HP OpenVMS Version 8.2 New Features and Documentation Overview

Order Number: BA322-90003

January 2005

This manual describes the new features associated with the OpenVMS Alpha and I64 Version 8.2 operating systems and provides an overview of the documentation that supports this software.

Revision/Update Information:	This is a new manual.
Software Version:	OpenVMS I64 Version 8.2 OpenVMS Alpha Version 8.2

Hewlett-Packard Company
Palo Alto, California

ZK6675

The HP OpenVMS documentation set is available on CD.

This document was prepared using DECdocument, Version 3.3-1b.

Contents

Preface ix

Part I OpenVMS Version 8.2 New Features

1 Summary of HP OpenVMS Version 8.2 New Features

1.1 Summary Table 1–1
1.2 Major Differences Between OpenVMS Alpha and OpenVMS I64 Systems 1–4

2 General User Features

2.1 DCL Commands and Lexical Functions 2–1
2.2 License Management Facility Enhancements (LMF) 2–2
2.3 Monitor Utility Enhancements 2–3
2.4 OpenVMS I64 Operating Environments (OEs) 2–3

3 System Management Features

3.1 OpenVMS I64 Boot Manager (BOOT_OPTIONS.COM) Utility 3–1
3.2 Clustering on OpenVMS I64 Systems 3–1
3.2.1 OpenVMS I64 Cluster Interconnect Support 3–2
3.2.2 Mixed-Architecture Clusters 3–3
3.2.2.1 Storage in a Mixed-Architecture Cluster 3–4
3.3 EFI for OpenVMS Utilities 3–5
3.4 HP Performance Data Collector (TDC) 3–5
3.5 Ethernet LAN Drivers: Full-Duplex or Half-Duplex Mode Mismatch 3–6
3.5.1 Result of Duplex Mode Mismatch 3–6
3.5.2 Detection and Correction of Duplex Mode Mismatch 3–7
3.6 Host-Based Adapter (HBA) Support 3–7
3.6.1 Fibre Channel HBA Support on OpenVMS I64 and OpenVMS Alpha Systems 3–7
3.6.2 Ultra SCSI HBA Support on OpenVMS I64 Systems 3–8
3.7 System Analysis Tools Enhancements 3–8
3.7.1 New and Enhanced SDA Commands 3–8
3.7.2 System Service Logging 3–9
3.8 System Parameters 3–9
3.8.1 New System Parameters 3–10
3.8.2 Changed System Parameters 3–11
3.9 Additional Time Zones Added to Database 3–12
3.10 Volume Shadowing for OpenVMS New Features 3–12

4 Programming Features

4.1 Analyze Utility Enhancements (I64 Only) 4–1
4.2 OpenVMS Calling Standard Changes for OpenVMS I64 4–1
4.3 Checksum Utility 4–1
4.3.1 CHECKSUM/OBJECT Enhanced for I64 Objects 4–2
4.4 C Run-Time Library Enhancements 4–2
4.4.1 File-Locking Functions 4–2
4.4.2 Standard-Compliant stat Structure 4–2
4.4.3 File-System Statistics Support 4–3
4.4.4 fcntl File Status Flags 4–3
4.4.5 UNIX Style Pipe Support 4–3
4.4.6 DECC$POPEN_NO_CRLF_REC_ATTR 4–3
4.4.7 glob and globfree 64-Bit Support 4–3
4.4.8 socketpair 4–4
4.5 DCE RPC Now Supports IEEE Floating-Point Type 4–4
4.6 OpenVMS Debugger 4–4
4.6.1 Intel® Itanium® Hardware Support 4–4
4.6.2 OpenVMS I64 Language Support 4–5
4.6.3 Heap Analyzer Available on OpenVMS I64 Systems 4–5
4.7 Extended Lock Value Block 4–5
4.8 Librarian Utility and Library Routines (I64 only) 4–5
4.8.1 Librarian Usage Summary 4–6
4.8.2 Changes to the Librarian Utility 4–6
4.8.2.1 Librarian Defaults to Intel® Itanium® Architecture 4–6
4.8.2.2 No Support for /ALPHA and /VAX Qualifiers 4–7
4.8.2.3 Enhanced /REMOVE Qualifier 4–7
4.8.3 Changes to the Library (LBR) Routines 4–7
4.8.3.1 New Library Types Added 4–7
4.8.3.2 Accessing ELF Object Libraries 4–8
4.8.4 New Librarian (LBR) Routines for ELF Object Libraries 4–9
LBR$LOOKUP_TYPE 4–10
LBR$MAP_MODULE 4–12
LBR$PUT_MODULE 4–13
LBR$UNMAP_MODULE 4–14
4.8.5 Extended Library (LBR) Routines for ELF Object Libraries 4–15
LBR$DELETE_DATA 4–16
LBR$DELETE_KEY 4–18
LBR$GET_INDEX 4–20
LBR$INSERT_KEY 4–23
LBR$LOOKUP_KEY 4–25
LBR$PUT_RECORD 4–27
LBR$REPLACE_KEY 4–29
LBR$SEARCH 4–31
4.8.6 Library Format Changed due to New UNIX-Style Weak Symbols 4–34
4.8.6.1 New ELF Type for Weak Symbols 4–34
4.8.6.2 Version 6.0 Library Index Format 4–34
4.8.6.3 New Group Section Symbols 4–35
4.8.6.4 Precedence Ordering Rules 4–35
4.9 Linker Utility 4–35
4.10 HP OpenVMS Migration Software 4–35
4.11 POSIX Threads Features 4–36
4.11.1 Lowercase Symbol Names for /NAMES=AS_IS Compilation 4–36

4.11.2 Support for Process-Shared Mutexes and Condition Variables 4–36
4.11.3 SET and SHOW Commands Enhanced (I64 Only) 4–36
4.11.4 New Routine Added to Thread Independent Services API 4–36
tis_mutex_init_type 4–37
4.12 New RTL LIB Routines 4–39
4.12.1 Change to LIB$GETDVI routine (I64 only) 4–40
4.13 New RTL OTS Routines 4–40
4.14 Patch Utility Now Available on OpenVMS Alpha and OpenVMS I64 4–41
4.15 New and Revised System Services 4–41
4.16 Time Zone Information Compiler (zic) Updates 4–44
4.17 Traceback Facility 4–44
4.18 XDELTA New Features 4–46

5 Associated Products Features

5.1 ATI RADEON 7500 Graphics 5–1
5.2 Common Data Security Architecture (CDSA) Now Supports a New Encryption Type 5–1
5.3 Kerberos for OpenVMS 5–2
5.4 HP SSL for OpenVMS 5–3
5.5 HP TCP/IP Services for OpenVMS Version 5.5 5–5

