The CREATE_BUFFER built-in now optionally accepts a fourth parameter specifying the name of the journal file to be used with the buffer.

Syntax

[buffer :=] CREATE_BUFFER (string1 [, [string2] [, [buffer] [, string3]]])

Parameters

string1
The name of the buffer you want to create.

string2
Optionally, a string specifying the input file for the buffer. If you do not specify an input file, you create an empty buffer.

buffer
The buffer you want to use as a template for the buffer being created. The new buffer has the same attributes (such as tabs, margins, etc.) as the template buffer. For a list of all the attributes inherited by the new buffer, see the description of the CREATE_BUFFER built-in in the *VAX Text Processing Utility Manual*.

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 default journal file naming algorithm, see Section 8.4.1.

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.

Description

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.

Examples

```
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.

```
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.

8.4.3 Enhancements to the DELETE Built-In

When a buffer is deleted, the associated journal file (if any) is closed and deleted.

8.4.4 GET_INFO (buffer_variable, "journaling") Built-In

The new GET_INFO (buffer_variable, "journaling") built-in returns 1 if the specified buffer is being journaled or returns 0 if it is not.

Syntax

{1 | 0} := GET_INFO (buffer_variable, "journaling")

Parameters

buffer_variable
The variable containing the buffer whose journaled status you want to know.

"journaling"
A string directing VAXTPU to return the journaled status of the specified buffer.

Example

```
the_result := GET_INFO (main_buffer, "journaling");
```

This statement returns 1 if journaling is turned on for the buffer stored in the variable *main_buffer* or returns 0 if journaling is turned off.

8.4.5 GET_INFO (buffer_variable, "journal_file") Built-In

The new GET_INFO (buffer_variable, "journal_file") built-in 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.

Syntax

{file_name_string | 0} := GET_INFO (buffer_variable, "journal_file")

Parameters

buffer_variable
The variable containing the buffer whose journal file name you want to know.

"safe_for_journaling"
A string directing VAXTPU to return the name of the specified buffer's journal file.

Example

```
the_name := GET_INFO (main_buffer, "journal_file");
```

This statement assigns to the variable *the_name* the name of the journal file for the buffer stored in the variable *main_buffer* or returns 0 if journaling is turned off.

8.4.6 GET_INFO (buffer_variable, "safe_for_journaling") Built-In

The new GET_INFO (buffer_variable, "safe_for_journaling") built-in returns 1 if the specified buffer is safe for journaling or returns 0 if it is not. 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.

Syntax

{1 | 0} := GET_INFO (buffer_variable, "safe_for_journaling")

Parameters

buffer_variable
The variable containing the buffer whose safety status you want to know.

"safe_for_journaling"
A string directing VAXTPU to return the safety status of the specified buffer.

Example

```
the_result := GET_INFO (main_buffer, "safe_for_journaling");
```

This statement returns 1 if journaling is safe on for the buffer stored in the variable *main_buffer* or returns 0 if not.

8.4.7 GET_INFO (buffer_variable, "journal_name") Built-In

The new GET_INFO (buffer_variable, "journal_name") built-in converts a buffer's name to a journal file name using VAXTPU's 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 8.4.1.

Syntax

file_name_string := GET_INFO (buffer_variable, "journal_name")

Parameters

buffer_variable
The variable containing the buffer whose name you want converted to a journal file name.

"journal_name"
A string directing VAXTPU to convert the specified buffer's name to a journal file name.

Example

```
the_name := GET_INFO (main_buffer, "journal_name");
```

This statement assigns to the variable *the_name* a journal file name created by applying the journal file naming algorithm to the name of the buffer contained in the variable *main_buffer*.

8.4.8 GET_INFO (string_variable, "journal") Built-In

The new GET_INFO (string_variable, "journal") built-in returns an array containing information about the journal file whose name you specify with the string parameter. If the specified file is not a journal file, the integer 0 is returned.

Syntax

{array | 0} := GET_INFO (string_variable, "journal")

Parameters

string_variable
A string that is the name of the journal file about which you want information.

"journal"
A string constant directing VAXTPU to return an array containing information on the specified journal file.

Description

The array indexes 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	VAXTPU's identification string for the version of VAXTPU that wrote the journal file.

Note that all elements are of type string.

Example

```
the_array := GET_INFO ("foo_bar.tpu$journal", "journal");
```

This statement returns an array whose elements contain strings that are attributes of the journal FOO_BAR.TPU$JOURNAL.

8.4.9 RECOVER_BUFFER Built-In

The new RECOVER_BUFFER built-in 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.

Syntax

new_buffer := RECOVER_BUFFER (old_buffer_name, [, [journal_file_name] template_buffer]])

Parameters

old_buffer_name

The name of the buffer you are trying to recover.

journal_file_name

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.

template_buffer

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 in the *VAX Text Processing Utility Manual* of the CREATE_BUFFER built-in.

Description

Do not confuse the RECOVER_BUFFER built-in with the /RECOVER command qualifier in DCL. /RECOVER is used with /JOURNAL when invoking VAXTPU to recover a session by using a keystroke journal file. 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.

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 records from the buffer were never written to the journal file because there were not enough records 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 operations 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 in the *VAX Text Processing Utility Manual.*

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 aborts the recovery, closes the journal file, and deletes the newly created buffer.

After a successful recovery, VAXTPU continues journaling new changes into the journal file that was used during the recovery.

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 8.4.1. For example, suppose you were editing two buffers, one called FOO! and the other called FOO?. The default journal file naming algorithm creates for each buffer a journal file named FOO_ .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 FOO! had the lower version number, then RECOVER_BUFFER would recover the journal file created for the buffer FOO?.

When you write the contents of a buffer to a file, the old journal file is closed, a new journal file is created, and the old journal file is deleted. If you write the contents of the buffer to a file other than the default output file, the new 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 new 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.

Examples

```
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.

```
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 calling the contents of 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!]".

8.4.10 SET (JOURNALING) Built-In

The SET (JOURNALING) built-in can now be used to turn on or turn off buffer change journaling and to specify a journal file name.

Syntax

SET (JOURNALING, buffer, {ON | OFF} [,file_name_string])

Parameters

JOURNALING
A keyword indicating that the SET built-in is being used to enable, disable, or set the frequency of journaling.

buffer
The buffer for which you want to turn on 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 8.4.1.

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, never modified, or 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 one is started.

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 8.4.1.

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.

Example

```
SET (JOURNALING, CURRENT_BUFFER, ON, "disk1:[reinig]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 [REINIG] as the journal file.

8.4.11 WRITE_FILE Built-In

The WRITE_FILE built-in has been enhanced. 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.

8.4.12 TPU$_FILE_RECOVERABLE Item Code

The VAXTPU callable interface routine TPU$INITIALIZE now makes available a new item code, TPU$_FILE_RECOVERABLE. (For more information on the VAXTPU callable interface and the TPU$INITIALIZE routine, see the *VMS Utility Routines Manual*.)

This item code indicates whether the file being opened will be available when VAXTPU attempts a recovery. A file might not be available if the layered application does not automatically create and continuously update a file associated with each buffer created during the session. For example, if the file being opened is a mailbox (as opposed to a file on disk), then the mailbox will probably not exist when the user attempts to recover the file.

By default, the item code contains a 0, indicating that the file will not be available at recovery time.

If this item code is set to 0, VAXTPU copies the contents of the buffer into the journal file when the journal file is created.

8.5 Enhancements to VAXTPU's Pattern Support

The enhancements to VAXTPU's pattern support are as follows:

- The new keywords BUFFER_BEGIN and BUFFER_END can now be used to construct patterns.
- Searches using sophisticated patterns are executed more efficiently.
- The SCAN, SCANL, SPAN, and SPANL built-ins can now be used to specify a new kind of reverse search.

8.5.1 New Pattern Keywords

BUFFER_BEGIN and BUFFER_END are new keywords within a pattern. BUFFER_BEGIN matches the beginning of the buffer in which the search is executed. BUFFER_END matches the end of the buffer in which the search is executed.

8.5.2 Search Performance

Searching with complicated patterns is now more efficient in three areas.

When performing a non-exact search, VAXTPU must translate both the pattern and the searched text to ensure correct comparisons. VAXTPU now allocates space within each pattern element to store the translated version of the element. This means that VAXTPU no longer translates the pattern each time the SEARCH built-in is invoked. The pattern is only translated during the first non-exact SEARCH using that pattern.

The pattern code now translates whole records at a time and saves the last record it translated. Thus, after the SEARCH built-in has checked the first half of a line, it does not have to translate the line again to see if the rest of the pattern matches the second half of a line.

Alternation previously matched both alternate pattern elements and then chose which alternate to use for the pattern match. Now the alternation algorithm does not try to match the second alternate pattern element anyplace at or after the start of the first alternate pattern element. In reverse search, it does not try to match the second alternate at or before the start of the first alternate pattern element. For example, suppose you search this paragraph with the following pattern:

```
"pattern" | "xyzzy"
```

The search does not proceed past the first line because the first line contains the first alternate, which is *pattern*.

8.5.3 The New Reverse Search Algorithm

A new optional parameter is allowed for the SCAN, SPAN, SCANL, and SPANL built-ins. This parameter can be the either of the keywords REVERSE or FORWARD. The REVERSE keyword specifies new behavior in reverse searches.

For more information on the SCAN, SCANL, SPAN, and SPANL built-ins, see the descriptions of those built-ins in the *VAX Text Processing Utility Manual*.

Syntax

```
                {SCAN   |
pattern :=      SCANL   |   ({string | range | buffer} [, {FORWARD | REVERSE}])
                SPAN    |
                SPANL   }
```

Parameters

string
The string on which you want the built-in to perform pattern matching.

range
The range on which you want the built-in to perform pattern matching.

buffer
The buffer on which you want the built-in to perform pattern matching.

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 not member of the set of characters in the specified buffer, range, or string (in the case of SPAN and SPANL) or that is a member of the set of characters (in the case of SCAN and SCANL). 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 (in the case of SPAN and SPANL) or that is in the set (in the case of SCAN and SCANL). You can specify REVERSE only if you are using the built-in 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.

Example

```
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.

8.6 Record Attributes

Previous versions of VAXTPU supported the setting and modifying of one record attribute, the left margin. A record attribute is a characteristic of a record or of an integer value associated with a record, defining how the record appears or behaves and what actions can be performed on it.

VAXTPU supports two new record attributes, display value and modifiability. For more information on the display value attribute, see Section 8.6.1. For more information on the modifiability attribute, see Section 8.6.2.

VAXTPU provides a new built-in, SET (RECORD_ATTRIBUTE), for setting or changing record attributes. (For more information on SET (RECORD_ ATTRIBUTE), see Section 8.6.3.6.) Copying or moving a record in insert mode generally preserves all its attributes. VAXTPU does not preserve record attributes in overstrike mode. Splitting a record creates two records with the same record attributes, except that the new record receives the buffer's default left margin value, not the original record's left margin value. If you append to one record another record with different attributes, the resulting record has the attributes of the record to which the second record was appended.

For example, copying an unmodifiable record in insert mode with MOVE_TEXT or COPY_TEXT usually creates a new record that is unmodifiable. Note, however, that the first or last line created during the copy of a range may or may not be unmodifiable.

The major exception to the general rule stated in the previous paragraph is that splitting a line at the beginning or end of a record creates a new record. The new record is created with the default values of the various record attributes, no matter what the values of the *split* line.

8.6.1 Display Value Attribute

A display value is an integer that controls whether a given record is visible in a given window. You can assign a display value to a record using the new SET (RECORD_ATTRIBUTE) built-in. For more information on this built-in, see Section 8.6.3.6. You can assign a display value to a window using the new SET (DISPLAY_VALUE) built-in. For more information on this built-in, see Section 8.6.3.4.

If a record's display value is greater than or equal to the display value of the window mapped to the record's buffer, VAXTPU makes the record visible in that window; otherwise, VAXTPU makes the record invisible.

Display values can range from -127 to +127. The default value is zero for both records and windows, meaning that records are visible by default.

The display value of a buffer's end of buffer text cannot be changed. The end of buffer text is permanently assigned a display value of +127 and, thus, is always visible in any window. You can, without error, include the end of buffer text in a range when you set the display value of a set of records. However, this does not change the display value of the end of buffer text.

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." To designate code to be executed when the cursor is detached, use the new SET (DETACHED_ACTION) built-in. For more information on this built-in, see Section 8.11.1.

Note that the built-ins SCROLL, CURSOR_VERTICAL and CURSOR_HORIZONTAL always leave the editing point and the cursor position on a visible record. Thus, SCROLL (CURRENT_WINDOW, 0), CURSOR_VERTICAL (0), and CURSOR_HORIZONTAL (0) all have the effect of moving the editing point from an invisible record to the next visible record in the current window.

8.6.2 Modifiability Attribute

Now you can use the new SET (RECORD_ATTRIBUTE) built-in to make an individual record unmodifiable within a buffer, regardless of whether the buffer is unmodifiable. For more information on SET (RECORD_ATTRIBUTE), see Section 8.6.3.6.

If a buffer is unmodifiable, then no records within it can be modified. However, making a buffer modifiable does not automatically make all records within a buffer modifiable. Any record that was individually made unmodifiable remains so.

By default, any newly created records in a modifiable buffer are modifiable.

You cannot change the left margin of an unmodifiable record. You can change the display value of a record at any time.

8.6.3 New Built-Ins Implementing Record Attribute Support

VAXTPU has the following new built-in procedures to implement support for setting, changing, and determining record attributes:

- GET_INFO (buffer, "erase_unmodifiable")
- GET_INFO ({buffer | marker | range}, "unmodifiable_records")
- GET_INFO ({mark | window}, "display_value")
- SET (DISPLAY_VALUE)
- SET (ERASE_UNMODIFIABLE)
- SET (RECORD_ATTRIBUTE)

These built-ins are described in the following subsections.

8.6.3.1 GET_INFO (buffer_variable, "erase_unmodifiable") Built-In

The new GET_INFO (buffer_variable, "erase_unmodifiable") built-in returns 1 if unmodifiable records can be erased from the specified buffer and returns 0 if the records cannot be erased. For more information on enabling and disabling the erasing of unmodifiable records, see Section 8.6.3.5.

Syntax

```
status := GET_INFO (buffer, "erase_unmodifiable")
```

Parameters

buffer
The buffer holding the records of where you are fetching the erase-unmodifiable record status.

"erase_unmodifiable"
A string constant indicating that you want to know if unmodifiable records can be erased from the specified buffer.

Example

```
the_result := GET_INFO (CURRENT_BUFFER, "erase_unmodifiable");
```

This statement assigns the value 1 to the variable *the_result* if the records in the current buffer can be erased. Otherwise, the value 0 is returned.

8.6.3.2 GET_INFO ({buffer | marker | range}, "unmodifiable_records") Built-In

The new GET_INFO ({buffer | marker | range}, "unmodifiable_records") built-in returns 1 if the specified buffer or range contains one or more unmodifiable records or if the record containing the specified marker is unmodifiable. The call returns 0 if no unmodifiable records are present in the specified location.

Syntax

status := GET_INFO ({buffer | marker | range}, "unmodifiable_records")

Parameters

buffer
The buffer in which you want to determine whether unmodifiable records are present.

marker
The marker marking the line whose unmodifiability you want to determine.

range
The buffer in which you want to determine whether unmodifiable records are present.

"unmodifiable_records"
A string constant indicating that you want to know whether unmodifiable records are present.

Example

```
the_marker := MARK (none);
the_result := GET_INFO (the_marker, "unmodifiable_records");
```

This code fragment establishes a marker at the editing point and then determines whether the record containing the marker is unmodifiable.

8.6.3.3 GET_INFO ({marker | window}, "display_value") Built-In

The new GET_INFO ({marker | window}, "display_value") built-in returns the display value of the specified window or of the record in which the specified marker is located. For more information on how display values are used, see Section 8.6.1.

Syntax

display_value_integer := GET_INFO ({marker | window}, "display_value")

Parameter

marker
The marker marking the record whose display value you want to know.

window
The window whose display value you want to know.

"display_value"
A string constant indicating that you want to fetch the display value associated with the specified window or with the record containing the specified marker.

Example

```
the_value := GET_INFO (CURRENT_WINDOW, "display_value");
```

This statement assigns to the variable *the_value* the display value associated with the current window.

8.6.3.4 SET (DISPLAY_VALUE) Built-In

The new SET (DISPLAY_VALUE) built-in sets the display value of the specified window. For information on setting the display value of a record, see Section 8.6.3.6.