6 Host-Based Minimerge (HBMM) in Volume Shadowing for OpenVMS

6.1 HBMM Configuration Requirements 6–1
6.2 HBMM Restrictions 6–2
6.2.1 Cluster Configuration Restrictions 6–2
6.2.2 Shadow Set Member Restrictions 6–2
6.2.3 System Parameter Restrictions 6–2
6.3 HBMM in a Mixed-Version or Mixed-Architecture OpenVMS Cluster System 6–2
6.4 Overview of Full Merge and Minimerge Operations 6–4
6.4.1 Merge Resulting from a System Failure 6–4
6.4.2 Merge Resulting from Mount Verification Timeout 6–5
6.4.3 Merge Resulting from Use of SET SHADOW/DEMAND_MERGE 6–5
6.4.4 Comparison of Merge and Minimerge Operations 6–5
6.5 Overview of HBMM 6–6
6.5.1 Bitmaps: Master and Local 6–6
6.5.2 HBMM Policies 6–7
6.6 HBMM Policy Specification Syntax 6–7
6.7 Rules Governing HBMM Policies 6–8
6.8 Guidelines for Establishing HBMM Policies 6–10
6.8.1 Selecting the Systems to Host Master Bitmaps 6–11
6.8.2 Setting the Bitmap RESET_THRESHOLD Value 6–11
6.8.3 Using Multiple Policies 6–12
6.9 Configuring and Managing HBMM 6–13
6.9.1 How to Define an HBMM Policy 6–13
6.9.2 How to Assign an HBMM Policy to a Shadow Set 6–13
6.9.3 How to Activate HBMM on a Shadow Set 6–13
6.9.4 How to Disable HBMM on a Shadow Set 6–14
6.9.5 How to Remove a Policy Association from a Shadow Set 6–14
6.9.6 How to Change a Policy Assignment of a Shadow Set 6–14
6.9.7 How to Delete a Named Policy from the Cluster 6–14
6.9.8 How to Apply a Changed DEFAULT Policy 6–15

6.9.9 How to Display Policies 6–15
6.9.10 How to Display the Merge Status of Shadow Sets 6–17
6.9.11 How to Prevent Merge Operations on a System 6–18
6.9.12 Considerations for Multiple-Site OpenVMS Cluster Systems 6–18
6.10 New System Parameters That Affect HBMM 6–18
6.10.1 SHADOW_REC_DLY Parameter 6–19
6.10.2 SHADOW_HBMM_RTC Parameter 6–19
6.11 Use of /DEMAND_MERGE When HBMM Is Enabled 6–19
6.12 Prioritizing Merge and Copy Operations 6–20
6.12.1 Default Management of Merge and Copy Operations 6–20
6.12.2 Hierarchy of Transient State Operations 6–21
6.12.3 Assigning Priorities 6–21
6.12.4 Displaying Priority Values 6–22
6.12.5 Controlling Which Systems Manage Merge and Copy Operations 6–23
6.12.6 Managing Merge Operations 6–23
6.12.7 Managing Copy Operations 6–24
6.12.8 Managing Transient States in Progress 6–25
6.13 Visible Impact of Transient State Events 6–25

7 Linker Utility

7.1 Linker Utility New Features 7–1
7.1.1 Differences When Linking on OpenVMS I64 Systems 7–2
7.1.1.1 Data Types of Symbols must Match on I64 7–2
7.1.1.2 Specifying Based Clusters Vary by OpenVMS Platform 7–4
7.1.1.3 Handling of Initialized Overlaid Program Sections on OpenVMS I64 Systems 7–4
7.1.1.4 Behavior Difference When Linking ELF Common Symbols 7–7
7.1.1.5 Flags Set When /TRACEBACK, /DEBUG, and /DSF are Used 7–8
7.1.2 New Aspects for Linking on OpenVMS I64 Systems 7–9
7.1.2.1 Understanding Linkage Messages 7–10
7.1.2.2 Considerations for Images Compiled with Reduced Floating Point Model 7–12
7.1.2.3 Considerations for Linking with ELF Groups and UNIX-style Weak Symbols 7–12
7.1.3 New Linker Qualifiers for OpenVMS I64 7–13
7.1.3.1 New /BASE_ADDRESS Qualifier 7–13
7.1.3.2 New /SEGMENT_ATTRIBUTE Qualifier 7–14
7.1.3.3 New /FP_MODE Qualifier 7–14
7.1.3.4 New Linker Qualifiers: /EXPORT_SYMBOL_VECTOR and /PUBLISH_GLOBAL_SYMBOLS 7–14
7.1.3.5 New GROUP_SECTIONS and SECTION_DETAILS keywords for the /FULL Qualifier 7–17
7.1.4 New Linker Options for OpenVMS I64 7–17
7.1.4.1 New Alignments for the PSECT_ATTRIBUTE Option 7–17
7.1.5 New Ways to Use Existing Linker Qualifiers and Options 7–17
7.1.5.1 Mixed-Case Arguments in Linker Options on I64 Systems 7–18
7.1.5.2 Conventions for Specifying Image Names 7–18
7.1.5.3 Using the PSECT_ATTRIBUTE Option to Specify Alignment 7–19
7.1.5.4 Special Linker Handling of Nonexistent Files 7–19
7.1.6 New OpenVMS I64 Linker Map 7–21

Part II OpenVMS Documentation

8 OpenVMS Documentation Overview

9 OpenVMS Printed and Online Documentation

9.1 Printed Documentation 9–1
9.1.1 OpenVMS Media Kit Documentation 9–2
9.1.2 OpenVMS Documentation Sets 9–2
9.1.3 Operating Environments Extensions Documentation Set (I64 Only) 9–4
9.1.4 Documentation for System Integrated Products 9–5
9.1.5 Archived OpenVMS Documentation 9–5
9.2 Authoring Tool for OpenVMS Documentation 9–5
9.3 Online Documentation on CD 9–6
9.3.1 Online Formats 9–6
9.3.2 PDF Reader 9–6
9.4 Online Documentation on the OpenVMS Web Site 9–6
9.5 Online Help 9–6

10 Descriptions of OpenVMS Manuals

10.1 Manuals in the OpenVMS Media Kit 10–1
10.2 Manuals in the OpenVMS Base Documentation Set 10–2
10.3 Additional Manuals in the OpenVMS Full Documentation Set 10–3
10.4 RMS Journaling Manual 10–7
10.5 Manuals in the OpenVMS I64 OE Extensions Kit 10–8
10.6 Archived Manuals 10–9

Index

Examples

7–1 Linker Map Showing Program Section Synopsis 7–6

Figures

3–1 OpenVMS Cluster Systems with Alpha and I64 Systems 3–3
3–2 Storage in Mixed-Architecture OpenVMS Cluster 3–4
7–1 Object and Image Synopsis and Cluster Synopsis 7–22
7–2 Image Segment Synopsis 7–23
7–3 Program Section Synopsis 7–24
7–4 Program Section Synopsis (Continued) 7–25
7–5 Symbol Cross Reference 7–26
7–6 Symbols by Value 7–27
7–7 Image Synopsis 7–28
7–8 Link Run Statistics 7–29

Tables

1–1	Summary of OpenVMS Version 8.2 Software Features	1–1
2–1	Updates to DCL Commands and DCL Documentation	2–1
2–2	Updates to DCL Lexicals and Lexicals Documentation	2–2
4–1	Libraries Created by OpenVMS Platforms	4–6
4–2	RTL LIB Routines	4–39
4–3	RTL OTS Routines	4–40
4–4	New System Services	4–41
4–5	Revised System Services	4–42
6–1	Visible Impact of Transient State Events	6–26
8–1	Documentation Set Changes for OpenVMS Version 8.2	8–1
9–1	OpenVMS Media Kit Manuals	9–2
9–2	OpenVMS Full Documentation Set (QA-001AA-GZ.8.2/BA554MN)	9–3
9–3	System Integrated Products Documentation	9–5
10–1	Archived OpenVMS Manuals	10–9
10–2	Archived Networking Manuals and Installation Supplements	10–10

Preface

Intended Audience

This manual is intended for general users, system managers, and programmers who use the HP OpenVMS operating system.