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.

Syntax

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.

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.

8.6.3.5 SET (ERASE_UNMODIFIABLE) Built-In

The new SET (ERASE_UNMODIFIABLE) built-in controls whether VAXTPU erases unmodifiable records in response to built-ins that delete lines from a buffer. 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 if a similar built-in encounters an unmodifiable record, the built-in returns an error and does not delete the record.

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.

Syntax

[previous_erase_setting] := 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.

Description

By default, unmodifiable records can be deleted from buffers by built-ins such as ERASE_LINE. However, some built-ins delete records as a side effect of their normal action. Table 8–2 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 8–2: 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 and the buffer is not erase 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 and the buffer is not erase unmodifiable.
ERASE_CHARACTER	Signals a warning if the current line is unmodifiable.
ERASE_LINE	Signals a warning if the current line is unmodifiable and the buffer is not erase unmodifiable.
FILL	Signals a warning if any of the lines in the range or buffer are unmodifiable.

(continued on next page)

Table 8–2 (Cont.): Selected Built-In Actions When ERASE_UNMODIFIABLE is Turned Off

Built-In	Action
MOVE_TEXT	Moves all records, perserving 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 and the buffer is not erase 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.
	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.

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.

8.6.3.6 SET (RECORD_ATTRIBUTE) Built-In

The new SET (RECORD_ATTRIBUTE) built-in sets or alters any of three possible attributes for the specified record or records. The record attributes you can set are its left margin, its modifiability, and its visibility.

Syntax

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 marking the record whose attribute you want to set.

range
The range containing the records whose attribute you want to set.

buffer
The buffer containing the records for which you want to set an attribute. When you specify a buffer for the second parameter, 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 record. 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. (The maximum valid value for the right margin is 960.)

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 using keyword 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. You can change the modifiability of a record if the buffer is modifiable.

Examples

```
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. If you do so, VAXTPU does not signal an error but does not make the requested modification to the record.

```
SET (RECORD_ATTRIBUTE, CURRENT_BUFFER, LEFT_MARGIN, 3);
```

This statement sets the left margin of all records in the current buffer to column 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.

```
SET (RECORD_ATTRIBUTE, MARK (FREE_CURSOR), MODIFIABLE, OFF);
```

This statement makes the current record unmodifiable.

8.7 Enhanced Support for DECwindows VAXTPU

This section describes the new and enhanced features of DECwindows VAXTPU.

8.7.1 Support for the Watch Cursor

When VAXTPU has been busy for more than one second, VAXTPU changes the pointer cursor to a watch cursor. No application intervention is necessary to cause this action, and applications cannot prevent it from happening. This makes VAXTPU and its layered applications more consistent with other DECwindows applications.

8.7.2 Support for Resizing Windows and Screens

VAXTPU has a new built-in, SET (HEIGHT), which facilitates resizing windows. Also, the SET (WIDTH) built-in has been enhanced to allow manipulation of the VAXTPU screen as well as windows.

In DECwindows, if using either the SET (WIDTH) or SET (HEIGHT) built-ins causes the terminal to resize, a resize event occurs. If a resize action routine has been declared, this routine is called the next time VAXTPU returns to its main loop.

When a resize event has occurred but has not yet been processed, the READ_KEY, READ_CHAR and READ_LINE built-ins abort. Thus, if you try to include these built-ins in a procedure that has performed a SET (WIDTH) or a SET (HEIGHT), you can get unexpected results.

The resize action routine is not called until control returns to the main VAXTPU input loop. This means that a SET (HEIGHT) or a SET (WIDTH) statement does not take effect until after section file initialization within the DECwindows environment.

No resize actions are available in the non-DECwindows version of VAXTPU. Thus, if you call SET (WIDTH) or SET (HEIGHT), you must be sure your code adjusts any application data that depends on the size of the terminal. No action routine is called, and the screen resizes immediately. The following subsections describe the SET (HEIGHT) and SET (WIDTH) built-ins in more detail.

8.7.2.1 SET (HEIGHT) Built-In

The new SET (HEIGHT) built-in sets the height of the VAXTPU screen without modifying the height or location of any VAXTPU window. For more information on the VAXTPU screen, see the *VAX Text Processing Utility Manual.*

Syntax

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

Note that, 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.

Example

```
SET (HEIGHT, SCREEN, 20);
```

This statement causes the screen to have a height of 20 lines.

8.7.2.2 SET (WIDTH) Built-In

The SET (WIDTH) built-in sets the width of a window or the VAXTPU screen. The following parameters are enhancements to the built-in. For more information on the VAXTPU screen, see the *VAX Text Processing Utility Manual.*

Syntax

SET (WIDTH, {window | ALL | SCREEN}, width_int)

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.

width_int
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. By default, the width of a window is the same as the physical width of the terminal when the window is created.

Example

```
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.

```
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.

8.7.3 Support for Icons

VAXTPU now supports the use of pixmaps as well as icon names. The following subsections describe the two new built-ins implementing this support: SET (ICON_PIXMAP) and SET (ICONIFY_PIXMAP).

8.7.3.1 SET (ICON_PIXMAP) Built-In

The new SET (ICON_PIXMAP) built-in determines the pixmap for the icon creation in the DECwindows icon box when the user selects the Large Window Manager Icon Style.

Syntax

Choose either of two variants:

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 for the icon creation in the DECwindows icon box when the user selects 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 non-standard, 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 syntax variant shown in the Syntax section. To specify an icon created in a bitmap file, use the second syntax variant shown in the Syntax 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.

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.

8.7.3.2 SET (ICONIFY_PIXMAP) Built-In

The new SET (ICONIFY_PIXMAP) built-in determines the pixmap for the icon creation in the DECwindows icon box when the user selects 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.

Syntax

Choose either of two variants:

SET (ICONIFY_PIXMAP, integer, icon_pixmap [,widget])

or

SET (ICONIFY_PIXMAP, bitmap_file_name [,widget])

Parameter

ICONIFY_PIXMAP

A keyword indicating that the SET built-in is being used to determine the pixmap for the icon creation in the DECwindows icon box when the user selects 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 syntax variant shown in the Syntax section. To specify an icon created in a bitmap file, use the second syntax variant shown in the Syntax 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.

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.

8.7.4 Support for Sending and Detecting Client Messages

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.

The following new built-ins enable applications layered on VAXTPU or EVE to designate and fetch client message action routines and to send and detect client messages:

- SET (CLIENT_MESSAGE)
- GET_INFO (SCREEN, "client_message_routine")
- SEND_CLIENT_MESSAGE
- GET_INFO (SCREEN, "client_message")

The following sections describe these new built-ins.

8.7.4.1 SET (CLIENT_MESSAGE) Built-In

The new SET (CLIENT_MESSAGE) built-in assigns the action routine to be executed when DECwindows VAXTPU receives a client message from another DECwindows application.

Syntax

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.

8.7.4.2 GET_INFO (SCREEN, "client_message_routine") Built-In

The new GET_INFO (SCREEN, "client_message_routine") built-in returns the program or learn sequence that is assigned as the application's client message handler routine.

Syntax

{program | learn_sequence} := GET_INFO (SCREEN, "client_message_routine")

Parameter

SCREEN
A keyword used to preserve compatibility with future versions of VAXTPU.

"client_message_routine"
A string indicating that you want GET_INFO to return the application's message action routine.

8.7.4.3 SEND_CLIENT_MESSAGE Built-In

The new SEND_CLIENT_MESSAGE built-in sends either of two client messages—STUFF_SELECTION or KILL_SELECTION—to other DECwindows applications.

Syntax

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.

Use the SEND_CLIENT_MESSAGE (KILL_SELECTION) message in response to a user request to copy and remove text from another application (that owns the input focus) into the VAXTPU/EVE layered application.

For this copy and delete function, the user selects the text in the application that owns the input focus by placing the pointer cursor on the desired location in the VAXTPU/EVE layered application; the user then presses the Ctrl key and clicks MB3. For this event, the VAXTPU/EVE layered application responds with a SEND_CLIENT_MESSAGE (KILL_SELECTION) 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.

Use the SEND_CLIENT_MESSAGE (STUFF_SELECTION) message in response to a user request to copy something from an application layered on VAXTPU or EVE into some other DECwindows application that owns the input focus.

For this copy function, the user presses and holds MB3 and drags the pointer over the selected text in the VAXTPU/EVE layered application. The VAXTPU /EVE layered application grabs ownership of this secondary global selection and responds by sending a SEND_CLIENT_MESSAGE (STUFF_SELECTION) message to the other application, which owns the input focus.

In response to the message, the other application requests to read the secondary global selection. This causes the VAXTPU/EVE layered application to write out the secondary global selection, which is then received by the other application.

Note that, if the user presses Ctrl/MB3 instead of just MB3 when dragging the pointer over selected text, 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.

8.7.4.4 GET_INFO (SCREEN, "client_message") Built-In

The new GET_INFO (SCREEN, "client_message") built-in 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 or EVE layered application's client message routine. This routine provides the application's response to a client message received from another application.

Syntax

```
{KILL_SELECTION   |
 STUFF_SELECTION | 0} := GET_INFO (SCREEN, "client_message")
```

Parameters

SCREEN

A keyword used to preserve compatibility with future versions of VAXTPU.

"client_message"

A string indicating that you want GET_INFO to return a keyword describing the type of client message returned.

Return Values

KILL_SELECTION

The call GET_INFO (SCREEN, "client_message") returns the keyword KILL_SELECTION when the user is copying from the input focused application (layered on VAXTPU or on EVE) to another application.

To perform this copy function, the user selects text in the VAXTPU/EVE layered application with the input focus. This designates the text to be placed in the primary global selection when another application asks to read the selection. The user then positions the pointer to the location in the other application where the text is to be inserted and clicks MB3. This causes the text in the primary global selection to be copied to the location indicated by the pointer.

If the user presses Ctrl/MB3 to copy the selection into the other application, the selection is then copied into the other application and deleted from the VAXTPU/EVE layered application where the selection was made. In this case, after the other application inserts the text from the primary global selection, the application sends a KILL_SELECTION client message to the VAXTPU/EVE layered application. When the VAXTPU/EVE layered application detects the

client message, 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.

STUFF_SELECTION

The call GET_INFO (SCREEN, "client_message") returns the keyword STUFF_SELECTION when the user copies from some other application that does not own the input focus to the VAXTPU/EVE layered application that owns the input focus.

The user performs a drag operation using MB3 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 receives the client message, 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.

0

An integer indicating that there is no current client message.

8.7.5 Other New Built-Ins Extending DECwindows VAXTPU

The following subsections describe the following nine new built-ins in DECwindows VAXTPU:

- GET_INFO (WIDGET, "children")
- GET_INFO (widget_variable, "class")
- GET_INFO (widget_variable, "is_managed")
- GET_INFO (widget_variable, "is_subclass")
- GET_INFO (widget_variable, "parent")
- GET_INFO (widget_variable, "resources")
- REALIZE_WIDGET
- SET (MAPPED_WHEN_MANAGED)
- SET (WIDGET_CALL_DATA)

8.7.5.1 GET_INFO (WIDGET, "children") Built-In

The new GET_INFO (WIDGET, "children") built-in 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 VAXTPU's main window widget.

Syntax

integer := GET_INFO (WIDGET, "children", {SCREEN | widget}, array_variable)

Parameters

WIDGET

A keyword indicating that you want information about a widget.

"children"

A string constant indicating that you want information about the children of a widget.

SCREEN

A keyword indicating that you want to fetch the children controlled by VAXTPU's main window widget.

widget

The widget instance whose children you want to fetch.

array_variable

An array created by VAXTPU and assigned to the variable you specify if the widget has any children. The array is integer indexed; its elements contain the children. If the widget has no children, the array variable is assigned the type UNSPECIFIED.

Example

The following example shows how to use this GET_INFO to display the entire hierarchy of widgets known to a VAXTPU session.

```
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;
```

8.7.5.2 GET_INFO (widget_variable, "class") Built-In

The new GET_INFO (widget_variable, "class") built-in returns the name of the class to which the specified widget instance belongs.

Syntax

string := GET_INFO (widget, "class")

Parameters

widget

The widget instance whose class you want to know.

"class"

A string constant indicating that you want to know the class of the specified widget.

Example

```
the_name := GET_INFO (example_widget, "class");
```

This statement places in the variable *the_name* the string that is the name of *example_widget's* class.

8.7.5.3 GET_INFO (widget_variable, "is_managed") Built-In

The new GET_INFO (widget_variable, "is_managed") built-in returns 1 (TRUE) if the specified widget is managed; otherwise, it returns (FALSE). This built-in calls the DECwindows Toolkit routine IS MANAGED.

Syntax

{0 | 1} := GET_INFO (widget, "is_managed")

Parameters

widget
The widget instance whose managed status you want to know.

"is_managed"
A string constant indicating that you want to know if a widget is managed.

Example

```
status := GET_INFO (example_widget, "is_managed");
```

This statement assigns to the variable *status* a 1 if the widget contained in *example_widget* is managed; otherwise, it assigns a 0.

8.7.5.4 GET_INFO (widget_variable, "is_subclass") Built-In

The new GET_INFO (widget_variable, "is_subclass") built-in 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).

Syntax

{0 | 1} := GET_INFO (widget, "is_subclass", integer)

Parameters

widget
The widget instance whose membership in a class or subclass you want to know.

"*is_subclass*"
A string constant indicating that you want to know whether the specified widget is in the class referred to by the specified integer or belongs to a subclass of that class.

integer
The integer returned by the DEFINE_WIDGET_CLASS built-in.

8.7.5.5 GET_INFO (widget_variable, "parent") Built-In

The new GET_INFO (widget_variable, "parent") built-in returns the parent of the specified widget instance. If the widget has no parent, the call returns 0.

Syntax

{0 | widget1} := GET_INFO (widget2, "parent")

Parameters

widget2
The widget instance whose parent you want to fetch.

"*parent*"
A string constant indicating that you want to fetch the specified widget instance's parent.

Example

```
the_parent := GET_INFO (example_widget, "parent");
```

This statement assigns to the variable *the_parent* the widget that is the parent of the widget stored in the variable *example_widget*.

8.7.5.6 GET_INFO (widget_variable, "resources") Built-In

The new GET_INFO (widget_variable, "resources") built-in 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)).

Syntax

array := GET_INFO (widget, "resources")

Parameters

widget

The widget instance whose resource classes and data types you want to know.

"*resources*"

A string constant indicating that you want to know the resource classes and data types for the specified widget instance.

Example

```
the_array := GET_INFO (example_widget, "resources");
```

After this statement is executed, the array contains elements that are example_ widget's resources. For example, if the widget assigned to *example_widget* is a previously created list box widget, this GET_INFO call causes VAXTPU to create elements in the array assigned to *the_array* and to index those elements with strings that are the names of a list box widget's resources. The contents of each array element are the data type and class of the named resource. Thus, the following statement:

```
first_value := GET_INFO (the_array, "first");
```

returns the following string index for the first element:

```
"accelerators"
```

The following statement:

```
MESSAGE (the_array{"accelerators"});
```

causes the following data type and class to be displayed in the message buffer:

```
AcceleratorTable Accelerators
```

8.7.5.7 REALIZE_WIDGET Built-In

The new REALIZE_WIDGET built-in creates a window for the specified widget instance and maps the window to the display. 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.*

Syntax

REALIZE_WIDGET (widget)

Parameter

widget

The widget instance you want VAXTPU to realize.

Example

```
REALIZE_WIDGET (example_widget);
```

This statement realizes the widget stored in *example_widget.*

8.7.5.8 SET (MAPPED_WHEN_MANAGED) Built-in

The new SET (MAPPED_WHEN_MANAGED) built-in controls whether a widget instance is mapped to the screen when it is managed. For more information on managing widgets, see the *VMS DECwindows Guide to Application Programming* and the *VMS DECwindows Toolkit Routines Reference Manual.*

Syntax

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.

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.

8.7.5.9 SET (WIDGET_CALL_DATA) Built-In

The new SET (WIDGET_CALL_DATA) built-in lets you create a template telling VAXTPU how to interpret the information in the fields of a widget's callback data structure.

Syntax

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

Request String	Associated Keyword(s)
"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).

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:DECWDWTDEF.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 in the "item number" field as type INTEGER.

8.8 Support for Setting and Fetching the Default Directory

VAXTPU now lets the application modify the default directory, using the SET (DEFAULT_DIRECTORY) built-in. To determine the current default directory, you can now use the new GET_INFO (SYSTEM, "default_directory") built-in. Applications that use the callable interface can intercept requests to change the default directory.

The following subsections describe the two new built-ins implementing this enhancement.

8.8.1 SET (DEFAULT_DIRECTORY)Built-in

The new SET (DEFAULT_DIRECTORY) built-in determines the directory that will be used as the default.

Syntax

[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 quoted 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 enters 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's tracking of the current default directory, you can use EVE command DCL SET DEFAULT.

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.

8.8.2 GET_INFO (SYSTEM, "default_directory") Built-In

The new GET_INFO (SYSTEM, "default_directory") built-in returns the name of the current default directory.

Syntax

directory_name_string := GET_INFO (SYSTEM, "default_directory")

Parameters

SYSTEM
A keyword indicating that you are fetching information about a system setting.

"default_directory"
A string constant indicating that you want to know the current default directory.

Example

```
the_directory := GET_INFO (SYSTEM, "default_directory");
```

This statement assigns to the variable *the_directory* the string that is the name of the current default directory.

8.8.3 Callable Interface Issues

VAXTPU lets you intercept requests to set or determine the current default directory. This feature parallels VAXTPU's callable interface support for other aspects of input and output.

The VAXTPU callable interface routine TPU$INITIALIZE now makes available two new item codes, TPU$_SET_DEFAULT and TPU$_GET_DEFAULT. If you do not specify these item codes, VAXTPU uses its own default routines.

For more information on the VAXTPU callable interface and the TPU$INITIALIZE routine, see the *VMS Utility Routines Manual.*

User-supplied routines must take the same arguments as the TPU-supplied routines and must return any errors as status codes. VAXTPU signals TPU$_ SYSERROR with the returned error code as a secondary status if a false value is returned.

The default routine for setting the default directory is as follows:

```
status = TPU$$SET_DEFAULT_DIRECTORY (result: REF $desc, new_default:
  REF $desc)
```

If an error occurs during execution of this routine, the default directory is not changed.

Parameters

result
Either 0 or a descriptor that is to be initialized as a dynamic string and set to the old default directory.

new_default
A string descriptor for the new default directory.

The default routine for determining the default directory is as follows:

```
status = TPU$$GET_DEFAULT_DIRECTORY (result: REF $desc)
```

Parameter

result
A descriptor that is to be initialized as a dynamic string and set to the current default directory.

8.9 Enhancements to the VAXTPU Compiler

The VAXTPU compiler has been enhanced in the following ways:

- VAXTPU now allows the programmer to limit the scope of local variables in unbound code (code that is not in a procedure).
- There is a new EQUIVALENCE statement.
- The compiler can perform conditional compilation. Lexical keywords of the form "%xxx" are used to indicate compilation directives. These keywords are case insensitive.
- Lexical prefixes can be used to specify the radix of numeric constants.

8.9.1 EQUIVALENCE Statement

VAXTPU now supports the creation of synonyms using the EQUIVALENCE statement.

Syntax

```
EQUIVALENCE synonym_name1 = real_name1, synonym_name2 =
real_name2, ...;
```

Elements of the EQUIVALENCE Statement

real_name
A user-defined global variable or procedure name. If *real_name* is undefined, VAXTPU defines it as an ambiguous name. This ambiguous name can become a variable or procedure later.

synonym_name
A name to be defined as a synonym for the *real_name*.

Description

Equivalences work only when both the *real_name* and the *synonym_name* are defined at the same time. You cannot save a section file containing the *real_name* and then later use that section file to extend code that uses an EQUIVALENCE of the saved name. To avoid problems, include all EQUIVALENCE statements in the same compilation unit where the *real_name* is defined. Alternatively, the equivalences can reside in different compilation units, but all of the compilation units must be used when building the section file from scratch. If you use a base section file that you extend interactively, you cannot make equivalences to procedures or variables defined in the base section file.

8.9.2 Support for Local Variables in Unbound Code

In previous versions of VMS, VAXTPU variables used in unbound code (code not in a procedure) were always global in scope. VAXTPU now lets you to define local variables in unbound code. Such variables are accessible only within that unbound code.

Unbound code can occur in the following places:

- Module initialization code. This occurs after all procedure declarations within a module but before the ENDMODULE statement.
- Executable code. This occurs after all module and procedure declarations in a file but before the end of file.

Example

The following example shows a complete compilation unit. This unit contains a module named *mmm* that, in turn, contains a procedure *foo* and some initialization code *mmm_module_init*, a procedure *bar* defined outside the module, and some unbound code at the end of the file. In each of these sections of code, a local variable *x* is defined. The variable is displayed using the MESSAGE built-in.

```
MODULE mmm IDENT "mmm"

PROCEDURE foo;       ! Declare procedure "foo" in module "mmm"

LOCAL
    x;  ! "x" is local to procedure "foo"

    x := "Within procedure foo, within module mmm";
MESSAGE (x);

ENDPROCEDURE; ! End procedure "foo"

LOCAL
        x;                      ! "x" is local to
                                ! procedure "mmm_module_init"

x := "Starting or ending the module init code";
MESSAGE (x);
foo;
MESSAGE (x);

ENDMODULE;              ! End module "mmm"

PROCEDURE bar           ! Declare procedure "bar"

LOCAL
        x;                      ! "x" is local to procedure "bar"

x := "In procedure bar, which is outside all modules";
MESSAGE (x);

ENDPROCEDURE;           ! End procedure "bar"

LOCAL
        x;                      ! "x" is local to the unbound code...
```

```
x := "Starting or ending the unbound, non-init code";
MESSAGE (x);
mmm_module_init;
foo;
bar;
MESSAGE (x);
EXIT;
```

If this code is included in a file TEMP.TPU, the following DCL command demonstrates the scope of the various local variables:

```
$ EDIT/TPU/NOSECTION/NOINITIALIZE/NODISPLAY/COMMAND=temp.tpu
42 lines read from file TEMP.TPU;1
Starting or ending the unbound, non-init code
Starting or ending the module init code
Within procedure foo, within module mmm
Starting or ending the module init code
Within procedure foo, within module mmm
In procedure bar, which is outside all modules
Starting or ending the unbound, non-init code
```

8.9.3 Support for Conditional Compilation

VAXTPU now provides lexical keywords for controlling what code is compiled under different conditions. The new lexical keywords are as follows:

- %IF
- %IFDEF
- %THEN
- %ELSE
- %ENDIF

Syntax

Conditional compilation lexical keywords are used in a manner similar to ordinary IF/THEN/ELSE/ENDIF statements. The syntax is as follows:

%IFDEF variable_or_proc_name %THEN ... [%ELSE ...] %ENDIF

or

%IF boolean_expression %THEN ... [%ELSE ...] %ENDIF

Description

If you use the %IFDEF structure, specify *variable_or_proc_name* as the name of a VAXTPU procedure or variable. IFDEF is a statement that says: "if a variable or procedure with this name is defined." If the name is defined, the compiler compiles the code marked by %THEN. If the name is not defined, the compiler compiles the code marked by %ELSE.

If you use the %IF structure, specify *boolean_expression* as either a numeric constant or a defined global variable whose value is an integer. Any odd value is true and any even value is false. If the variable or constant contains a value that is odd, the compiler compiles the code marked by %THEN. If the variable or constant contains a value that is even, the compiler compiles the code marked by %ELSE.

You do not have to put conditional compilation lexical keywords at the beginning of a line. You can nest conditional statements to a depth of 2**32-1.

Example

```
ON_ERROR
    [TPU$_CREATEFAIL]:
%IF eve$x_option_decwindows
%THEN
        IF eve$x_decwindows_active
        THEN
            eve$popup_message (MESSAGE_TEXT (EVE$_CANTCREADCL, 1));
        ELSE
            eve$message (EVE$_CANTCREADCL);
        ENDIF;
%ELSE
        eve$message (EVE$_CANTCREADCL);
%ENDIF
        eve$learn_abort;
        RETURN (FALSE);
    [OTHERWISE]:
ENDON_ERROR;
```

This ON_ERROR procedure determines whether a popup message widget or a simple message is used, depending on whether the code is being compiled by a DECwindows version of VAXTPU.

8.9.4 Support for Specifying the Radix of Numeric Constants

VAXTPU now supports specifying constants with binary, octal, hexadecimal, and decimal radixes.

To specify a numeric constant in binary, precede the number with %B. The number can consist only of digits 0 and 1.

To specify a numeric constant in octal, precede the number with %O. The number can consist only of digits 0 through 7.

To specify a numeric constant in hexadecimal, precede the number with %X. The number can consist of digits 0-9 and A-F.

There is no radix specifier for decimal. Any numeric constant without an explicit radix specifier is assumed to be decimal.

Examples

The following are examples of correct numeric constants:

```
!
! Many different ways of saying the same thing.
!
CONSTANT binary_constant := %b11111;
CONSTANT octal_constant := %o37;
CONSTANT decimal_constant := 31;
CONSTANT hex_constant := %x1f;
!
! Compile time expressions work, too.
!
CONSTANT negative_value := -%x1f;
CONSTANT strange_zero := hex_constant - %x1f;
```

Invalid constructs for numeric constants return the error level message TPU$_UNKLEXICAL, "Unknown lexical element," during compilation. The following examples are not valid:

```
constant bad_binary := %b123;       ! only 0's and 1's are legal.
constant bad_hex := %x10abg;        ! 'g' is illegal digit.
constant not_a_radix := %z0123;     ! No such radix.
```

8.10 Reserved Keywords

VAXTPU now reserves seven keywords for use in future versions. At present these keywords are not used by VAXTPU, but you should avoid using variables with these names to prevent future name conflicts. The reserved keywords are as follows:

- FILL_NOT_BEGIN
- FILL_NOT_END
- PHONETIC_OFFSET
- PHONETIC_LENGTH
- CLAUSE_NUMBER
- MAX_CLAUSE_NUMBER
- ALIGNMENT_DEFAULT

8.11 Support for Handling Detached Cursor Conditions

VAXTPU provides three new built-ins for handling detached cursor conditions. A detached cursor condition occurs when the cursor position cannot accurately represent the editing point in the current window. This condition occurs in any of five situations:

- The current window is not mapped to the current buffer. This state is known as a disjointed cursor.
- The editing point is off the left side of the (shifted) current window.
- The editing point is off the right side of the current window.
- The editing point is on an invisible record.
- No current window exists. This state is called an unmapped cursor.

VAXTPU allows an application to detect these conditions and to attempt to remedy them using an action routine. VAXTPU's main loop checks for a detached cursor condition after each key is pressed and after performing any applicable pre-key, post-key, and margin action routines. If the cursor is detached, VAXTPU executes the detached action routine if the application has defined one.

The following subsections describe the new built-ins implementing VAXTPU's detached cursor support.

8.11.1 SET (DETACHED_ACTION) Built-In

The new SET (DETACHED_ACTION) built-in 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).

Syntax

```
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 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").

Example

The following procedure is a simple detached cursor action routine:

```
PROCEDURE detached_routine

LOCAL rightmost_column,
      the_offset;

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");
```

8.11.2 GET_INFO (SCREEN, "detached_action") Built-In

The new GET_INFO (SCREEN, "detached_action") built-in returns the current detached action routine. If no such routine is designated, returns the type UNSPECIFIED.

Syntax

{unspecified | program} := GET_INFO (SCREEN, "detached_action")

Parameters

SCREEN
A keyword indicating that the call seeks information about a characteristic affecting the entire session, not just one buffer or window.

"detached_action"
A string constant indicating that you want VAXTPU to fetch the detached action routine.

Example

```
the_routine := GET_INFO (SCREEN, "detached_action")
```

This statement assigns to the variable *the_routine* the detached action routine if one exists. Otherwise, it assigns unspecified to the variable the_routine.

8.11.3 GET_INFO (SCREEN, "detached_reason") Built-In

The new GET_INFO (SCREEN, "detached_reason") built-in returns a bit-encoded integer that identifies to one of the five possible detached cursor states.

Syntax

integer := GET_INFO (SCREEN, "detached_reason")

Parameters

SCREEN
A keyword indicating that the call seeks information about a characteristic affecting the entire session, not just one buffer or window.

"detached_reason"
A string constant indicating that you want VAXTPU to return an integer representing the reason for the detached action condition.

Description

Digital recommends that you use VAXTPU's predefined constants, rather than the actual integers, to refer to the reasons for detachment. Table 8–3 shows the correspondence of constants, integers, and reasons.

Table 8–3: 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.

Example

```
the_value := GET_INFO (SCREEN, "detached_reason");
```

If the editing point is off the right edge of the screen, this statement causes VAXTPU to assign to *the_value* the integer 2.

8.12 Other Enhanced Built-Ins

The subsections in this section describe enhancements to 10 miscellaneous built-ins.

8.12.1 CHANGE_CASE Built-In

The CHANGE_CASE built-in 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.

Syntax

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

Parameter

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. CHANGE_CASE has no effect on string constants.

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.

Examples

```
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.

```
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".

8.12.2 CREATE_RANGE Built-In

The CREATE_RANGE built-in 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.

Syntax

```
range := CREATE_RANGE ({marker1 | delimiting_keyword},
                       {marker2 | delimiting_keyword}
                       [, attribute_keyword])
```

Parameters

marker1
The marker marking the point in the buffer where the range begins.

marker2
The marker marking the point in the buffer where the range ends.

delimiting_keyword
A keyword indicating the point in the buffer where you want the range to begin or end. Table 8–4 shows the valid keywords and their meanings.

Table 8–4: 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, regardless of whether there is a character there. This is the same as the point referred to by END_OF (CURRENT_BUFFER).

attribute_keyword

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

Description

If a marker defining a range is a free marker, VAXTPU creates a new bound marker, tied 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.

Example

```
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.

8.12.3 EDIT Built-In

The EDIT built-in modifies a string according to the keywords you specify. Currently, EDIT returns a value.

Syntax

```
[returned_buffer  |
 returned_range   |
 returned_string  := ] EDIT ({buffer | range | string},
                             keyword1[,...] [,keyword2] [,keyword3])
```

Parameters

buffer

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.

range

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.

string

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. EDIT has no effect on string constants.

keyword1

A keyword specifying the editing operation you want to perform on the string. Valid keywords are: COLLAPSE, COMPRESS, INVERT, LOWER, TRIM, TRIM_LEADING, TRIM_TRAILING, or UPPER. For more information on the effect of these keywords, see the description of the EDIT built-in procedure in the *VAX Text Processing Utility Manual*.

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 meaning are as follows:

Keyword	Meaning
IN_PLACE	Make the indicated change in place. This is the default.
NOT_IN_PLACE	Leave the specified string unchanged and return 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.

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

Example

```
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*.

8.12.4 GET_INFO (buffer_variable) Built-In

A new set of GET_INFO calls are provided to determine whether the editing point in a buffer to is bound or free. Table 8–5 shows the new calls and their previous equivalents.

Table 8–5: New GET_INFO Calls for the Editing Point and Their Previous Equivalents

New GET_INFO Call	Equivalent code
GET_INFO (CURRENT_BUFFER, "record_number")	GET_INFO (MARK (FREE_CURSOR), "record_number")
GET_INFO (CURRENT_BUFFER, "bound")	GET_INFO (MARK (FREE_CURSOR), "bound")
GET_INFO (CURRENT_BUFFER, "beyond_eol")	GET_INFO (MARK (FREE_CURSOR), "beyond_eol")

(continued on next page)

Table 8–5 (Cont.): New GET_INFO Calls for the Editing Point and Their Previous Equivalents

New GET_INFO Call	Equivalent code
GET_INFO (CURRENT_BUFFER, "before_bol")	GET_INFO (MARK (FREE_CURSOR), "before_bol")
GET_INFO (CURRENT_BUFFER, "middle_of_tab")	GET_INFO (MARK (FREE_CURSOR), "middle_of_tab")
GET_INFO (CURRENT_BUFFER, "beyond_eob")	GET_INFO (MARK (FREE_CURSOR), "beyond_eob")

8.12.5 LENGTH Built-In

The LENGTH built-in now accepts a buffer as well as a string or range. LENGTH returns the length of the buffer, in characters. There is no difference between asking for the length of a buffer or for the length of a range that spans the buffer.

Syntax

integer := LENGTH ({buffer | range | string})

Parameter

buffer
The buffer whose length you want to determine.

range
The range whose length you want to determine.

string
The string whose length you want to determine.

8.12.6 MESSAGE Built-In

The MESSAGE built-in now accepts a buffer as well as a range or string.