This document describes the new features related to Version 8.2 of the OpenVMS operating system. For information about how some of the new features might affect your system, read the release notes before you install, upgrade, or use Version 8.2.

Document Structure

This manual contains the following parts and chapters:

- Part I, OpenVMS Version 8.2 New Features
  - Chapter 1 summarizes the new OpenVMS software features.
  - Chapter 2 describes new features of interest to general users of the OpenVMS operating system.
  - Chapter 3 describes new features that are applicable to the tasks performed by system managers.
  - Chapter 4 describes new features that support programming tasks.
  - Chapter 5 describes significant layered product new features.
  - Chapter 6 provides information on Host-Based Minimerge in Volume Shadowing for OpenVMS.
  - Chapter 7 provides an overview of new linker functionality in OpenVMS Version 8.2 as well as differences and considerations you should review before linking programs on OpenVMS I64 systems.
- Part II, OpenVMS Documentation
  - Chapter 8 describes the OpenVMS documentation changes from the previous version.
  - Chapter 9 describes how the documentation is delivered.
  - Chapter 10 describes each manual in the OpenVMS documentation set.

Related Documents

For additional information about HP OpenVMS products and services, visit the following World Wide Web address:

`http://www.hp.com/go/openvms`

Reader’s Comments

HP welcomes your comments on this manual. Please send comments to either of the following addresses:

Internet	**openvmsdoc@hp.com**
Postal Mail	Hewlett-Packard Company OSSG Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

How to Order Additional Documentation

For information about how to order additional documentation, visit the following World Wide Web address:

`http://www.hp.com/go/openvms/doc/order`

Conventions

The following conventions may be used in this manual:

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 and then press and release another key or a pointing device button.
$\boxed{\text{Return}}$	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.) In the HTML version of this document, this convention appears as brackets, rather than a box.
. . .	A horizontal ellipsis in examples 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 command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.

|	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold type	Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
italic type	Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TYPE	Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Example`	This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX commands and pathnames, PC-based commands and folders, and certain elements of the C programming language.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

Part I

OpenVMS Version 8.2 New Features

1 Summary of HP OpenVMS Version 8.2 New Features

OpenVMS Version 8.2 introduces a major new platform release for the OpenVMS operating system, HP Integrity servers. OpenVMS continues to deliver the highest levels of availability, scalability, flexibility, performance, and security that are required for operating in a 24x365 environment. With more than 20 years of proven reliability, OpenVMS continues to enhance its availability and performance by including new technology in the base operating system and in the OpenVMS Cluster software environment.

1.1 Summary Table

OpenVMS Version 8.2 includes all the capabilities of OpenVMS Version 7.3–2 plus the new features added to the OpenVMS operating system. Table 1–1 summarizes each feature provided by OpenVMS Version 8.2 and presents these features according to their functional component (general user, system management, programming, and associated products).

Table 1–1 Summary of OpenVMS Version 8.2 Software Features

General User Features	
Documentation	Documentation updates are now available on the Internet.
	One new manual is provided with this release, *Porting Applications from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers*.
	Documentation for OpenVMS Volume Shadowing support for Host-Based Minimerge (HBMM) is provided in Chapter 6. Documentation for the Linker utility new features is provided in Chapter 7.
DCL commands and lexical functions	Version 8.2 provides new and changed DCL commands, qualifiers, and lexical functions.
EV7 chip speedup	Support for new EV7 chip speedup for AlphaServer ES47, ES80, and GS1280 systems.
LMF enhancements	Added support for Integrity server OE packages and per-processor licensing on I64 systems.
Monitor utility enhancements	Several enhancements were made to the Monitor utility during the port.
Operating Environments	OpenVMS I64 is provided in three different operating environments: Foundation Operating Environment (FOE), Enterprise Operating Environment (EOE), and Mission Critical Operating Environment (MCOE).

(continued on next page)

Table 1–1 (Cont.) Summary of OpenVMS Version 8.2 Software Features

System Management Features	
Clusters	With few exceptions, OpenVMS Cluster software provides the same features on OpenVMS I64 systems as it currently offers on OpenVMS Alpha and VAX systems.
The Performance Data Collector (TDC)	Gathers performance data for systems running OpenVMS Version 7.3–2 or later.
OpenVMS I64 Boot Manager utility	A menu-based utility that allows you to easily manage EFI boot options on an Integrity server running OpenVMS I64.
EFI for OpenVMS Utilities	Utilities that provide device management functions at the EFI console on I64 systems.
Fibre Channel HBA support	Two new Fibre Channel host-based adapters (HBAs) are supported in OpenVMS Version 8.2, one for OpenVMS I64 systems and one for OpenVMS Alpha systems.
Host-Based Minimerge (HBMM)	HBMM improves merge operations by decreasing the number of comparisons that are needed to complete a merge operation.
System Dump Analyzer	New commands and qualifiers have been added and changes have been made to the System Dump Analyzer. Some changes are to accommodate I64 systems.
Prioritizing the merge and copy operations of volume shadowing	You can now prioritize shadow sets for merge and copy operations on a per-system basis, using two new SET SHADOW command qualifiers and a new system parameter. In addition, you can manipulate where a merge or copy operation will occur.
New system parameters	— ERLBUFFERPAG_S2 specifies the amount of S2 space memory to allocate for each S2 space error log buffer. — ERRORLOGBUFF_S2 specifies the number of S2 space error log buffers reserved for system error log entries. — SCSI_ERROR_POLL causes OpenVMS to send a SCSI Test Unit Ready command every hour to each SCSI disk. — SHADOW_ENABLE is a special parameter reserved for HP use. — SHADOW_HBMM_RTC controls the interval the system waits between the checking of reset thresholds for shadow sets that have Host-Based Minimerge (HBMM) bitmaps. — SHADOW_PSM_DLY allows system managers to adjust the delay that shadowing adds. — SHADOW_REC_DLY helps determine the length of time a system waits before it attempts to manage recovery operations on shadow sets. — SHADOW_SITE_ID allows a system manager to define a site value, which Volume Shadowing uses to determine the best device to perform reads. — SYSSER_LOGGING enables logging of system service requests for a process. — VHPT_SIZE controls the number of kilobytes to allocate for the Virtual Hash Page Table (VHPT) on each CPU in the system.
Time zones	Additional time zones are now supported by adding them to the database.
Ultra SCSI adapter support on OpenVMS I64	OpenVMS I64 supports two Ultra-SCSI host based adapters (HBAs): the Ultra-160 dual channel and the Ultra-320 dual channel.
Programming Features	
ANALYZE utility enhancement (I64 only)	Analyzes Executable and Linkable Format (ELF) object and image files.