Syntax

MESSAGE ({buffer | range | string} [, integer1])

Parameters

buffer
A buffer whose contents you want displayed in the message area.

range
A range whose contents you want displayed in the message area.

string
A string to be displayed in the message area.

integer1
An integer indicating the severity of the message placed in the message buffer. For more information on this parameter, see the description of the MESSAGE built-in in the *VAX Text Processing Utility Manual*.

8.12.7 MODIFY_RANGE Built-In

The MODIFY_RANGE built-in now accepts the keywords LINE_BEGIN, LINE_END, BUFFER_BEGIN, and BUFFER_END as well as marks.

Syntax

```
MODIFY_RANGE (range, [{marker1 | delimiting_keyword},
                      {marker2 | delimiting_keyword}]
                      [, attribute_keyword])
```

Parameters

range
The range to be modified.

marker1
The starting mark for the range.

marker2
The ending mark for the range.

delimiting_keyword
A keyword indicating the point in the buffer where you want the range to begin or end. Table 8–6 shows the valid keywords and their meanings. Use of the delimiting keywords are more efficient than the BEGINNING_OF and END_OF built-ins.

Table 8–6: 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).

attribute_keyword
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.

8.12.8 POSITION Built-In

The POSITION built-in now accepts the BUFFER_BEGIN and BUFFER_END keywords for its parameter.

POSITION (BUFFER_BEGIN) puts the editing point at the beginning of the current buffer and is equivalent to POSITION (BEGINNING_OF (CURRENT_ BUFFER)).

POSITION (BUFFER_END) puts the editing point at the end of the current buffer and is equivalent to POSITION (END_OF (CURRENT_BUFFER)).

The use of the BUFFER_BEGIN and BUFFER_END keywords is more efficient than using the built-ins BEGINNING_OF and END_OF.

8.12.9 SUBSTR Built-In

The SUBSTR built-in now accepts a buffer as well as a string or range in its first parameter. In addition, the third parameter is now optional. If no third parameter is specified, it is assumed that the substring extends to the end of the input buffer, range, or string.

Syntax

string2 := SUBSTR ({buffer | range | string}, integer1 [, integer2])

Parameter

buffer
A buffer containing the substring.

range
A range containing the substring.

string
A string containing 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 substring's end point at the end of the specified buffer, range, or string.

Example

```
!
! The following two calls to SUBSTR return the same value.
!
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;
```

8.12.10 TRANSLATE Built-In

The TRANSLATE built-in procedure now returns a value. This value might be the range or buffer translated, or might be a string representation of the translated text.

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.

Syntax

```
[returned_buffer |
 returned_range  |
 returned_string  := ] TRANSLATE (buffer | range | string1}, string2,
                          string3 [, {IN_PLACE | NOT_IN_PLACE}])
```

Parameters

buffer
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.

range
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.

string1
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. TRANSLATE has no effect on string constants.

string2
The string of replacement characters.

string3
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

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 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, when you specify a string for the first parameter. TRANSLATE can return a string even if you specify IN_PLACE.

Example

The following statements show how the 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.

8.13 TPU$_FILEIO Item Code

The TPU$INITIALIZE routine calls a user-specified routine, which returns an item list. In previous versions of VAXTPU, this item list was required to include an entry for item code TPU$_FILEIO. (Most users simply specified the default VAXTPU routine, TPU$$FILEIO.)

The item list is no longer required to specify the TPU$_FILEIO item code. If not present, the default of TPU$$FILEIO is used.

8.14 TPU$_CHAIN Item Code

Item lists can now be chained, as they are for item lists of the VMS system services. Chained item lists allow applications to add additional item list entries to an item list returned by some other code. Note that item list entries later in the list override earlier entries. Also note that this does not allow an application to set additional option bits.

The BUFFER ADDRESS portion of the TPU$_CHAIN item list entry contains the address of the new item list segment.

Any item list entries after TPU$_CHAIN are ignored.

VAXTPU scans each item list segment for a memory-freeing routine and calls it (if specified) to allow an application to free up item list memory when processing is completed. Memory-freeing routines are effective only for the item list segment in which they are declared. If more than one is specified in a single segment, VAXTPU calls only the last one declared in the segment.

8.15 Enhancements to Keyboard Support

VAXTPU's keyboard support is enhanced as follows:

- Several new keys are supported as function keys.
- Two key modifiers can now modify main array keys (keys found on the main keyboard on most Digital terminals) and control-modified keys.

VAXTPU now supports F1, F2, F3, F4, and F5 as valid function keys.

Note that some operatings systems or terminals might trap these keys before the signals they send can be processed by VAXTPU. In such an operating system or on such a terminal, it is possible to bind routines to these keys, but the bound routines cannot be executed.

VAXTPU now lets you to modify main array keys and control modified keys with the modifiers ALT_MODIFIED and HELP_MODIFIED. For more information on these modifiers, see the descriptions of KEY_NAME and GET_INFO (any_keyname) in the *VAX Text Processing Utility Manual*.

Modified main array keys and control modified keys do not have a graphic rendition. These keys might be saved in section files and will be handled properly by the STR and GET_INFO (keyname, "name") built-ins.

The KEY_NAME built-in has been enhanced to handle all modifiers on main array and control modified keys. For example, the following statements return the keyword KEY_NAME ("A") in *this_result* and the keyword CTRL_A_KEY in *that_result*:

```
this_result := key_name ("a", SHIFT_MODIFIED);
that_result := key_name ("a", CTRL_MODIFIED);
```

8.16 Enhancements to the /NODISPLAY Command Qualifier

Previously, some VAXTPU built-ins did not work when VAXTPU was invoked with the /NODISPLAY qualifier. The built-ins that did not work included UPDATE, CREATE_WINDOW, and ADJUST_WINDOW.

All VAXTPU built-ins now work in /NODISPLAY mode as they do in /DISPLAY mode. The only difference between /NODISPLAY mode and /DISPLAY mode is that in /NODISPLAY mode no output occurs. The only exception to this is the MESSAGE built-in, which continues to write to SYS$OUTPUT if there is no message buffer.

You might want to alter any existing code that checks the /NODISPLAY and /DISPLAY modes before using certain built-ins. This check is no longer necessary.

Chapter 9

EVE

The Extensible VAX Editor (EVE) is a general-purpose text editor based on the VAX Text Processing Utility (VAXTPU). To invoke EVE, use the EDIT/TPU command.

EVE has the following new and changed features:

- Improved handling of input files
- Buffer change journaling and recovery as well as keystroke journaling and recovery
- Saving attributes in a section file or command file
- User-defined menu items on DECwindows
- Case-exact searches
- Enhanced key definitions and key names
- Enhanced paragraph boundaries for FILL commands
- Improved Buffer List buffer
- Batch editing with the EDIT/TPU/NODISPLAY command
- Changes to other commands, including enhanced SHOW command output
- Program-level changes (of interest if you build your own VAXTPU applications using EVE as a base)

EVE provides extensive online help for all commands, keys, and other topics. Each HELP topic provides examples or a list of steps and other information.

9.1 Input File Handling

When you invoke EVE, if the input file is ambiguous, EVE delays applying the following command line qualifiers until you resolve the file name:

/[NO]MODIFY
/[NO]OUTPUT
/[NO]READ_ONLY
/START_POSITION
/[NO]WRITE

For example, the following command invokes EVE to edit a file with the type TXT, putting the cursor on line 5, column 20:

```
$ EDIT/TPU *.txt/START_POSITION=(5,20)
```

If more than one file matches your wildcard request—for example, if you have two files, LETTER.TXT and MEMO.TXT—EVE displays the matching files so you can choose the one you want. The list appears in an EVE system buffer named $CHOICES$ in a second window. (For information about using the $CHOICES$ buffer, see the EVE online help topic called Choices Buffer.) After you resolve the file name, EVE copies the file into a buffer and then applies the /START_POSITION qualifier.

If you specify an input file using a search list or a wildcard directory (such as [...]), EVE gets the first matching file found—without displaying the $CHOICES$ buffer.

In the following example, you define a search list called STAFFMEMOS and then invoke EVE to edit a file from that search list:

```
$ DEFINE staffmemos hiring.dat,promotion.lis,salary.txt
$ EDIT/TPU staffmemos
```

In this example, if the first file in the search list exists, EVE copies that file (HIRING.DAT) into a buffer, using the file name and file type as the buffer name; if it does not exist, EVE tries to get the second file (PROMOTION.LIS), and so on. If none of the files in the search list exists, EVE treats the name of the search list as a file name and then creates an empty buffer named STAFFMEMOS.

In the following example, you invoke EVE using a wildcard directory ([...]) to edit a file called JABBER.TXT in your current directory or in a subdirectory of the current directory. EVE searches through the directory tree and gets the first JABBER.TXT file found.

```
$ EDIT/TPU [...]jabber.txt
```

This way of handling a search list or wildcard directory applies not only to the EDIT/TPU command for invoking EVE, but also to the following EVE commands that use a file specification as a parameter:

@ (at sign)
GET FILE
INCLUDE FILE
OPEN
OPEN SELECTED
RECOVER BUFFER

9.2 Journaling and Recovery

Journal files record your edits so that, if a system failure interrupts your editing session, you can recover your work. EVE now provides two types of journaling and recovery:

- **Buffer change journaling** creates a separate journal file for each text buffer you create. This is the new EVE default. Buffer change journaling works both on DECwindows and on character-cell terminals. You recover one buffer at a time, typically by using RECOVER BUFFER commands in EVE. You can recover buffers from different editing sessions. The recovery restores only your text—it does *not* restore settings, key definitions, or the contents of system buffers (such as the Insert Here buffer) before the system failure.
- **Keystroke journaling** is unchanged. Keystroke journaling creates a single journal file for the editing session. Keystroke journaling works only on character-cell terminals—it does not work on DECwindows—and has other restrictions. The recovery re-creates your editing session stroke for stroke, in a "player piano" fashion.

It is possible to have both types of journaling for an editing session, although there is usually no reason to do so. Generally, buffer change journaling is the better method to use because it has fewer restrictions and the recovery is usually quite fast.

You can disable both kinds of journaling by using the /NOJOURNAL qualifier when you invoke EVE—typically, when you use EVE to examine a file without making any edits (such as with /READ_ONLY) or for demonstration sessions.

For information about the new and changed VAXTPU features for journaling and recovery, see Chapter 8.

9.2.1 Buffer Change Journaling

Buffer change journaling creates a journal file for each text buffer. (EVE does not create buffer change journal files for system buffers such as the Insert Here buffer, DCL buffer, or $RESTORE$ buffer.) As you edit a buffer, the journal file records the changes you make, such as erasing, inserting, or reformatting text. When you exit from EVE or when you delete the buffer, the journal files are deleted. If a system failure interrupts your editing session, the journal files are saved. Your last few keystrokes before the system failure might be lost. Table 9–1 summarizes the new EVE commands for buffer change journaling and recovery.

Table 9–1: EVE Commands for Buffer Change Journaling and Recovery

Command	Usage or Effects
RECOVER BUFFER	Recovers a specified buffer by using the journal file for the buffer. You can specify the name of the buffer or file you want to recover or the name of the journal file for the buffer.
RECOVER BUFFER ALL	Recovers all your text buffers—one at a time—by using the journal files for the buffers, if there are any.
SET JOURNALING	Enables buffer change journaling for a buffer that you specify.
SET JOURNALING ALL	Enables buffer change journaling for all your buffers. (Default setting.)
SET NOJOURNALING	Disables buffer change journaling for a buffer you specify.
SET NOJOURNALING ALL	Disables buffer change journaling for all your buffers.

Buffer change journal files are written in a directory defined by the logical name TPU$JOURNAL. By default, this directory is SYS$SCRATCH, which is typically your top-level or login directory. You can redefine the TPU$JOURNAL logical name to have the journal files written in a different directory. (This logical name does not apply to keystroke journal files.) For example, the following commands create a subdirectory called [USER.JOURNAL] and then define TPU$JOURNAL as this subdirectory:

```
$ CREATE/DIRECTORY [user.journal]
$ DEFINE TPU$JOURNAL [user.journal]
```

You can also put the definition in your LOGIN.COM file.

Buffer change journal files can be quite large (even larger than the text files you edit). Because of the potential size of buffer change journal files and because there is a journal file for each text buffer, you might want to define TPU$JOURNAL as a directory or subdirectory on a large disk, rather than as SYS$SCRATCH.

The name of the buffer change journal file derives from the name of the file or buffer being edited and the file type TPU$JOURNAL, as follows:

Text Buffer Name	Buffer Change Journal File
JABBER.TXT	JABBER_TXT.TPU$JOURNAL
GUMBO_RECIPE.RNO	GUMBO_RECIPE_RNO.TPU$JOURNAL
MAIN	MAIN.TPU$JOURNAL
LATEST NEWS	LATEST_NEWS.TPU$JOURNAL

To find the name of the journal file for the current buffer, use the SHOW command.

There are two ways to recover your edits with buffer change journal files—using the /RECOVER qualifier on the command line when you invoke EVE or using RECOVER BUFFER commands within EVE.

In the following example, you are editing a file called JABBER.TXT when a system failure interrupts your editing session. You then recover your edits by using the /RECOVER qualifer:

```
$ EDIT/TPU jabber.txt
                .
                .
                .
   ***  system failure  ***
                .
                .
                .
$ EDIT/TPU jabber.txt/RECOVER
```

Alternatively, you can invoke EVE and use the following command to recover your text:

```
Command: RECOVER BUFFER jabber.txt
```

If the buffer change journal file is available (in this case, a file named JABBER_TXT.TPU$JOURNAL), EVE shows the following information and asks if you want to recover that buffer:

Name of the buffer
Original input file for the buffer, if any
Output file for the buffer, if any
Source file for recovery, if any
Starting date and time of the editing session
Journal file creation date and time

If you want to recover the buffer, press Return. Otherwise, type NO and press Return.

If the buffer you want to recover exists—typically, the MAIN buffer—EVE first deletes that buffer and then does the recovery. If the buffer you want to recover has been modified, EVE prompts you whether to delete the buffer before recovering.

If you are unsure of the buffer names or journal file names, specify the asterisk wildcard, as follows:

```
Command: RECOVER BUFFER *
```

EVE then displays a list of all your available journal files so you can choose the one you want. The list appears in an EVE system buffer named $CHOICES$ in a second window. For information about using the $CHOICES$ buffer, see the EVE online help topic called Choices Buffer.

To recover all your text buffers—one at a time—use the RECOVER BUFFER ALL command. EVE then tries to recover each text buffer for which there is a buffer change journal available. The effect is the same as repeating the RECOVER BUFFER command, without having to specify buffer names or journal file names. For each text buffer, EVE displays information such as the buffer name, the files associated with the buffer, and the time and date the journal file was created. EVE prompts you for one of the following:

Response	Effects
YES	Recovers the buffer, and then asks you whether to recover the next buffer, if there is one. This is the default response—you can simply press Return.
NO	Skips this recovery. If there is another buffer to recover, EVE asks you about the other buffer.
QUIT	Cancels—does not recover the buffer and does not continue recovery operations.

You can disable buffer change journaling for a particular buffer by using the SET NOJOURNALING command. To disable buffer change journaling for all your buffers, use the SET NOJOURNALING ALL command. Typically, you disable buffer change journaling if you are using keystroke journaling instead (see Section 9.2.2) or if there is no need to journal the edits (such as when the buffer is simply a "scratchpad" or temporary storage area for reading a file).

SET NOJOURNALING commands do not delete the buffer change journal files. To delete the journal files, use the DCL command DELETE. For example, to delete all the buffer change journal files, use the following command:

```
$ DELETE TPU$JOURNAL:*.TPU$JOURNAL;*
```

If you disabled buffer change journaling, you can enable journaling by using the SET JOURNALING command. For example, the following command enables journaling for a buffer named JABBER.TXT:

```
Command: SET JOURNALING jabber.txt
```

If you invoked EVE with the /NOJOURNAL qualifier and then want to enable buffer change journaling during the editing session, use the SET JOURNALING ALL command (which is otherwise the EVE default).

Note that you cannot enable buffer change journaling after the buffer has been modified in an editing session. In such a case, EVE displays the following message:

```
Command: SET JOURNALING memo.txt
Buffer MEMO.TXT is not safe for journaling
```

You should first write out (save) the buffer by using the WRITE FILE or SAVE FILE command, and then enable journaling.

9.2.2 Keystroke Journaling and Recovery

Keystroke journaling is unchanged. Keystroke journaling creates a single journal file for the editing session, regardless of the number of buffers you create. The journal file records your keystrokes, including commands, for the editing session rather than simply the changes to text. To enable keystroke journaling, invoke EVE using the /JOURNAL qualifier and specify the keystroke journal file you want created. The default file type for keystroke journal files is TJL. The journal file is written in your current directory (or whatever directory you specify on the command line).

Normally, when you exit or quit, the keystroke journal file, if any, is deleted. If a system failure interrupts your editing session, the journal file is saved. Your last few keystrokes before the system failure might be lost. To recover your edits, you re-enter the command for the interrupted editing session, including all command line qualifiers, and add the /RECOVER qualifier. EVE then replays your editing session in a "player piano" fashion. Typically, you then exit to save the recovered text.

In the following example, you invoke EVE to edit a file called JABBER.TXT. The keystroke journal file is called MYJOU.TJL. (EVE also creates a buffer change journal file by default.)

```
$ EDIT/TPU/JOURNAL=myjou jabber.txt
                .
                .
                .
   ***  system failure  ***
                .
                .
                .
$ EDIT/TPU/JOURNAL=myjou jabber.txt/RECOVER
```

Note that, when recovering your edits with a keystroke journal file, you must specify the journal file name on the command line. If you use /RECOVER without using /JOURNAL and the keystroke journal file name, EVE tries to execute a RECOVER BUFFER command on a buffer change journal file. (See Section 9.2.1.)

Keystroke journaling has some restrictions that do not apply to buffer change journaling. Before recovering your edits with a keystroke journal file, make sure all relevant files and terminal settings are the same as when you began the original editing session. If you wrote out any buffers before the system failure, you might want to rename the saved files or move them to a different directory to ensure that the recovery uses the original versions of the files. If you saved

attributes, make sure that the recovery uses the original version of your section file or command file. Also, check that the following terminal settings are the same as when you began the editing session you are recovering:

Device_Type
Edit_mode
Eightbit
Page
Width

Recovery with a keystroke journal file might fail or might not work properly if you used Ctrl/C to halt or cancel an operation during the editing session. Keystroke journaling does not record Ctrl/C. Therefore, when you replay your keystrokes, the operation continues uninterrupted.

Keystroke journaling is particularly useful to record (and re-create) a problem for debugging purposes. If you have a problem with EVE or VAXTPU and want to submit a software performance report (SPR), be sure to submit the keystroke journal file if there is one, as well as other relevant files, the output from the SHOW SUMMARY command, and a description of the problem.

9.3 Attribute Saving

You can now save global attribute settings (**Attributes**) for future editing sessions in a section file or TPU command file, either by using the SAVE ATTRIBUTES command during an editing session or responding YES to EVE's prompt when exiting from a session. Table 9–2 lists the default settings that you can save.

Table 9–2: EVE Commands for Saving Default Attributes

Command	Default Setting
SET CLIPBOARD	SET NOCLIPBOARD
SET CURSOR BOUND	SET CURSOR FREE
SET DEFAULT COMMAND FILE	SET NODEFAULT COMMAND FILE
SET DEFAULT SECTION FILE	SET NODEFAULT SECTION FILE
SET FIND CASE EXACT	SET FIND CASE NOEXACT
SET NOEXIT ATTRIBUTE CHECK	SET EXIT ATTRIBUTE CHECK
SET NOSECTION FILE PROMPTING	SET SECTION FILE PROMPTING

(continued on next page)

Table 9–2 (Cont.): EVE Commands for Saving Default Attributes

Command	Default Setting
SET PENDING DELETE	SET NOPENDING DELETE
SET TABS MOVEMENT or SET TABS SPACES	SET TABS INSERT
SET TABS VISIBLE	SET TABS INVISIBLE

If you have an EVE initialization file containing commands for these settings, you can delete those command lines after you save the settings in your section file or command file.

Other global settings (such as scroll margins or the type of wildcards) and any buffer settings (such as margins or tab stops) are not saved. Typically, you use an initialization file for those settings. For a list of the EVE default settings, see the EVE online help topic called Defaults.

Table 9–3 summarizes the new and changed commands for saving attributes.

Table 9–3: EVE Commands for Saving Attributes

Command	Usage or Effects
SAVE ATTRIBUTES	Saves attributes in a section file or command file, depending on your responses to EVE prompts or settings done with other EVE commands. If you save attributes in a section file, the effect is the same as entering the SAVE EXTENDED EVE command. If you save attributes in a command file, EVE generates a specially marked block of VAXTPU statements for attribute settings and menu definitions, and either creates a command file or updates an existing command file with this block of statements.
SAVE SYSTEM ATTRIBUTES	Saves EVE default attributes in a section file or command file. This is useful if you want to restore your section file or command file to the standard EVE settings and menu definitions. See Section 9.3.3.

(continued on next page)

Table 9–3 (Cont.): EVE Commands for Saving Attributes

Command	Usage or Effects
SAVE EXTENDED EVE	Creates a section file, saving attributes, key definitions, menu definitions, compiled procedures, and other extensions such as global variables set with a VAXTPU statement. If you do not specify a section file on the command line, EVE prompts you for one or uses your default section file (if you set a default).
SET DEFAULT COMMAND FILE	Determines the command file for saving attributes. Does not determine the command file to be executed at startup, if any.
SET DEFAULT SECTION FILE	Determines the section file for saving attributes. Does not determine the section file to be executed at startup.
SET EXIT ATTRIBUTE CHECK	Default setting. If you changed attributes, then EVE asks if you want to save your changes when you exit or quit.
SET NODEFAULT COMMAND FILE	Default setting. When you save attributes, the default command file is TPU$COMMAND.TPU in your current directory, or the command file that was executed at startup. See Section 9.3.2.
SET NODEFAULT SECTION FILE	Default setting. When you save attributes, EVE asks for the name of the section file you want to create (unless you disabled section file prompting).
SET NOEXIT ATTRIBUTE CHECK	Disables attribute checking, typically to speed up or simplify exiting or quitting. Does not apply to the editing session in which you enter the command, but only to the editing sessions in which you use the section file or command file in which you saved the setting.
SET NOSECTION FILE PROMPTING	Disables prompting for a section file when you save attributes, typically to speed up or simplify saving attributes in a default section file or in a command file.
SET SECTION FILE PROMPTING	Default setting. When you save attributes, EVE prompts you for the name of a section file.

You can save attributes during your editing session by using the SAVE ATTRIBUTES or SAVE EXTENDED EVE command—or as part of exiting or quitting. By default, if you have changed attributes and not saved them, then on exiting, EVE prompts you as follows:

```
Command: SET CURSOR BOUND
Command: SET FIND CASE EXACT
Command: SET TABS VISIBLE
           .
           .
           .
Command: EXIT
Attributes were changed.  Save them? [YES]
```

If you want to save the changes, simply press Return. Effectively, EVE then executes the SAVE ATTRIBUTES command before exiting. If you do not want to save the changes, type NO and press Return. EVE then continues exiting.

To disable this prompting to make exiting faster or simpler, use the SET NOEXIT ATTRIBUTE CHECK command. However, the command does not apply to the current editing session because exit checking is itself a global setting and can be saved in a section file or command file. After you save it, the setting applies to future editing sessions in which you use the relevant section file or command file.

9.3.1 Saving Attributes in a Section File

Typically, you save attributes in a section file. A section file is in binary form and saves attributes, key definitions (including learn sequences), menu definitions, compiled procedures, and other extensions to the editor—including any saved in the section file you are using. In effect, the section file is your customized version of EVE. Because the section file is binary, it is executed quickly at startup. The default file type for section files is TPU$SECTION.

To create a section file, you can use the SAVE EXTENDED EVE command (as in previous versions of EVE) or the SAVE ATTRIBUTES command. Using SAVE EXTENDED EVE, you can specify the section file on the command line or let EVE prompt you for the section file name. Using SAVE ATTRIBUTES, you specify the section file as a response to a prompt.

For example, the following command saves attributes and other customizations in a section file called MYSEC.TPU$SECTION in your current directory:

```
Command: SAVE ATTRIBUTES
Save attributes in a section file [YES]? Return
File to save in: mysec
DISK$1:[USER]MYSEC.TPU$SECTION;1 created
```

To speed up saving attributes in a section file, you can set a default section file—that is, the section file you want to save in without having to specify the file each time save attributes—and you can disable section file prompting. Table 9–4 shows the interaction of the settings for default section file and section file prompting.

Table 9–4: EVE Settings for Saving Attributes

Commands (Settings)	Effects with SAVE ATTRIBUTES
SET DEFAULT SECTION FILE SET SECTION FILE PROMPTING	When you save attributes, EVE asks you whether to save in a section file. If you respond YES (the default response), EVE saves in your default section file. If you respond NO, EVE asks whether to save in a command file.
SET DEFAULT SECTION FILE SET NOSECTION FILE PROMPTNG	When you save attributes, EVE saves in your default section file without prompting.
SET NODEFAULT SECTION FILE SET SECTION FILE PROMPTING	Default settings. When you save attributes, EVE asks whether to save in a section file. If you respond YES, EVE asks for the name of a section file. If you respond NO, EVE asks whether to save in a command file.
SET NODEFAULT SECTION FILE SET NOSECTION FILE PROMPTNG	When you save attributes, EVE asks whether to save in a command file. (See Section 9.3.2.)

When you use the SET DEFAULT SECTION FILE command, you usually specify the section file you are going to use at startup for future editing sessions. The command does not determine the section file to be executed when you invoke the editor, but only the section file in which you save attributes and other customizations. To specify the section file you want executed at startup, do either of the following:

- Enter the EDIT/TPU/SECTION command and specify the section file you want to use.
- Define the logical name TPU$SECTION in your LOGIN.COM file to specify the section file, and then use the EDIT/TPU command.

Note that, in specifying the section file to be executed, you must use a complete file specification, including the device (or disk) and directory. Otherwise, VAXTPU assumes the section file is in SYS$SHARE.

Section files can be quite large, depending on the number of key definitions, menu definitions, and procedures you save. If you have limited disk space, you should save attributes in a command file, which requires less disk space. For more information about creating and using section files, see the EVE online help topic called Section Files.

9.3.2 Saving Attributes in a Command File

A command file contains VAXTPU procedures and statements that are compiled and executed at startup—in effect, a series of programs for extending EVE. (You can also use a command file for batch editing.) A command file might be slower at startup than a section file (depending on the number of procedures to be compiled and statements to be executed), but it takes up less disk space than a section file, and a command file can be edited and printed. Also, if you edit your command file, you can recompile procedures during your editing session by using EXTEND commands. The default file type for command files is TPU.

When you use the SAVE ATTRIBUTES command or when you save attributes on exiting or quitting, you can have EVE create or update a command file. EVE then generates a specially marked block of VAXTPU statements for your settings and menu definitions. Thus, if you created a command file with procedures and key definitions of your own, you can have EVE append the block of attribute settings to this command file. Example 9–1 is a sample of the EVE-generated code.

Example 9–1: EVE-Generated Code for Saving Attributes in a Command File

```
! EVE-generated code begin
! EVE attributes begin
eve_set_find_case_exact;
eve_set_cursor_bound;
eve_set_nodefault_command_file;
eve_set_nodefault_section_file;
eve_set_exit_attribute_check;
eve_set_pending_delete;
eve_set_nosection_file_prompting;
eve_set_tabs ('INSERT');
eve_set_tabs ('VISIBLE');
! EVE attributes end
! EVE-generated code end
```

To save attributes in a command file, use the SAVE ATTRIBUTES command, as follows:

```
Command:  SAVE ATTRIBUTES
Save attributes in a section file [YES]?  no
Save attributes in a command file [YES]?  Return
Enter file name [TPU$COMMAND.TPU]  mycom
14 written to file DISK$1:[USER]MYCOM.TPU;1
```

Note that the prompt for the command file name shows, in brackets, the default command file that EVE uses if you press Return at the prompt without typing a file name. This default is one of the following:

- The command file specified with the /COMMAND qualifier when you invoked EVE

- The command file defined by the logical name TPU$COMMAND
- A command file called TPU$COMMAND.TPU in your current directory

You can set your preferred default command file—that is, the command file you want EVE to create or update without having to specify the file each time you save attributes. For example, the following command sets your default command file as MYCOM.TPU in your current directory:

```
Command: SET DEFAULT COMMAND FILE mycom
```

If you want to save attributes in a command file rather than in a section file, you should also use the SET NOSECTION FILE PROMPTING command. Then, when you save attributes, EVE asks whether to save in a command file, without first asking whether to save in a section file.

When you use the SET DEFAULT COMMAND FILE command, you usually specify the command file you are going to use at startup for future editing sessions. The command does not determine the command file to be executed when you invoke EVE, only the command file in which you save attributes and menu definitions. To specify the command file you want executed at startup, do any of the following:

- Use EDIT/TPU/COMMAND and specify the command file you want to use.
- Name the command file TPU$COMMAND.TPU in your current directory and then invoke EVE using EDIT/TPU.
- Define the logical name TPU$COMMAND in your LOGIN.COM file to specify the command file and then invoke EVE using EDIT/TPU.

For more information about creating and using command files, see the EVE online help topic called Command Files.

9.3.3 Saving EVE Default Attributes

The SAVE SYSTEM ATTRIBUTES command saves EVE default settings and menu entries in a section file or command file. Thus, if you set several attributes and defined or undefined menu entries, you can use SAVE SYSTEM ATTRIBUTES to restore the standard EVE settings and menus to your section file or command file.

SAVE SYSTEM ATTRIBUTES does not change the settings currently in effect—for example, it does not enable free cursor motion or invisible tabs—but only saves the EVE defaults in a section file or command file.

9.4 Menu Entries

If you invoke EVE with the /DISPLAY=DECWINDOWS qualifier, you can add and remove menu items by using the DEFINE MENU ENTRY and UNDEFINE MENU ENTRY commands or by choosing Extend Menu from the Customize menu. You can save your menu definitions in a section file or command file for future editing sessions.

To add a menu item, do the following steps:

1. Enter the DEFINE MENU ENTRY command.
2. Enter the name of the menu to which you want to add an item—any of the following:

   File Pulldown
   Edit Pulldown
   Search Pulldown
   Display Pulldown
   Format Pulldown
   Customize Pulldown
   Help Pulldown
   Select Popup

   Put the menu name in quotes or let EVE prompt you for the menu name. You need only use the first term of the menu name, but you cannot abbreviate this term.
3. Enter the name of the EVE command you want the menu item to execute. Put the command name in quotes or let EVE prompt you for the command name.
4. Enter the label or name that you want to appear on the menu. Put the label in quotes or let EVE prompt you for it. If you simply press Return at the prompt, without typing anything, the label is the same as the command name.
5. Enter YES or NO to specify whether you want a separator line to appear above the label as a visual aid or for aesthetics. Put the response in quotes or let EVE prompt you for it.

For example, the following command adds SHOW BUFFERS to the File menu, labeling the item Buffer List, and adds a line separator above the item:

```
Command: DEFINE MENU "File" "SHOW BUFFERS" "Buffer List" "Yes"
```

The following command removes CENTER LINE from the pop-up menu that is displayed with MB2 when there is no selection:

```
Command: UNDEFINE MENU "Noselect" "Center Line"
```

You can also add or remove menu items by choosing Extend Menu from the Customize menu. Extend Menu displays a dialog box containing three list boxes, similar to the Verbs and Menus list boxes in FileView, and has buttons for adding or removing a menu item. The lists show the available commands, the names of EVE menus, and the items in a particular menu. The dialog box also as entry lines so you can enter the name of the command you want to add to a menu and the label you want to appear in the menu.

To save your menu definitions in a section file, use the SAVE EXTENDED EVE command or, if section file prompting is enabled, use the SAVE ATTRIBUTES command. To save your menu definitions in a command file, use SAVE ATTRIBUTES as described in Section 9.3.

9.5 Case-Exact Search

The new SET FIND CASE EXACT command specifies that searches always match the case of your search string exactly. This is particularly useful to find or replace lowercase occurrences only. For example, the following commands enable case-exact search and then find "digital" when it appears in lowercase only, skipping occurrences such as "Digital" or "DIGITAL":

```
Command: SET FIND CASE EXACT
Command: FIND digital
```

The default setting is SET FIND CASE NOEXACT—if you enter the search string in all lowercase, EVE searches for any occurrence; if you enter the search string in uppercase or mixed case, EVE searches for an exact match.

The setting applies to the FIND, REPLACE, and WILDCARD FIND commands. If you want case-exact search for future editing sessions, save the setting in your section file or command file. (See Section 9.3.)