(continued on next page)

Table 1–1 (Cont.) Summary of OpenVMS Version 8.2 Software Features

Programming Features	
HP C Run-Time Library (CRTL) enhancements	— File-locking functions — Standard-compliant stat structure — File-system statistics support — fnctl file status flags — UNIX style pipe support — New logical name, DECC$POPEN_NO_CRLF_REC_ATTR — glob and globfree 64-bit support — Socketpair — stat function enhancements
Calling Standard	Supports the Intel Itanium calling standard on OpenVMS I64 systems, with some exceptions.
Checksum utility	CHECKSUM/OBJECT command enhanced for I64 objects.
DCE RPC	Supports both G_FLOAT and IEEE floating-point type on both OpenVMS Alpha and OpenVMS I64 systems.
Extended Lock Value Block	Adds support for 64-byte lock value blocks in the OpenVMS Lock Manager. Existing applications will continue to use a 16-byte value block. Usage of the extended value block requires source changes.
HP OpenVMS Migration Software	Converts Alpha executable images or shareable images into a translated image that runs on an OpenVMS I64 system.
New RTL LIB$ routines	See Section 4.12 for routines and their descriptions.
New RTL OTS$ routines	See Section 4.13 for routines and their descriptions.
Patch utility	Patch utility is now available on OpenVMS Alpha and OpenVMS I64 systems.
POSIX threads features	– Process-shared mutexes and condition variables – Enhancements to SET and sHOW commands (I64 only) – New routine, tis_mutex_init_type, added to thread independent services
New system services	— $CLEAR_UNWIND_TABLE — $GET_UNWIND_ENTRY_INFO — $GOTO_UNWIND_64 — $IEEE_SET_ROUNDING_MODE — $IEEE-SET PRECISION_MODE — $RPCC_64 — $SET_RETURN_VALUE — $SET_UNWIND_TABLE
Traceback Application Programming Interface for OpenVMS I64	Allows a user application access to Traceback information.

(continued on next page)

Table 1–1 (Cont.) Summary of OpenVMS Version 8.2 Software Features

Associated Products Features	
Common Data Security Architecture (CDSA) Version 2.1	Supports new encryption type.
Kerberos for OpenVMS	Kerberos Version 2.1 for OpenVMS is based on MIT Kerberos V5 Release 1.2.6 with CERT patches up to Release 1.2.8. Support for both Kerberos clients and servers is provided on OpenVMS I64, OpenVMS Alpha, and OpenVMS VAX. New features in Kerberos for OpenVMS Version 2.1 include the `ktutil` command, which invokes a menu from which an administrator can read, write, or edit entries in a Kerberos V5 keytab or V4 srvtab file.
HP OpenVMS Management Station Version 3.2-D	OpenVMS Management Station Version 3.2-D is included with OpenVMS Alpha Version 8.2.
HP SSL for OpenVMS	HP SSL Version 1.2 is based on OpenSSL 0.9.7d with fixes to security vulnerabilities reported by openssl.org. Support for HP SSL is provided on OpenVMS I64, OpenVMS Alpha, and OpenVMS VAX. New features in HP SSL Version 1.2 include OCSP (Online Certificate Status Protocol), AES (Advanced Encryption Standard), and Elliptic Curve cryptography.
HP TCP/IP Services for OpenVMS	Version 5.5 of this product is supported on OpenVMS Version 8.2.
UNIX Portability Features	UNIX portability features are introduced for easier porting of UNIX applications to OpenVMS.
Open Source Tools for OpenVMS	The Open Source Tools CD, included with OpenVMS Version 8.2, has a number of utilities and sources to expedite porting from UNIX to OpenVMS.

1.2 Major Differences Between OpenVMS Alpha and OpenVMS I64 Systems

The following list describes some of the very pronounced differences between Alpha and I64 platforms:

- Consoles and console line connections

  The Integrity server allows console line connections by two different serial lines on current Integrity platforms. One serial line connects to the BMC console and the other to an optional Management Processor (MP). In addition, the Management Processor allows for a network connection and TELNET access for even more management flexibility.

  Both consoles allow for management of hardware and can perform functions such as resetting the system or powering on or off the system. The Management Processor is the preferred method, because it provides more management features than the BMC console.

- Connecting to the system console

  The BMC and MP consoles allow for connection to the system console. For Integrity servers, this is the EFI shell. The mechanism for setting up boot devices and boot options are very different from the Alpha SRM console. In addition, once the system is booted, the EFI shell is no longer available. On Alpha, you can still get back to the SRM console while the system is booted.

- Bootable devices for Integrity servers

  Bootable devices for Integrity servers have a FAT partition with the initial bootstrap programs.The EFI shell can find and read this partition. The system is booted by running an initial bootstrap program called vms_loader.efi, contained in the FAT partition.

  There is no environment variable that specifies the boot device. Instead, the EFI shell allows for the creation of menu items that execute vms_loader.efi on a specific disk.

- Console environment variables on Integrity servers

  The boot flags can be specified by *vms_flags*. On Alpha, the name BOOT_OSFLAGS is used. There are no environment variable to configure SCSI or LAN devices. Any LAN configuration settings would need to be done within the LANCP program.

- Licensing

  While the basic LMF management functions remain the same, new licensing practices for OpenVMS I64 differ from what is available for OpenVMS Alpha and OpenVMS VAX. The primary differences involve:

  - The way license units are assigned and counted
  - The types of licenses offered
  - Update licenses:
    - For I64 systems, update licenses are not offered. A customer must be an active Software Updates Service customer to be eligible to receive new revisions of software or the customer must purchase the product again. Information regarding Software Updates service and other HP Service offerings are available at:

      `http://www.hp.com/hps/software`
    - For Alpha systems, update licenses continue to be available for non-Service customers or Service customers whose contract has lapsed.

  Refer to the *HP OpenVMS License Management Utility Manual* for information about the licensing for OpenVMS I64 systems.

- Privileged architecture library code (PALcode) functions moved into OpenVMS

  OpenVMS Alpha calls PALcode to execute atomic functions such as memory barrier (MB), Halt and Bugcheck. These functions are now part of OpenVMS I64 instead of in the console firmware.

- OpenVMS Calling Standard

  The implementation of the OpenVMS Calling Standard on the Intel® Itanium® processor family is based on the Intel Calling Standard. OpenVMS extensions were added for compatibility with VAX and Alpha code.

  If your current application has detailed knowledge of the internals of the calling standard format, refer to the *HP OpenVMS Calling Standard* for information about the OpenVMS Calling Standard implementation for OpenVMS Version 8.2.

- IEEE floating-point format

  AlphaServers support VAX floating-point data types and IEEE floating-point data types in hardware. Integrity servers support IEEE floating-point in hardware but VAX floating-point data types are supported in software. The OpenVMS I64 compilers provide the /FLOAT_D_FLOAT and /FLOAT_G_FLOAT qualifiers to enable you to produce VAX floating-point data types. If you do not specify one of these qualifiers, IEEE floating-point data types are used.

  Refer to *Porting Applications from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers* manual for more information about IEEE floating-point data types as they apply to OpenVMS Version 8.2.

Prior to getting started with an Integrity server system, read the *HP OpenVMS Version 8.2 Release Notes*, *HP OpenVMS Version 8.2 Upgrade and Installation Manual*, and this manual.

2
General User Features

This chapter provides information about new features for all users of the HP OpenVMS Alpha and OpenVMS I64 operating systems.

2.1 DCL Commands and Lexical Functions

Table 2–1 and Table 2–2 summarize new and changed DCL commands, qualifiers, and lexical functions for OpenVMS Version 8.2. For more information, refer to online help or the *HP OpenVMS DCL Dictionary*.

Table 2–1 Updates to DCL Commands and DCL Documentation

DCL Command	Documentation Update
ANALYZE/IMAGE	New qualifiers: /FLAG_VALUES, /SECTIONS, /SEGMENTS, /SELECT
ANALYZE/OBJECT	New qualifiers: /FLAG_VALUES, /SECTIONS, /SELECT
ANALYZE/SSLOG	New command. Refer to the *HP OpenVMS System Analysis Tools Manual* for details.
APPEND	New /BLOCK_SIZE qualifier.
ASSIGN	New /CLUSTER_SYSTEM qualifier.
CHECKSUM	New command.
COPY	New /BLOCK_SIZE qualifier.
CREATE/MAILBOX	New command.
DEASSIGN	New /CLUSTER_SYSTEM qualifier.
DEFINE	New /CLUSTER_SYSTEM qualifier.
DELETE 'file'	New /GRAND_TOTAL qualifier.
DELETE/BITMAP	New command.
DELETE/MAILBOX	New command.
DIRECTORY	New keyword VERSION for /SELECT.
INITIALIZE	New /GPT qualifier, new keywords for /ERASE, changes to /LIMIT, and /CLUSTER_SIZE default raised to 16.
OPEN	New /NOSHARE qualifier.
PATCH	Former VAX-only command now runs on all three platforms.
PURGE	New /GRAND_TOTAL qualifier.
SEARCH	New qualifiers: /LIMIT, /SKIP, /WILDCARD_MATCHING
SET BOOTBLOCK	New command (for I64 only).
SET DISPLAY	Additional values for /TRANSPORT and /PMTRANSPORT to support Internet Protocol version 6 (IPv6).

(continued on next page)

Table 2–1 (Cont.) Updates to DCL Commands and DCL Documentation

DCL Command	Documentation Update
SET IMAGE	New command.
SET PROCESS	New /TOKEN and /SSLOG qualifiers and updated /RESOURCE_WAIT qualifier.
SET SERVER	Revised /CONFIGURE description. SET SERVER is now documented as three unique commands, which makes online help more user friendly.
SET SHADOW	Several additions to support Host-Based Minimerge.
SET TERMINAL	New /BACKSPACE qualifier.
SHOW DEVICES	Support added to display data in bytes when /FULL is specified.
SHOW FASTPATH	New command.
SHOW IMAGE	New command.
SHOW LICENSE	New /HIERARCHY and /OE qualifiers.
SHOW LOGICAL	New /CLUSTER qualifier; expanded display for /FULL.
SHOW PROCESS	New /CASE_LOOKUP and /TOKEN qualifiers; revised example for Red Sox fans.
SHOW SERVER	SHOW SERVER is now documented as two unique commands, which makes online help more user friendly.
SHOW SHADOW	Several additions to support host-based minimerge.
SHOW SYSTEM	New /IMAGE qualifier.
SHOW TERMINAL	New field at end of display.
SHOW WORKING_SETS	New capability to display in bytes.
WRITE	New /WAIT[/NOWAIT] qualifier.

Table 2–2 Updates to DCL Lexicals and Lexicals Documentation

DCL Lexical	Documentation Update
F$FID_TO_NAME	New lexical function.
F$GETDVI	New **pathname** argument and many new item codes.
F$GETJPI	New TOKEN item code.
F$GETSYI	List of item codes updated.
F$LICENSE	New lexical function.
F$MULTIPATH	New lexical function.
F$UNIQUE	New lexical function.

2.2 License Management Facility Enhancements (LMF)

The License Management Facility has been updated to support two new business practices for OpenVMS on I64 systems: operating environments (OEs) and per-processor licensing (PPL).

Operating environments include the operating system and bundled applications in an integrated package. Currently, three different operating environments are available for OpenVMS I64 systems:

- HP OpenVMS Foundation Operating Environment (FOE)
- HP OpenVMS Enterprise Operating Environment (EOE)

- HP OpenVMS Mission Critical Operating Environment (MCOE)

New qualifiers to LMF commands allow you to manage your operating environment licenses. One license now turns on FOE, EOE, or MCOE depending on which one you have purchased. This change reduces complexity of LMF management and improves flexibility of operations.

A new license type—per-processor or PPL—is required to run the operating environments. You need a license for each active processor in your I64 system.

Refer to the *HP OpenVMS License Management Utility Manual* manual for more information about licensing.

2.3 Monitor Utility Enhancements

The Monitor utility has been ported as a native utility for both OpenVMS Alpha and OpenVMS I64 systems. Several enhancements were made to the utility during the port:

- Several classes of data (such as DISK) now utilize the entire screen height when displaying information.
- The SYSTEM class now displays the number of "current" processes in Process States. Previously, current processes were grouped in the "Other" category.
- The format of the recorded data file has changed with this release. The format changes were made to improve the alignment of the recorded data. For information about the Supplemental MONITOR Information - Record Formats, see the appendix in *HP OpenVMS System Management Utilities Reference Manual*. These changes do not allow pre-V8.2 systems to read the new format of a recorded data file. There is, however, a utility in SYS$EXAMPLES which allows a new format data file to be converted to the pre-V8.2 format. The utility is called SYS$EXAMPLES:MONITOR_CONVERT.C. An executable is also provided. To use the utility, use the following command:

  ```
  $ mc system$examples:monitor_convert input-file output-file
  ```
- Various performance improvements were made to the MONITOR data collection routines to reduce system overhead when using the Monitor utility.

2.4 OpenVMS I64 Operating Environments (OEs)

OpenVMS Version 8.2 introduces a new way of providing the OpenVMS operating system and its layered products with the OpenVMS I64 system. Unlike OpenVMS Alpha, the OpenVMS I64 operating system provides three different packages available on Integrity servers:

- HP OpenVMS Foundation Operating Environment (FOE) is an Internet-ready feature-rich offering with leading price and performance.
- HP OpenVMS Enterprise Operating Environment (EOE) delivers enhanced manageability functions, single-system availability, and performance.
- HP OpenVMS Mission Critical Operating Environment (MCOE) delivers the highest levels of multi-system availability and workload management.

All three operating environments are included on one DVD. Your license agreement determines to which operating environment you have access.

Note

These OpenVMS I64 operating environments are licensed on a per-processor basis (PPL) and not on system capacity. Alpha licensing remains unchanged.

For a comprehensive list of technical specifications, go to the Software Products Description Web site:

`http://www.hp.com/info/spd`

For more information about HP OpenVMS Operating Environments, contact your HP Sales Representative.

3
System Management Features

This chapter provides information about new features, changes, and enhancements for system managers.

3.1 OpenVMS I64 Boot Manager (BOOT_OPTIONS.COM) Utility

The OpenVMS I64 Boot Manager (BOOT_OPTIONS.COM) utility is a menu-based utility that allows you to easily manage EFI boot options on an Integrity server running OpenVMS I64. The utility allows you to:

- Set your Integrity server with a boot option for your system disk, dump device, or debug device
- Display OpenVMS boot entries
- Determine the position of the entry in the EFI Boot Manager; for example, you can ensure that your system disk is first on the option list so that it boots automatically when the system is powered on or rebooted
- Set boot flags for the entry
- Remove an entry
- Set or disable the EFI timeout (the time the EFI Boot Manager waits before booting the first or next available entry on the boot list)
- Validate boot entries
- Configure boot, dump, and debug devices while OpenVMS is running. To configure boot devices, you do not have to shut down the operating system and enter commands at the console as you do on Alpha systems.