9.6 Key Definitions

You can specify the keys on the mini keypad by their engraved labels as well as by their positional number (E1–E6), as follows:

Label on the Mini Keypad	EVE Key Name
Find	FIND or E1
Insert Here	INSERT_HERE or E3
Remove	REMOVE or E3
Select	SELECT or E4
Prev Screen	PREV_SCREEN or E5
Next Screen	NEXT_SCREEN or E6

You can abbreviate key names so long as your abbreviation is not ambiguous. For example, G–REM is a valid abbreviation for Gold–Remove. Note that G–R is an abbreviation for Gold–R. The case of letters does not matter in a key definition. For example, Gold–A and Gold–a are the same.

You can specify control keys using Ctrl, Control, or the circumflex character (^). For example, the following key names are the same:

Ctrl/A
Control/A
^A

Do not use the circumflex to specify a control key in combination with Gold or other modifier keys. For a list of the control keys defined by EVE, see the EVE online help topic called Control Keys.

With DECwindows, you can define shifted function keys and Alt key combinations. Shifted function keys combine holding down the Shift key on the main keyboard while you press a function key (such as as F14, Remove, PF4, or <⌫) or mouse button (such as MB2). With DECwindows, the Compose Character key on the keyboard serves as the Alt key. You can combine Alt with a function key, shifted function key, typing key, control key, mouse button, or Gold key combination. To enter a compose character sequence, use Alt/Space.

In specifying key combinations, use a slash (/), dash (–), or underscore (_) as a delimiter in the key name. For example, the following key names are the same:

Alt/A
Alt-A
Alt_A

As a convention, EVE shows key names (with the SHOW KEY or HELP KEYS command) using a slash for control keys, shifted function keys, and Alt key combinations and a dash for Gold key combinations. Thus, key combinations that require you to hold down one key (such as Ctrl) while pressing another key are shown with a slash; key combinations in which you press one key after another (such as Gold–Help) are shown with a dash.

For more information about using EVE key names, see the EVE online help topic called Names For Keys.

9.6.1 DECwindows-Style Function Keys

Use the SET FUNCTION KEYS DECWINDOWS command to enable DECwindows-style key definitions, as follows:

Key	DECwindows-Style Definition
Shift/<X]	ERASE CHARACTER. (In insert mode, erases the current character; in overstrike mode, replaces it with a space.)
Shift/Find	FIND NEXT
F12	START OF LINE
Shift/F12	END OF LINE
F13	EDT/WPS Delete Previous Word. (Erases all or part of the word left the cursor; at the start of a line, erases the line break for the previous line.)
Shift/F13	EDT/WPS Delete Word. (Erases from the current character to the end of the word; at the end of a line, erases the line break.)

This overrides the current definitions of the keys, whether EVE default, EDT keypad, or WPS keypad, but does not override definitions of your own. For example, if you defined the F13 key, your definition applies.

Remember that shifted function keys work only when you invoke EVE with /DISPLAY=DECWINDOWS qualifier. They do not work on character-cell terminals, such as a VT220 or VT100.

The default setting is SET FUNCTION KEYS NODECWINDOWS—there are no shifted function keys (other than any you defined), and the F12 and F13 keys are defined as follows:

Key	Non-DECwindows-Style Definition
F12	MOVE BY LINE or EDT Start Of Line
F13	ERASE WORD or EDT/WPS Delete Previous Word

9.6.2 Repeat Counts with Gold Key

With the EDT keypad or WPS keypad, you can press the Gold key (PF1) and type a number to repeat the next keystroke or command (much like using the REPEAT command). For example, to move the cursor down five lines, you can do the following:

1. Press the Gold key and type the number 5. The number appears in reverse video on the command line.
2. Press the down arrow key (↓).

The maximum repeat count is 32767. If you enter more than five digits, EVE displays an error message and sets the repeat count to 0.

You cannot use Gold–number combinations to repeat the following keys. Instead, use the REPEAT command to repeat the key.

- The Delete key, which lets you erase the repeat count in case you mistyped the number
- The EDT SpecIns key (Gold–KP3), which uses a Gold–number combination to specify the decimal value of the character to be inserted
- The WPS Paste key (Comma or Gold–Comma on the keypad or Insert Here on the mini keypad), which uses Gold–1 through Gold–9 to specify an optional WPS-style alternate paste buffer

9.6.3 WPS Ruler Keys

When you use the WPS keypad Ruler key (Gold–R), you can press KP4 to move to the next or previous indicator in the ruler, and press Gold–H to get help on ruler keys. These definitions are independent of how the keys are otherwise defined. For a list of the keys you can use with the WPS Ruler, see the EVE online help topic called Ruler Keys.

9.6.4 Mouse Buttons

You can define any mouse button, including MB1. If you press a mouse button that is undefined, such as Gold–MB1, EVE does the corresponding action for MB1:

1 click	Moves the cursor to where you are pointing and cancels any select range or found range.
2 clicks	Moves the cursor to where you are pointing and selects all of the word at that location.
3 clicks	Moves the cursor to where you are pointing and selects all of the line at that location.
4 clicks	Moves the cursor to where you are pointing and selects all of the paragraph at that location.
5 clicks	Moves the cursor to where you are pointing and selects all of the buffer.
Drag	Selects text. The cursor moves to where you are pointing when you release the mouse.

If you press a mouse button you have defined, EVE first executes a position cursor operation (moving the cursor to where you are pointing and canceling a select range or found range) and then executes whatever command is bound to that mouse button. For more information about defining mouse buttons, see the EVE online help topic called Mouse.

9.6.5 LEARN Sequences and Prompts

Under certain circumstances, some EVE commands reported informational messages that aborted a learn sequence. This has been changed as follows:

- With the cursor on an empty prompt line, the following commands (or keys defined for those commands) no longer abort a learn sequence:

  ERASE PREVIOUS WORD
  ERASE START OF LINE
  DELETE
  MOVE LEFT

- With the cursor positioned at the end of a prompt, the MOVE RIGHT command (or right arrow key) no longer aborts a learn sequence.

9.7 FILL and Paragraph Boundaries

Paragraph boundaries for FILL commands and for the WPS keypad Paragraph key (KP5) are any of the following:

- Blank line
- Top or bottom of the buffer
- Page break (form feed at the start of a line)
- DIGITAL Standard Runoff command (such as .LE; or .HL1) at the start of a line
- VAX DOCUMENT tag (such as <LE> or <EMPHASIS>) at the start of a line

When you fill a select range or found range, the FILL or FILL RANGE command does not reformat a line that begins with a page break, RUNOFF command, or VAX DOCUMENT tag but does reformat the other lines in the range. Filling a range does not delete blank lines.

9.8 Buffer List

When you use the SHOW BUFFERS or SHOW SYSTEM BUFFERS command, EVE puts the cursor on the name of the buffer you were in and highlights the name of that buffer in video bold. The list appears in a buffer named Buffer List. If you were in a buffer that is not in the list—for example, if you are in the DCL buffer or other EVE system buffer and then use the SHOW BUFFERS command to get a list of your text buffers—EVE puts the cursor on the name of the first buffer in the list.

To view a buffer, put the cursor anywhere on the line referring to that buffer and then use the SELECT or RETURN command—effectively, the same as using the BUFFER command withut having to type the buffer name. For example, you can press the Select key on the mini keypad, the Return key, or any key defined as SELECT or RETURN. Because EVE defines the Enter key on the keypad as RETURN (except with the VT100 keypad or WPS keypad), pressing Enter is the same as pressing Return. With DECwindows, you can use the mouse to point to the name of a buffer and click MB1.

To delete a buffer, put the cursor anywhere on the line referring to that buffer and then use REMOVE or CUT.

With DECwindows, you can select all or part of the Buffer List buffer by using the mouse as follows:

1. Point where you want to begin the selection.
2. Drag MB1 to select the text.
3. Use REMOVE or CUT to cut the selection, or use STORE TEXT or COPY to copy the selection.

This is useful if you want to paste all or part of the buffer list into another buffer—for example, if you want to save the list of buffers you were editing or the list of EVE system buffers.

9.9 Batch Editing

You can use EVE for batch editing by using the EDIT/TPU/NODISPLAY command. Typically, you also use the /INITIALIZATION or /COMMAND qualifier to specify a special initialization file or command file containing the editing operations you want EVE to perform. For example, the following DCL command file invokes EVE:

```
$ SET DEFAULT disk$1:[user.work]        ! go to work area
$ EDIT/TPU/NODISPLAY/INIT=batchedit     ! run EVE in batch
$ PURGE                                 ! clean up
$ RUNOFF/LOG/MESSAGE=USER ch1,ch2       ! create .MEM files
$ PRINT/AFTER=18:30 *.mem               ! print output
```

Example 9–2 shows the EVE initialization file (BATCHEDIT.EVE).

Some EVE commands cannot be used in batch mode. Generally, any command that requires a parameter, such as a file name or search string, will not work unless all the required information is given on the command line. Also, commands that prompt for a key press (such as REMEMBER) or keyword response (such as REPLACE) or that use a key press to exit from some special state (such as HELP) will not work. Using such a command might cause the the batch job to fail.

Example 9–2: EVE Initialization File for Batch Editing

```
GET FILE ch1.rno
!  find the placeholder for inserting the change
FIND {add info here}
! cut the placeholder
REMOVE
!  add the new information to CH1.RNO
INCLUDE FILE newinfo.txt
!  save the edits
WRITE FILE
!  similar change in CH2.RNO
GET FILE ch2.rno
FIND NEXT
REMOVE
TPU EVE$INSERT_TEXT (" See Chapter 1. ")
WRITE FILE
QUIT
```

Note that the initialization file or command file should end with an EXIT or QUIT command—that is, it should be a complete editing session. Because the EXIT and QUIT command can prompt you for additional input, all your edits should be written out before the EXIT or QUIT is executed.

For complex edits in batch mode, you should use a VAXTPU command file containing the procedures you want executed. Generally, you get greater precision or flexibility with VAXTPU procedures, particularly for complex reformatting, string replacements, and so on. For simpler edits, you can use an EVE initialization file.

For more information about invoking the editor from a DCL command file, see the *VAX Text Processing Utility Manual*.

9.10 Other Changes

The following sections describe other changes to EVE commands, keys, and features. For detailed information about these commands or keys, use EVE online help.

9.10.1 Bound Cursor Motion

When you enable bound cursor motion—with the SET CURSOR BOUND or SET KEYPAD WPS command—if the cursor is in an unused area of the buffer (or "whitespace"), EVE moves the cursor to the nearest text. This effect is called **snapping**.

9.10.2 Commands Buffer

The Commands buffer—the EVE system buffer that stores the commands you enter—is unmodifiable except for the line in the command window (that is, the command you are typing or the command line you have recalled). This ensures that cutting and pasting from the Commands buffer does not alter the command history of the editing session. For example, if you select and cut a line from the Commands buffer, EVE does a copy instead.

For more information about EVE command-line editing, see the EVE online help topic called Editing Command Lines.

9.10.3 SHOW Command

The output from the SHOW and SHOW DEFAULTS BUFFER commands includes information about journaling, paragraph indentation, word wrap, and other buffer settings. The following is typical output from the SHOW command:

```
EVE V2.4 1990-08-31 11:03

Information about buffer JABBER.TXT

Input file:   DISK$1:[USER.POEMS]JABBER.TXT;1
Output file:  DISK$1:[USER.POEMS]JABBER.TXT;1
Journal file: DISK$1:[USER]JABBER_TXT.TPU$JOURNAL;1

    Modified                    Left margin set to: 5
    Mode: Insert                Right margin set to: 72
    Paragraph indent: -4        WPS word wrap indent: none
    Write                       Modifiable
    Direction: Forward          Window width set to: 80
    28 lines

Tab stops set every 8 columns.

Word wrap: on

Marks:

slithy toves                            brillig
```

The status line when you use the SHOW command appears in reverse video as follows:

```
Buffer: SHOW                    To go back use RESET or DO
```

Thus, to return to the buffer you were in before the SHOW command, you can press the Do key and type any command, or use the RESET command (for example, by pressing Gold–Select).

9.10.4 RESTORE SELECTION Command with DECwindows Quick Copy

Use the RESTORE SELECTION command to insert the text you last erased with a pending delete operation or the text you moved with DECwindows Quick Copy functions (Ctrl/MB3 or Ctrl/MB3Drag). For more information, see the EVE online help topic called Quick Copy.

9.10.5 PREVIOUS BUFFER Command

Use the PREVIOUS BUFFER command to put your previous buffer into the current window, if you have two or more buffers. EVE then returns the cursor to your last position in that buffer. This lets you toggle between different buffers or cycle through several buffers without having to type the buffer names. It does not re-create a deleted buffer and it does not return you to an EVE system buffer.

If you have more than two buffers, the previous buffer is determined by the order in which you created the buffers. Conceptually, the list of buffers is circular, so that repeating PREVIOUS BUFFER cycles through your buffers. If you have only two buffers, the PREVIOUS BUFFER and NEXT BUFFER commands are the same.

9.10.6 Help Topics

EVE has several new Help topics covering the new commands and features. In particular, the help topic called Journal Files includes information about buffer change journaling and recovery, and there is a new help topic called Attributes that explains how to save global settings in a section file or command file.

With DECwindows, you can get help on menu items by combining the Help key and the mouse as follows:

1. Drag the pointer to the menu item you want help on, such as Global Attributes in the Customize menu.
2. Press and hold the Help key. Release the mouse button, and then release the Help key.

Similarly, you can get help on buttons within some dialog boxes, such as the ALL button in the Replace dialog box, as follows:

1. Point to the button you want help on.
2. Press and hold down the Help key. Click MB1 and then release the Help key.

In most cases, the HELP topic for a menu item or toggle button is the same as for the corresponding command. Command topics contain examples or a list of steps (or both) and also contain other information, such as any default key definitions for the command.

9.11 Program-Level Changes

The following sections describe EVE program-level changes. These are of interest if you use EVE as a base on which to build your own VAXTPU application. If you use EVE only as an editor, you need not read this section. For information about new features of VAXTPU, see Chapter 8.

9.11.1 Renamed Variable for the MAIN Buffer

The variable EVE$X_MAIN_BUFFER replaces the variable MAIN_BUFFER. This variable points to the initial buffer that is mapped to the main window on startup—either a buffer named after the input file specified on the command line or, if you did not specify a file when you invoked EVE, a buffer named MAIN.

If the buffer pointed to by EVE$X_MAIN_BUFFER is deleted (for example, by a DELETE BUFFER command), EVE does not reset the variable to point to another buffer. In this case, an application layered on EVE can reset the variable to point to another buffer.

9.11.2 Pre-Key and Post-Key Procedure Sharing

EVE lets layered applications use the pre-key and post-key procedure resources provided by VAXTPU. All requests to create or delete a key procedure from any key-map list must go through a single procedure, EVE$SET_KEY_PROCEDURE. (Previously, EVE allowed only a single application key procedure to be set on any key-map lists except the command key-map list, and key procedures for a layered application were not active while the EVE key procedure was active for any key-map list.)

9.11.3 Detached Cursor Handling

To handle a detached cursor, EVE uses the new VAXTPU built-in procedure SET (DETACHED_ACTION). A detached cursor condition occurs when the cursor position cannot represent the editing point in the current window, as follows:

- If the editing point is positioned to an invisible line, EVE moves the editing position to the next visible line.
- If the editing point is in a buffer that is not mapped to the current window, EVE moves the editing point to the buffer in the current window.
- If there is no current window, EVE re-initializes its windows and maps the current buffer to the current window.
- If the cursor is past the left or right edge of the current window, EVE takes no action.

For more information, see Chapter 8.

9.11.4 Status Line Processing

If you reduce the width of the EVE window, the status line shrinks as well, but EVE tries to keep the buffer-name field at full size unless the window is so narrow that the buffer name does not fit, in which case the field is truncated.

The other status line fields—direction, mode, read/write status—are not truncated. If any of these fields does not fit completely, it is simply omitted, beginning with the rightmost field (depending on the direction of the buffer).

For more information, see the EVE online help topic called Status Line.

9.11.5 EVE$INTERNATIONALIZATION Module

The EVE$INTERNALIZATION module has been added to the EVE sources (normally contained in SYS$EXAMPLES). This module facilitates translating EVE commands from English into some other language. For more information, see the comments in the EVE source files.

9.11.6 Obsolete Keywords for Message Constants

The following EVE keywords are no longer defined in EVE. These keywords were message constants used as parameters for the EVE$MESSAGE procedure. In writing your own procedures, do not use any such EVE message constants because these constants are not supported as part of the EVE public interface. The only exception to such use is for any message constant used by EVE as part of a call to an EVE$ routine or as a return status from an EVE$ routine. Digital will ensure that such constants remain totally upwards compatible.