After installing OpenVMS I64, HP recommends using the the utility to add your system disk as the first boot option in the EFI Boot Manager list. This utility is required for configuring booting on Fibre Channel storage devices; it is optional for all other devices. Because it is so easy to use, HP recommends using this utility rather than the EFI Boot Manager wherever possible. For information on configuring Fibre Channel devices, refer to the *Guidelines for OpenVMS Cluster Configurations* manual. For more information on the OpenVMS Boot Manager utility, refer to the *HP OpenVMS System Manager's Manual*.

3.2 Clustering on OpenVMS I64 Systems

With few exceptions, OpenVMS Cluster software provides the same features on OpenVMS I64 systems as it currently offers on OpenVMS Alpha and VAX systems.

Key OpenVMS Cluster features include:

- Fully shared, multiple-node read/write disk access
- Clusterwide file system

- Clusterwide batch/print queue subsystem
- Distributed lock manager
- Votes/quorum-based membership management
- Single security domain
- Single system management domain
- Rich, clusterwide API
- Mixed-architecture clusters
- Support for rolling upgrades
- Support for multiple interconnects (see Section 3.2.1)
- Support for a maximum of sixteen systems in a mixed-architecture cluster, eight of which can be I64 systems
- Failover and load balancing
- Cluster network alias
- Disk and tape serving
- Disaster-tolerant capabilities with support for distances up to 500 miles (800 kilometers) using Disaster-Tolerant Cluster Services (DTCS)

Satellite booting is not supported in this release. It is planned for a future release.

3.2.1 OpenVMS I64 Cluster Interconnect Support

Ethernet, Fast Ethernet, and Gb Ethernet can be used for cluster communications (SCS traffic) on OpenVMS I64 systems. However, FDDI and ATM, which are supported for cluster communications on OpenVMS Alpha systems, are not supported on OpenVMS I64 systems.

While FDDI and ATM adapters are not supported as cluster interconnects on OpenVMS I64 systems, they are supported as inter-site interconnects in a multiple-site cluster. You can use bridges or switches to connect the OpenVMS I64 node's FastEthernet/GigabitEthernet NIC(s) to any inter-site interconnect the WAN supplier provides, such as T3, E3, SONET, ATM, FDDI, DWDM, or others.

OpenVMS Cluster software supports the following three proprietary cluster interconnects on Alpha systems, but they are not supported on OpenVMS I64 systems: DSSI (DIGITAL Systems Storage Interconnect), CI (Cluster Interconnect), and MEMORY CHANNEL.

Although DSSI and CI are not supported on OpenVMS I64 systems, data stored on DSSI and CI disks connected to Alpha systems can be served to OpenVMS I64 systems in the same cluster.

Fibre Channel is supported as a shared-storage cluster interconnect on OpenVMS I64 systems but SCSI is not. (SCSI as a shared-storage cluster interconnect is also not supported for OpenVMS Alpha systems for the recent SCSI adapters.)

However, data stored on SCSI disks directly attached to either OpenVMS I64 systems or to OpenVMS Alpha systems can be served to any other members of the cluster. This is also true for any locally attached storage in an OpenVMS Cluster system.

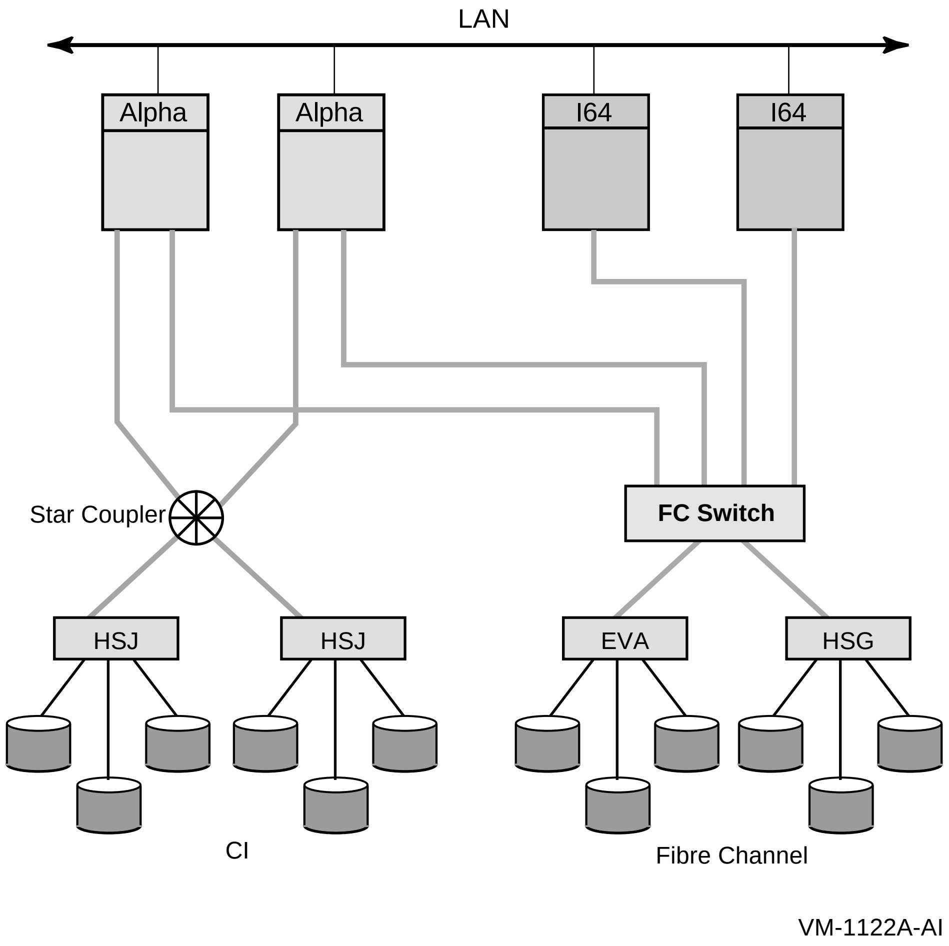
3.2.2 Mixed-Architecture Clusters

OpenVMS supports both OpenVMS Alpha and OpenVMS I64 systems in a mixed-architecture cluster. The OpenVMS Alpha version supported in this configuration is OpenVMS Alpha Version 7.3–2. Mixed-version support for all these versions requires the installation of one or more remedial kits, as described in the *HP OpenVMS Version 8.2 Release Notes*. See the following web site for the HP OpenVMS Version 8.2 documentation set:

`http://www.hp.com/go/openvms/doc`

Figure 3–1 shows an OpenVMS Cluster system to which OpenVMS I64 systems have been added.