EVE Keyword Message Constant	Numeric Value
EVE$_NODOCMSG	35959266
EVE$_SHOWBUFSSTATUS	35971419
EVE$_SHOWSTATUS	35971403
EVE$_HELPSTATUSDOWN	35971363
EVE$_HELPSTATUSUPDN	35971371
EVE$_HELPSTATUSN	35971339
EVE$_STATUSLINE	35971291
EVE$_PATSTATBIG	35971523
EVE$_DCLSTATUS	35971323
EVE$_HELPSTATUSNP	35971347
EVE$_HELPSTATUSUP	35971379
EVE$_HELPSTATUSP	35971355
EVE$_HELPSTATUS	35971331
EVE$_PATSTATBIGUPDN	35971507

The integer values of the message constants above are listed so that any code you have written that depends on these messages can continue to use the messages for this release. Use these integers in place of the constants that previously defined them.

Index

A

Action routine
  assigning for client messages, 8–35
  detached cursor, 8–57
Address
  converting to node address, 7–66
  converting to node name, 7–68
Attribute
  new for XUI toolkit, 6–5
Attribute (VAXTPU)
  modifiable record, 8–22
Attribute for DNS
  assigning, 7–4
  enumerating, 7–29
  modifying, 7–33
  reading, 7–35
  returning value, 7–77
  testing for one, 7–38
  types of, 7–12
Attribute for TPU
  setting records, 8–27
Attribute_Name identifier, 7–47
Attribute_Name_Str identifier, 7–47

B

Back-end converter, 6–9
  text, 4–3
  use of, 4–3, 4–15
Base
  specifying numeric constants, 8–55
Bookreader application, 3–7
Boolean identifier, 7–47
Buffer
  behavior of journal file, 8–17
Buffer (cont'd.)
  converting name to journal file name, 8–12
  end of buffer text
    visibility of, 8–21
  getting file name of journal, 8–11
  getting unmodifiable records erasible state, 8–22
  journal file, 8–6
  length (VAXTPU), 8–66
  recovering contents of, 8–13
  sensing journaling, 8–10
  sensing safe journaling, 8–11
  sensing unmodifiable records in, 8–23
  setting unmodifiable record erase state, 8–25
  unmodifiable
    effect of records, 8–22
Buffer change journaling, 8–6
  and keystroke journaling, 8–14
  converting buffer to journal file name, 8–12
  default file naming, 8–8
  enabling, 8–16
  getting enable state, 8–8
  getting file name of journal, 8–11
  getting information on journal file, 8–12
  naming algorithm, 8–8
  recovery, 8–13
  sensing, 8–10
  sensing safe state, 8–11
  specifying file name, 8–16
BUFFER_BEGIN keyword, 8–18, 8–62, 8–67, 8–68
BUFFER_END keyword, 8–18, 8–62, 8–67, 8–68

C

Calander application, 3–7
Calculator application, 3–7
Callable interface
  and default directory, 8–51
  item code TPU$_FILE_RECOVERABLE, 8–17
Callback data
  handling (VAXTPU), 8–46
Callback data structure
  of widget (VAXTPU), 8–46
Callback procedures
  UIL, 6–8
Cardfiler application, 3–8
CDA$CONVERT routine
  parameter changes, 6–10
CDA component, 6–9
CDA converter
  new processing options, 6–9
CDA converter library, 4–15
CDA new features
  CDA$_INPUT_FRONT_END_DOMAIN value, 6–10
  CDA$_INPUT_POSITION_PROCEDURE value, 6–10
  CDA$_OPTIONS_LINE processing option, 6–10
  CDA$_OUTPUT_BACK_END_DOMAIN value, 6–10
  CDA viewer routines, 6–10
  CDA viewers enhanced support, 6–10
  domain converter, 6–9
  DTIF support, 6–9
  DTIF_TO_DDIF format name, 6–9
  new values for standard-item-list, 6–10
CDA Viewer application, 3–8
CDA viewer routines
  CDA new feature, 6–10
  for character cell terminal, 6–11
  for DECwindows terminal, 6–11
CDA viewers, 6–10
CHANGE_CASE built-in procedure, 8–60
Child directory
  DNS, 7–5
Children
  getting information (VAXTPU), 8–40
Class
  getting widget information (VAXTPU), 8–41
  getting widget resource information (VAXTPU), 8–43
Class_Name identifer, 7–47
Class_Name_Str identifier, 7–47
Class_Version identifer, 7–47
Clearinghouse, 7–14
Client message
  assigning handler routine, 8–35
  fetching action routine for handling, 8–36
  sending from VAXTPU, 8–37
Client message (VAXTPU), 8–35
  handling, 8–35
  sensing type, 8–38
Client message handler
  getting routine, 8–36
Clock application, 3–8
Color mixing widget, 6–2
Command line
  /JOURNAL command qualifier, 8–6, 8–7
  /NODISPLAY command qualifier, 8–73
  /NOJOURNAL command qualifier, 8–7
  /RECOVER command qualifier, 8–6, 8–14
Compilation
  conditional in VAXTPU, 8–54
  lexical functions (VAXTPU), 8–54
Compiler
  VAXTPU enhancements, 8–52
Compound Document Architecture
  See CDA
Compound string routines
  list of, 6–3
Compound string text widget, 6–2
Conditional compilation
  in VAXTPU, 8–54
Condition value, 7–51 to 7–53
Confidence identifier, 7–47

Confidence level, 7–16
Constant
  specifying radix of, 8–55
Constraint argument
  defining, 6–8
  UIL, 6–8
Convenience routines
  list of, 6–3
CONVERT/DOCUMENT command, 4–2
  /FORMAT qualifier, 4–2
  /OPTIONS qualifier, 4–3
Copying
  of records
    effect on record attributes, 8–20
COPY_TEXT built-in
  effect on record attributes, 8–20
CREATE/TERMINAL command, 4–6
CREATE_BUFFER built-in procedure, 8–9, 8–13
CREATE_RANGE built-in procedure, 8–62
Creating the options file, 4–3
Cursor
  detached action routine (VAXTPU), 8–58
  detached handling (VAXTPU), 8–57
  detached routine (VAXTPU), 8–57
  getting detached reason (VAXTPU), 8–59
  watch-style
    in VAXTPU, 8–30
Cursor position
  relationship to invisible record, 8–21
CURSOR_HORIZONTAL built-in procedure
  operation, 8–21
CURSOR_VERTICAL built-in procedure
  operation, 8–21
Customize FileView menu
  changes, 3–2
Cut and Paste routines
  of XUI toolkit, 6–3

D

Data type
  UIL, 6–7
DCL commands
  CONVERT/DOCUMENT, 4–2
  CREATE/TERMINAL, 4–6
  SET DISPLAY, 4–12
  SHOW LICENSE, 4–12
  VIEW, 4–14
DDIF input file
  format, 4–15
DECnet event messages, 7–84
DECterm application, 3–8
DECwindows server
  transport, 2–2
DECwindows system, 2–1
  internationalization, 2–1
  MIT compliance, 2–2
  monitor independence, 2–1
  multiscreen, 2–2
DECwindows VAXTPU
  incompatible new features of, 8–3
Default directory
  and VAXTPU callable interface, 8–51
  getting (VAXTPU), 8–50
  setting (VAXTPU), 8–49
Default file naming algorithm
  buffer change journal, 8–8
DELETE built-in procedure
  journaling support, 8–10
Desktop applications, 3–6
Detached cursor
  defining handler routine, 8–57
  getting reason (VAXTPU), 8–59
  handling (VAXTPU), 8–57
Detached cursor action routine
  getting information, 8–58
Detached cursor flag
  TPU$K_DISJOINT, 8–60
  TPU$K_INVISIBLE, 8–60
  TPU$K_OFF_LEFT, 8–60
  TPU$K_OFF_RIGHT, 8–60
  TPU$K_UNMAPPED, 8–60

Device support
  driver, 6–12
Device Support
  SCSI, 6–12
Digital Command Language commands
  See DCL commands
Directory
  default setting (VAXTPU), 8–49
  DNS types, 7–5, 7–15
  enumerating in DNS, 7–30
  getting default (VAXTPU), 8–50
Display value
  getting, 8–24
  of VAXTPU window, 8–21
  setting for window, 8–24
  setting records, 8–27
  visability of VAXTPU record, 8–21
Display value attribute
  of VAXTPU record, 8–20
Distributed Name Service
  See DNS
DNS$APPEND_SIMPLENAME_TO_
    RIGHT routine, 7–56
DNS$COMPARE_FULLNAME routine,
    7–58
DNS$COMPARE_SIMPLENAME routine,
    7–60
DNS$CONCATENATE_NAME routine,
    7–62
DNS$CONTEXTVARNAME item, 7–50
DNS$CONTEXTVARTIME item, 7–50
DNS$COUNT_SIMPLENAMES routine,
    7–64
DNS$CVT_DNSADDRESS_TO_BINARY
    routine, 7–66
DNS$CVT_DNSADDRESS_TO_
    NODENAME routine, 7–68
DNS$CVT_NODENAME_TO_
    DNSADDRESS routine, 7–70
DNS$CVT_TO_USERNAME_STRING
    routine, 7–72
DNS$PARSE_USERNAME_STRING
    routine, 7–74
DNS$REMOVE_FIRST_SET_VALUE
    routine, 7–77
DNS$REMOVE_LEFT_SIMPLENAME
    routine, 7–80
DNS$REMOVE_RIGHT_SIMPLENAME
    routine, 7–82
DNS (Distributed Name Service), 7–1
  clearinghouse, 7–14
  overview, 1–2
  restrictions, 7–3
  root directory, 7–4
  wildcards, 7–9, 7–22
DNS call
  timeout in, 7–10
DNS clerk
  locating data in namespace, 7–26
  starting, 7–84
$DNS function code, 7–28 to 7–39
  converting from opaque, 7–33
  converting opaque name, 7–37
  converting string name, 7–34, 7–37
  creating an object, 7–28
  deleting an object, 7–29
  enumerating attributes, 7–29
  enumerating child directories, 7–30
  enumerating objects, 7–31
  enumerating soft links, 7–32
  modifying attribute, 7–33
  reading attribute, 7–35
  resolving soft link, 7–36
  testing a group, 7–38
  testing for attribute, 7–38
$DNS item code, 7–40 to 7–47
  arguments, 7–47 to 7–48
  attribute address, 7–44
  attribute name, 7–41
  attribute type, 7–40
  attribute value address, 7–45
  Boolean values, 7–42
  caching results, 7–43
  confidence level, 7–41
  converting names, 7–42, 7–44, 7–46
  entry type, 7–42, 7–43
  enumerating directories, 7–41
  enumerating functions, 7–41
  enumerating objects, 7–41
  member name, 7–43
  modifying attribute, 7–44

$DNS item code (cont'd.)
modifying attributes, 7–43
object class, 7–41
object name, 7–44
simple name address, 7–45
soft link name, 7–43
specifying groups, 7–42
suppressing namespace name, 7–46
target name address, 7–45
testing attribute value, 7–46
timeout value, 7–46
UID address, 7–46
version of object, 7–46
wildcard, 7–47
DNS name
case sensitivity, 7–9
comparing, 7–60
converting, 7–33, 7–34, 7–37
converting full name, 7–33
defining logicals, 7–7
format of, 7–3
source of, 7–3
DNS naming conventions
binary names, 7–9
format, 7–3
logical names, 7–7
quoted names, 7–9
syntax, 7–5
valid characters, 7–7
wildcards, 7–9
DNS object, 7–5
creating, 7–9 to 7–11, 7–28
deleting, 7–29
enumerating, 7–31
modifying, 7–12 to 7–14
reading attributes of, 7–19
DNS string name
converting, 7–37
converting to opaque, 7–34
format, 7–3
$DNS system service, 7–27
arguments, 7–27 to 7–50
building item list, 7–39
description, 7–50
format, 7–27, 7–50
function codes, 7–28
$DNS system service (cont'd.)
item code identifiers, 7–47
qualifying status, 7–49
returns, 7–27
status block, 7–27
$DNSW system service, 7–54
Domain converter, 4–4
CDA new feature, 6–9
processing options, 4–4
Driver, 6–12
DRM routines
list of, 6–3
DTIF support
CDA new feature, 6–9
DTIF_TO_DDIF format name
CDA new feature, 6–9

E

EDIT built-in procedure, 8–63
Editing point
relationship to invisible record, 8–21
EDIT/TPU command
/COMMAND qualifier, 9–14
input file handling, 9–2
/JOURNAL qualifier, 9–7
/NODISPLAY qualifier, 9–21
/NOJOURNAL qualifier, 9–6
/RECOVER qualifier
with buffer change journal file, 9–5
/SECTION qualifier, 9–12
%ELSE lexical keyword
compiling VAXTPU, 8–54
%ENDIF lexical keyword
compiling VAXTPU, 8–54
Entry_Type identifier, 7–47
Enumerate call
attributes, 7–29
directories, 7–30
objects, 7–31
soft links, 7–32
Enum_Att_Name identifier, 7–47
EQUIVALENCE statement
in VAXTPU, 8–52

ERASE_UNMODIFIABLE mode
and APPEND_LINE, 8–26
and CHANGE_CASE, 8–26
and COPY_TEXT, 8–26
and EDIT, 8–26
and ERASE (buffer), 8–26
and ERASE (range), 8–26
and ERASE_CHARACTER, 8–26
and ERASE_LINE, 8–26
and FILL, 8–26
and MOVE_TEXT, 8–27
and SPLIT_LINE, 8–27
and TRANSLATE, 8–27
Erasing unmodifiable records
VAXTPU, 8–25
Error messages
DNS, 7–85
EVE, 9–1 to 9–27
attributes, 9–8
saving in command file, 9–13
saving in section file, 9–11
saving system defaults, 9–14
batch editing, 9–21
bound cursor motion, 9–22
buffer change journaling, 9–3
buffers
Buffer List, 9–20
Commands, 9–23
case-exact search, 9–16
$CHOICES$ buffer
with input files, 9–2
with journal files, 9–5
code generated for saving attributes, 9–13
command file
saving attributes in, 9–13
TPU$COMMAND.TPU, 9–14
commands
DEFINE MENU ENTRY, 9–15
FILL, 9–20
PREVIOUS BUFFER, 9–24
RECOVER BUFFER, 9–5
RECOVER BUFFER ALL, 9–6
RESTORE SELECTION, 9–24
SAVE ATTRIBUTES, 9–9
with command file, 9–13
with section file, 9–11
EVE
commands (cont'd.)
SAVE EXTENDED EVE, 9–10
SAVE SYSTEM ATTRIBUTES, 9–9, 9–14
SET CURSOR BOUND, 9–22
SET DEFAULT COMMAND FILE, 9–10, 9–14
SET DEFAULT SECTION FILE, 9–10, 9–12
SET EXIT ATTRIBUTE CHECK, 9–10
SET FIND CASE EXACT, 9–16
SET FIND CASE NOEXACT, 9–16
SET FUNCTION KEYS DECWINDOWS, 9–17
SET FUNCTION KEYS NODECWINDOWS, 9–18
SET JOURNALING, 9–6
SET JOURNALING ALL, 9–6
SET NODEFAULT COMMAND FILE, 9–10
SET NODEFAULT SECTION FILE, 9–10, 9–12
SET NOEXIT ATTRIBUTE CHECK, 9–10, 9–11
SET NOJOURNALING, 9–6
SET NOJOURNALING ALL, 9–6
SET NOSECTION FILE PROMPTING, 9–10, 9–12, 9–14
SET SECTION FILE PROMPTING, 9–10, 9–12
SHOW, 9–23
SHOW BUFFERS, 9–20
SHOW DEFAULTS BUFFER, 9–23
SHOW SYSTEM BUFFERS, 9–20
UNDEFINE MENU ENTRY, 9–15
DECwindows function keys, 9–17
detached cursor handling, 9–26
input file handling, 9–2
journal file
deleting, 9–6
directory for, 9–4
naming, 9–5