Figure 3–1 OpenVMS Cluster Systems with Alpha and I64 Systems



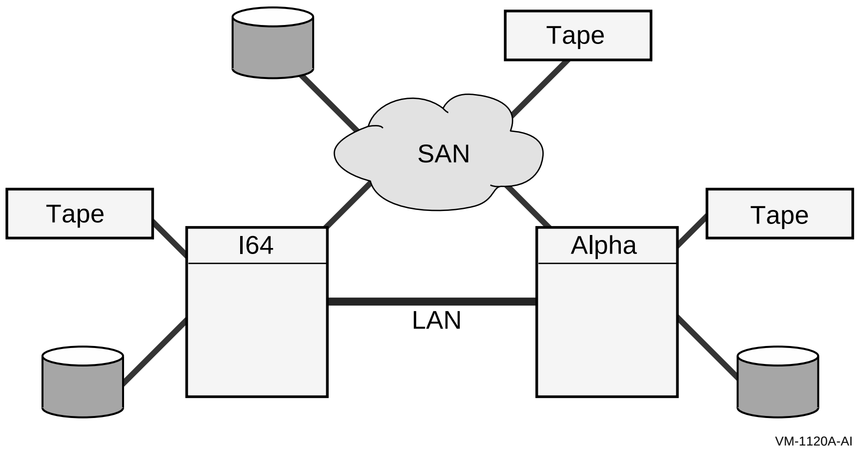
A LAN interconnect is used for cluster communications for all systems in the cluster. In this configuration, the same Fibre Channel storage can be accessed by both OpenVMS Alpha and OpenVMS I64 systems at the same time. Note that I64 systems directly connected to the Fibre Channel disks can be served data from the CI disks. In an OpenVMS mixed-architecture cluster, each architecture requires a minimum of one system disk. For this release, up to eight I64 systems are supported in a cluster. In a mixed-architecture cluster, this means you can include up to eight I64 systems with Alpha systems so that the total number of systems does not exceed sixteen.

3.2.2.1 Storage in a Mixed-Architecture Cluster

This section describes the rules pertaining to storage, including system disks, in a mixed-architecture cluster consisting of OpenVMS I64 and OpenVMS Alpha systems.

Figure 3–2 is a simplified version of a mixed-architecture cluster of OpenVMS I64 and OpenVMS Alpha systems with locally attached storage and a shared Storage Area Network (SAN).

Figure 3–2 Storage in Mixed-Architecture OpenVMS Cluster



I64 systems in a mixed-architecture OpenVMS Cluster system:

- Must have an I64 system disk, either a local disk or a shared Fibre Channel disk.
- Can use served Alpha disks and served Alpha tapes.
- Can use SAN disks and tapes.
- Can share the same SAN data disk with Alpha systems.
- Can serve disks and tapes to other cluster members, both I64 and Alpha systems.

Alpha systems in a mixed-architecture OpenVMS Cluster system:

- Must have an Alpha system disk, which can be shared with other clustered Alpha systems.
- Can use locally attached tapes and disks.
- Can serve disks and tapes to both I64 and Alpha systems.
- Can use I64 served data disks.
- Can use SAN disks and tapes.
- Can share the same SAN data disk with I64 systems.

3.3 EFI for OpenVMS Utilities

EFI Utilities provide device management capabilities for Integrity servers with OpenVMS I64 systems. These utilities interact with the EFI Shell. The commands that invoke them must be issued from \efi\vms at the EFI Shell> prompt, after shutting down the OpenVMS operating system:

- VMS_BCFG: Adds a boot entry to the EFI Boot Manager, allowing you to specify an OpenVMS device name for the entry. (HP recommends using the OpenVMS I64 Boot Manager (BOOT_OPTIONS.COM) utility for this purpose.)
- VMS_SET: Sets the dump device and the debug device to the specified OpenVMS device name.
- VMS_SHOW: Displays the equivalent OpenVMS device name for devices mapped by the EFI console.

For more information, refer to the EFI utilities chapter in the *HP OpenVMS System Management Utilities Reference Manual*.

3.4 HP Performance Data Collector (TDC)

HP Performance Data Collector for OpenVMS (TDC V2.1) is available for use with OpenVMS Version 8.2. The Performance Data Collector (TDC) can be used to collect approximately 1100 system performance metrics from Alpha and I64 systems for analysis by other application software.

Metrics that are provided include the following:

- Cache and memory utilization and performance
- Cluster configuration and communications
- CPU utilization
- Disk utilization and performance
- Distributed Lock Manager performance
- Distributed Transaction Manager performance
- File system performance
- Networking hardware and software performance
- Process metrics
- Miscellaneous system performance metrics (for example, paging, swapping, and faulting)
- System parameter settings

A run-time-only variant of the Performance Data Collector (TDC_RT V2.1) is installed with OpenVMS Version 8.2. The run-time variant provides a data collector application and support files. The collector application does not run automatically; however, a suitably privileged user can start and stop it manually.

A downloadable kit provides a Software Developer Kit (SDK) as well as run-time environments for all supported system configurations:

Platform	OpenVMS Version
Alpha systems	Version 7.3-2 or Version 8.2
I64 systems	Version 8.2

The SDK provides both a programmer manual that documents the TDC Application Programming Interface (API) and C header files and sample code. The API can be used to develop software to integrate TDC with other applications in various ways, including:

- Extracting data from a TDC data file for analysis
- Feeding data "live" to another application as the data is collected by TDC, without first storing the data in a file
- Supplementing the metrics provided by TDC with other metrics of interest, in a fully integrated and supported fashion

Software built using the SDK will work with any runtime environment provided either by the TDC_RT kit, which is distributed and installed with OpenVMS, or by the full TDC kit.

The downloadable full kit and additional documentation are available at the following web site:

```
http://h71000.www.7hp.com/openvms/products/tdc/
```

3.5 Ethernet LAN Drivers: Full-Duplex or Half-Duplex Mode Mismatch

Ethernet LAN drivers can operate in full-duplex or half-duplex mode under one of the following conditions:

- On Alpha systems, according to the console environment variable setting
- On Alpha or I64 systems, according to the LANCP device database setting
- If autonegotiation is enabled, in negotiation with the switch or link partner

For any of these conditions, if the duplex mode is set incorrectly, a duplex mode mismatch condition occurs.

An example of a duplex mode mismatch condition is the following:

A LAN device is set to operate in full-duplex mode at 100 megabits per second. However, the switch port is set to autonegotiate. The switch port determines the speed correctly, but selects half-duplex mode.

3.5.1 Result of Duplex Mode Mismatch

When a duplex mode mismatch condition occurs, the end of the link in full-duplex mode transmits whenever transmit data is available. It transmits without checking whether an incoming receive packet already occupies the link. This situation results in transmit and receive errors.

Any error represents a lost packet, which requires the application to do one of the following:

- Perform error recovery to detect and retransmit the packet.
- Have the link partner retransmit the packet.

Depending on the application, you might observe a significant degradation in performance. Therefore, it is important to detect and correct this condition.

3.5.2 Detection and Correction of Duplex Mode Mismatch

Ethernet LAN drivers for all full-duplex-capable LAN devices have been modified to detect and report this condition so that a system manager can correct it. Each driver checks error counters periodically. If it appears that a duplex mismatch condition exists, the driver displays the following console message:

```
%EWA0, Possible duplex mode mismatch condition detected
```

In addition, the LAN driver makes an error log entry that you can identify by the type OxDD (if the error log viewer does not decode the entry into English). The LAN drivers make other error log entries for link-up and link-down transitions.

You can decipher error log entries by searching LAN driver error logs for the following error types:

Error Type	Description
OxCA	Connection available (link up)
OxCD	Connection down (link down)
OxDD	Dubious duplex (possible duplex mismatch)

Note that each error log entry has the same format, and the type code is in the same location.

The error log entry and console message are repeated every hour until the condition is resolved.

You can use LANCP or ANALYZE/SYSTEM to gather more information from device counters.