EVE (cont'd.)
key names
abbreviating, 9–17
control keys, 9–17
for mini keypad, 9–16
keys
Alt combinations, 9–17
DECwindows-style, 9–17
Enter in buffer list, 9–20
Gold for repeat counts, 9–18
Return in buffer list, 9–20
shifted function, 9–17
with learn sequences, 9–20
with WPS Ruler, 9–19
keystroke journaling, 9–3
restrictions, 9–7
with software performance report, 9–8
learn sequence keys, 9–20
logical names
TPU$JOURNAL, 9–4
menus, 9–15
mouse
defining buttons, 9–19
in buffer list buffer, 9–20
paragraph boundaries, 9–20
programming
EVE$INTERNATIONALIZATION module, 9–26
EVE$SET_KEY_PROCEDURE, 9–25
EVE$X_MAIN_BUFFER variable, 9–25
obsolete keywords for messages constants, 9–26
post-key procedures, 9–25
pre-key procedures, 9–25
repeat counts with Gold key, 9–18
section file
saving attributes in, 9–11
status line, 9–26
WPS Ruler keys, 9–19
EVE Features
overview, 1–4
Event flag
$DNS system service, 7–27
Event messages
DNS, 7–84
Extensible VAX Editor
See EVE

F

File
default name for journaling, 8–8
File list
updating a, 3–3
FileView, 3–2
menus, 3–2
/FORMAT qualifier
for the CONVERT/DOCUMENT command, 4–2
for the VIEW command, 4–15
FORWARD keyword, 8–19
Front-end converter, 6–9
options file, 4–3
use of, 4–3, 4–15
Full name
converting to opaque, 7–34
converting to string, 7–33
Full_Name_String identifer, 7–48
Function keys
VAXTPU support for, 8–72

G

Gadget
pull-down menu entry, 6–2
Gadget routines
XUI toolkit, 6–5
GET_INFO (widget_variable) built-in procedure
behavior, 8–5
GET_INFO built-in procedure
"children" data, 8–40
"class" data, 8–41
"client_message", 8–38
"client_message_routine", 8–36
"default_directory", 8–50
"detached_action" data, 8–58
"detached_reason" data, 8–59

GET_INFO built-in procedure (cont'd.)
"is_managed" data, 8–42
"is_subclass" data, 8–42
"parent" data, 8–43
"resources" data, 8–43
string constant parameter
"display_value", 8–24
"erase_unmodifiable", 8–22
"journaling", 8–8, 8–10
"journal", 8–12
"journal_file", 8–8, 8–11
"journal_name", 8–12
"safe_for_journaling", 8–11
"unmodifiable_records", 8–23
string constant parameter for "widget_info", 8–5
use of, 8–8
Group_Member identifier, 7–48

H

HEIGHT *value* processing option
CDA new feature, 6–10

I

Icon
implementing in DECwindows VAXTPU, 8–32, 8–34
support DECwindows VAXTPU, 8–32
Icon box, 3–1
Icon creation (VAXTPU), 8–32, 8–34
%IFDEF lexical keyword
compiling VAXTPU, 8–54
%IF lexical keyword
compiling VAXTPU, 8–54
Initialization
incompatibility of in DECwindows VAXTPU, 8–3
Initialization coding
VAXTPU, 8–4
Initialization procedures
VAXTPU, 8–3
Insert mode
effecting record attributes, 8–20
Internationalization DECwindows, 2–1
Invisible record, 8–27
relationship to editing point, 8–21
Item code
TPU$_CHAIN, 8–72
TPU$_FILEIO, 8–71
TPU$_FILE_RECOVERABLE, 8–17
TPU$_GET_DEFAULT, 8–51
TPU$_SET_DEFAULT, 8–51

J

/JOURNAL command qualifier, 8–6, 8–7
Journal file, 8–14
default name, 8–8
getting characteristics of, 8–12
getting name of, 8–8
recovering buffer contents, 8–13
specifying with CREATE_BUFFER built-in, 8–9
Journaling
buffer change, 8–6
converting buffer to journal file name, 8–12
default file name, 8–8
EVE default behavior, 8–7
getting buffer change journaling enable state, 8–8
getting file name of buffer change journal, 8–11
getting journal file information, 8–12
layered application control, 8–7
recovery of buffer contents, 8–13
role of CREATE_BUFFER built-in, 8–9
role of source file, 8–15
sensing a safe buffer, 8–11
sensing buffer change journaling, 8–10
sensing the enable of keystroke journaling, 8–8
using both keystroke and buffer change journaling, 8–7
JOURNALING parameter
SET built-in procedure, 8–16
JOURNAL_OPEN built-in procedure, 8–7

K

Key
  modifiers (VAXTPU), 8–72
Keyboard
  KEY_NAME built-in procedure, 8–72
Keyboard keys
  expanded VAXTPU support, 8–72
Keys
  expanded VAXTPU support, 8–72
  function keys (F1 through F5), 8–72
Keystroke journaling
  and buffer change journaling, 8–14
  comparative to buffer change journaling, 8–6
  recovery, 9–7
  recovery restrictions, 9–7
  sensing the enable, 8–8
Keyword
  marking the range (VAXTPU), 8–62
  modifying string (VAXTPU), 8–63
  modify range, 8–67
  position (VAXTPU), 8–68
Keywords
  reserved in VAXTPU, 8–56
KEY_NAME built-in procedure, 8–72
KILL_SELECTION client message, 8–37

L

Left margin
  setting TPU records, 8–27
LENGTH built-in procedure, 8–66
License
  charge table, 4–12
  displaying an active, 4–12
LINE_BEGIN keyword, 8–62, 8–67
LINE_END keyword, 8–62, 8–67
Local variables
  compiling VAXTPU, 8–53
Lock manager limit, 5–1
Logical name, 7–7
  SYS$SCRATCH, 8–10
  TPU$JOURNAL, 8–10
Low End Graphics Subsystem (LEGGS), 6–12
Low-level widget routines
  list of, 6–5

M

Mail application, 3–8
Main array keys
  modifying and controlling, 8–72
Managed state
  sensing widget (VAXTPU), 8–42
Managing
  setting widget/screen mapping (VAXTPU), 8–45
MAP built-in procedure, 8–4
  enhancement to, 8–4
MAPPED_WHEN_MANAGED parameter to SET built-in procedure, 8–45
Mapping
  widget/screen (VAXTPU), 8–45
Margin
  setting TPU records, 8–27
Marker
  getting display value of marked record, 8–24
  sensing marked unmodifiable record, 8–23
Message (VAXTPU)
  sending, 8–37
MESSAGE built-in procedure, 8–66
MIT Release 3 compliance, 2–2
Modifiability
  of VAXTPU record, 8–22
  setting records, 8–27
Modifiability attribute
  of VAXTPU record, 8–20
Modifiable record, 8–22
MODIFY_RANGE built-in procedure, 8–67
Monitor independence, 2–1
MOVE_TEXT built-in
  effect on record attributes, 8–20
Multiple UIL callback procedures, 6–8
Multiscreen, 2–2

N

Name
  DNS
    See DNS name
Name service
  See DNS (Distributed Name Service)
Namespace, 7–2
  changing default, 7–84
  clearinghouses in, 7–14
  distributing, 7–14
  listing information, 7–22 to 7–25
  name of, 7–6, 7–48
  structure of, 7–4
  ways of using, 7–3
NCP executor, 5–1
  SET/DEFINE EXECUTOR command, 5–2
  SHOW EXECUTOR CHARACTERISTICS command, 5–2
Network name/object
  $QIO return, 6–13
Node name
  converting to address, 7–70
/NODISPLAY command qualifier
  enhancements to, 8–73
/NOJOURNAL command qualifier, 8–7
Notepad application, 3–9
Numeric constant
  specifying radix of, 8–55

O

Object
  See DNS object
Opaque name
  concatenating, 7–56, 7–62
  converting to string, 7–33, 7–37, 7–72
  converting user name, 7–74
  counting components, 7–64
  format of, 7–3
  returning simple name, 7–80, 7–82
Options file, 4–3
  CDA conversion, 4–3
/OPTIONS qualifier
  for CONVERT/DOCUMENT command, 4–3
OUTPUT parameter
  SET built-in procedure, 8–13
OVERRIDE_FORMAT processing option
  CDA new feature, 6–10
Overstrike mode
  effecting record attributes, 8–20

P

Paint application, 3–9
Parent
  getting widget information (VAXTPU), 8–43
Pattern enhancements
  BUFFER_BEGIN keyword, 8–18
  BUFFER_END keyword, 8–18
  reverse search, 8–19
Pattern search
  improvements, 8–18
Pixmap
  DECwindows icon (VAXTPU), 8–32
  use of to implent icon in DECwindows VAXTPU, 8–34
POSITION built-in procedure, 8–68
Processing options
  CDA converters, 6–9
  domain converter, 4–4
  Text back-end converter, 4–3
Programming, 6–1
  for network name/object number, 6–13
  in server, 6–11
  with CDA, 6–9
  with UIL, 6–7
  with XLIB, 6–6
  with XUI Toolkit, 6–1
Pull-down menu gadget, 6–2
Puzzle application, 3–9

Q

$QIO
  network naming return, 6–13

$QIO Calls, 6–11
Qualifier for UIL, 6–8

R

Radix
  specifiying numeric constants, 8–55
Range
  length (VAXTPU), 8–66
  sensing unmodifiable records in, 8–23
Read command, 3–2
READ_CHAR built-in procedure, 8–4
READ_KEY built-in procedure, 8–4
READ_LINE built-in procedure, 8–4
REALIZE_WIDGET built-in procedure, 8–45
Realizing
  widgets in VAXTPU, 8–45
Record
  attribute
    display value, 8–21
    effect of copying a record, 8–20
    effect of insert mode, 8–20
    modifiability, 8–22
    modifying, 8–20
    overstrike mode effect, 8–20
  getting display value of, 8–24
  getting unmodifiable erasible state, 8–22
  invisible
    relationship to editing point, 8–21
  sensing unmodifiable state, 8–23
  setting attributes, 8–27
  setting modifiability, 8–22
  setting unmodifiable erase state, 8–25
  visibility of, 8–21
/RECOVER command qualifier, 8–6, 8–14
Recovery
  keystroke journaling, 9–7
  of buffer contents, 8–6, 8–13
  role of CREATE_BUFFER built-in, 8–9
  role of source file, 8–15
  using keystroke journaling/not buffer change journaling, 8–14
RECOVER_BUFFER built-in procedure, 8–13
Resize event, 8–30
Resizing
  of windows and screens, 8–30
  screen height (VAXTPU), 8–30
  screen width (VAXTPU), 8–31
Resource
  getting widget class and data type information (VAXTPU), 8–43
REVERSE keyword, 8–19
Reverse search
  text pattern, 8–19
Routine
  CDA$CONVERT, 6–10
  CDA Viewer, 6–10
  XLIB, 6–6
  XUI Toolkit, 6–2
Run-time routines
  DNS, 7–55

S

SCAN built-in procedure, 8–19
SCANL built-in procedure, 8–19
Screen
  resizing, 8–30, 8–31
SCROLL built-in procedure
  operation, 8–21
SCSI (Small Computer System Interface)
  third-party support, 6–12
SCSI Features
  overview, 1–4
SCSI third-party device
  summary of VMS support for, 6–12
Searching pattern
  reverse, 8–19
SEND_CLIENT_MESSAGE built-in procedure, 8–37
Server
  $QIO Call programming, 6–11
  transport support, 2–2
Session manager, 3–2
SET (CLIENT_MESSAGE) built-in procedure, 8–35

SET (DEFAULT_DIRECTORY) built-in procedure, 8–49
SET (DETACHED_ACTION) built-in procedure, 8–57
SET (DISPLAY_VALUE) built-in procedure, 8–24
SET (ERASE_UNMODIFIABLE) built-in procedure, 8–25
SET (HEIGHT) built-in procedure, 8–30
SET (ICONIFY_PIXMAP) built-in procedure, 8–34
SET (ICON_PIXMAP) built-in procedure, 8–32
SET (JOURNALING) built-in procedure
  enhancements to, 8–16
SET (MAPPED_WHEN_MANAGED) built-in procedure, 8–45
SET (OUTPUT) built-in procedure, 8–13
SET (RECORD_ATTRIBUTE) built-in
  setting record modifiability, 8–22
SET (RECORD_ATTRIBUTE) built-in procedure, 8–27
SET (WIDGET_CALL_DATA) built-in procedure, 8–46
SET (WIDTH) built-in procedure, 8–31
SET/DEFINE EXECUTOR command, 5–2
SET DISPLAY command
  new qualifiers, 4–12
Set Protection command, 3–2
SHOW EXECUTOR CHARACTERISTICS command, 5–2
SHOW LICENSE command
  new qualifiers, 4–12
Simple name
  converting to opaque, 7–34
Simple_Name_Str identifier, 7–48
Skulk, 7–17
Small Computer System Interface
  See SCSI
Soft link
  DNS, 7–5
  enumerating, 7–32
  locating target entry, 7–36
SOFT_DIRECTIVES processing option
  CDA new feature, 6–10
Source file
  defined, 8–15
SPAN built-in procedure, 8–19
SPANL built-in procedure, 8–19
SS$_TOOMUCHDATA message, 6–13
Standard-item-list new values
  CDA new feature, 6–10
Startup
  incompatibility of DECwindows VAXTPU, 8–3
STUFF_SELECTION client message, 8–37
Subclass
  sensing widget information (VAXTPU), 8–42
SUBSTR built-in procedure, 8–68
Synonyms
  creating for VAXTPU, 8–52
SYS$DNS system service
  See $DNS system service
SYS$SCRATCH logical name
  and VAXTPU journaling, 8–10
SYSGEN parameters, 5–1
System features, 2–1
System management features, 5–1
System service, 7–26

T

Task Output Box, 3–3
Terminal emulator
  creating, 4–6
Text back-end converter
  processing options, 4–3
%THEN lexical keyword
  compiling VAXTPU, 8–54
TPU
  See VAXTPU
TPU$$GET_DEFAULT_DIRECTORY routine, 8–51
TPU$$SET_DEFAULT_DIRECTORY routine, 8–51
TPU$INITIALIZE routine
  item codes, 8–51
  item code TPU$_CHAIN, 8–72

TPU$INITIALIZE routine (cont'd.)
item code TPU$_FILEIO, 8–71
item code TPU$_FILE_RECOVERABLE, 8–17
TPU$JOURNAL logical name, 9–4
VAXTPU journaling, 8–10
TPU$_FILE_RECOVERABLE item code, 8–17
TPU$_UNKLEXICAL error message, 8–56
TRANSLATE built-in procedure, 8–69
Transport, 2–2

U

UIL Compiler, 6–7
callback procedure, 6–8
constraint argument, 6–8
data type, 6–7
version qualifer, 6–8
Unbound code
compiling local variables, 8–53
UNMAP built-in procedure, 8–5
enhancement to, 8–4
Unmodifiable record, 8–22, 8–27
getting erasible state, 8–22
sensing, 8–23
setting erase state, 8–25
User Interface Language
See UIL

V

VAX Text Processing Utility
See VAXTPU
VAXTPU
callable interface, 8–17
contants, 8–55
documents related to, 8–3
error message, 8–56
incompatibility of DECwindows initialization/startup, 8–3
journaling methods, 8–6
keywords reserved, 8–56
overview, 1–2, 8–1
VIEW command
/FORMAT qualifier, 4–15
new qualifiers, 4–14
Visibility
getting display value of record/window, 8–24
sensing record display value, 8–24
setting record, 8–27

W

Watch cursor, 8–30
Widget
color mixing, 6–2
compound text string, 6–2
creating window (VAXTPU), 8–45
getting children information (VAXTPU), 8–40
getting class and data type resource information (VAXTPU), 8–43
getting parent information (VAXTPU), 8–43
getting subclass information (VAXTPU), 8–42
mapping screen (VAXTPU), 8–45
new XUI toolkit attributes, 6–5
realizing (VAXTPU), 8–45
sensing class (VAXTPU), 8–41
sensing the managed state (VAXTPU), 8–42
setting screen map (VAXTPU), 8–45
using callback data structure (VAXTPU), 8–46
WIDTH *value* processing option
CDA new feature, 6–10
Wildcard
DNS, 7–9, 7–22
Window
creating (VAXTPU widget), 8–45
getting display value of, 8–24
resizing, 8–30
setting display value of, 8–24
Window manager, 3–1
Work in Progress Box, 3–3
WRITE_FILE built-in
journaling support, 8–17

X

XLIB
  new routines, 6–6
X library
  See XLIB
XUI Toolkit
  new routines, 6–2
  new widgets, 6–1
X User Interface Toolkit
  See XUI Toolkit

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]	————	SSB Order Processing - WMO/E15 *or* 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

VMS Version 5.3
New Features Manual
AA–MG29B–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

VMS Version 5.3
New Features Manual
AA–MG29B–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