3.6 Host-Based Adapter (HBA) Support

The following sections describe the HBA support in OpenVMS Version 8.2:

- Fibre Channel Host-Based Adapter
- Ultra SCSI Host-Based Adapter

3.6.1 Fibre Channel HBA Support on OpenVMS I64 and OpenVMS Alpha Systems

OpenVMS I64 Version 8.2 supports only the dual-port 2GB/1GB Fibre Channel Universal PCI-X HBA (A6826A) for external connection to Fibre Channel storage. This HBA is supported by a new driver, the PGQDRIVER, which has been implemented as a port driver to the existing DKDRIVER. The PCI-X 1-port FCA2404 2GB adapter (AB232A or KGPSA-EA, aka LP9802 for AlphaServers) supported on the OpenVMS I64 V8.1 Evaluation Release for Integrity servers is no longer supported on OpenVMS V8.2 or later. For external Fibre Channel storage, customers should plan accordingly to replace AB232A or KGPSA-EA FC adapters with the A6826A adapter.

OpenVMS Alpha Version 8.2 supports the new Fibre Channel HBA, the LP10000. The LP10000 is supported on the AlphaServer DS and ES server families. The LP10000 is available in a single-channel or dual-channel version.

The LP10000 is supported on earlier versions of OpenVMS Alpha (Version 7.3-1, and Version 7.3-2) by means of patch kits and by firmware console V6.6.

The version-specific patch kits with a root name of FIBRE_SCSI will be available at the following website:

`http://h71000.www7.hp.com/serv_support.html`

Under Service tools, select either Patches for OpenVMS or FTP site for OpenVMS patches.

For more detailed information about these HBAs, refer to the hardware documentation for your HP Integrity server or for your adapter.

3.6.2 Ultra SCSI HBA Support on OpenVMS I64 Systems

OpenVMS I64 V8.2 supports the following Ultra SCSI HBAs on HP Integrity server systems:

- Ultra-160 SCSI dual channel (A6829A)—on HP rx4640 Integrity servers
- Ultra-320 SCSI dual channel (A7173A)—embedded in HP Integrity rx2600 server and HP Integrity rx1600 server

External SCSI storage requires the addition of an Ultra-160 dual-port SCSI adapter (A6829A), which can be housed in DS2100, MSA30, or 4200/4300 series shelves.

Note

OpenVMS does not support shared SCSI storage using these adapters in an OpenVMS Cluster system consisting of either OpenVMS I64 systems only, or a mix of OpenVMS I64 systems and OpenVMS Alpha systems.

These adapters are also supported by HP on the following operating systems from HP: HP-UX, Linux, and the Microsoft XP 64-bit operating system.

For more detailed information, refer to the hardware documentation that accompanies your HP Integrity server.

3.7 System Analysis Tools Enhancements

The System Analysis Tools have been enhanced with new commands and new features for use with both Alpha and I64 systems. The new commands and features make it easier to analyze an OpenVMS system.

For more detailed information, refer to the *HP OpenVMS System Analysis Tools Manual*.

3.7.1 New and Enhanced SDA Commands

The following commands have been enhanced or are new for I64 use:

- EVALUATE
- EXAMINE
- FLT
- FORMAT
- READ

- SET CPU
- SET OUTPUT
- SHOW
  - CALL_FRAME
  - CBB
  - CEB
  - CPU
  - CRASH
  - DEVICE
  - EXCEPTION_FRAME
  - EXECUTIVE
  - GLOBAL_SECTION_TABLE
  - GST
  - IMAGE
  - KFE
  - PAGE_TABLE
  - PARAMETER
  - PROCESS
  - STACK
  - SWIS
  - TQEIDX
  - UNWIND
- VALIDATE TQEIDX
- WAIT

3.7.2 System Service Logging

System Service Logging records information about system service activity within a process. It is intended for use when troubleshooting a system.

Logging is turned on with the SET PROCESS/SSLOG=(STATE=ON) command. Logging is stopped with the SET PROCESS/SSLOG=(STATE=UNLOAD) or (STATE=OFF) command. Logged information is displayed with the ANALYZE/SSLOG command. SSLOG.DAT is the default file where logged information is captured.

For additional information, see the *HP OpenVMS System Analysis Tools Manual*.

3.8 System Parameters

The system parameters in the following sections are new in Version 8.2. The last section lists the system parameters that have been changed in Version 8.2. For more detailed information, refer to the *HP OpenVMS System Management Utilities Reference Manual*.

3.8.1 New System Parameters

The following parameters are new in OpenVMS Version 8.2:

- ERLBUFFERPAG_S2

  ERLBUFFERPAG_S2 specifies the amount of S2 space memory to allocate for each S2 space error log buffer requested by the ERRORLOGBUFF_S2 parameter.

- ERRORLOGBUFF_S2

  ERRORLOGBUFF_S2 specifies the number of S2 space error log buffers reserved for system error log entries. Each buffer is ERLBUFFERPAG_S2 in length. If ERRORLOGBUFF_S2 is too low, messages might not be written to the error log file. If it is too high, the buffers can consume unnecessary physical pages.

- SCSI_ERROR_POLL

  The purpose of SCSI_ERROR_POLL is to cause OpenVMS to send a SCSI Test Unit Ready command every hour to each SCSI disk, in an attempt to force latched errors to become unlatched and to be reported immediately. SCSI_ERROR_POLL has a default value of 1. However, it can be set to 0 by the user in order to stop the error polling activity.

- SHADOW_ENABLE

  Special parameter reserved for HP use.

- SHADOW_HBMM_RTC (Alpha and I64 only)

  SHADOW_HBMM_RTC controls the interval the system waits between the checking of reset thresholds for shadow sets that have Host-Based Minimerge (HBMM) bitmaps. If the reset threshold is exceeded, the bitmap is zeroed.

- SHADOW_PSM_DLY

  When a copy or merge operation is needed on a shadow set that is mounted on many systems, the Shadowing facility attempts to perform the operation on a system that has a local connection to all the shadow set members. Shadowing implements the copy or merge operation by adding a time delay based on the number of shadow set members that are MSCP-served to the system. No delay is added for local members; therefore, a system with all locally accessible shadow set members usually performs the copy or merge before a system on which one or more members is served—and is therefore delayed—does.

  SHADOW_PSM_DLY allows the system manager to adjust the delay that Shadowing adds. By default, the delay is 30 seconds for each MSCP-served shadow set member. The valid range for the specified delay is 0 through 65,535 seconds.

  When a shadow set is mounted on a system, the value of SHADOW_PSM_DLY is used as the default shadow set member recovery delay for that shadow set. To modify SHADOW_PSM_DLY for an existing shadow set, refer to the SET SHADOW/RECOVERY_OPTIONS=DELAY_PER_SERVED_MEMBER=n command.

- SHADOW_REC_DLY (Alpha and I64 only)

  The value of SHADOW_REC_DLY is added to the value of the RECNXINTERVAL parameter to determine the length of time a system waits before it attempts to manage recovery operations on shadow sets that are mounted on the system, using the priority of the shadow sets.

- SHADOW_SITE_ID (Alpha and I64 only)

  SHADOW_SITE_ID allows a system manager to define a site value, which Volume Shadowing uses to determine the best device to perform reads, thereby improving performance. The system manager can now define the site value to be used for all shadow sets mounted on a system.

- SYSSER_LOGGING (Alpha and I64 only)

  A value of 1 for SYSSER_LOGGING enables logging of system service requests for a process. The default is 1. This parameter is dynamic.

- TTY_DEFCHAR3 (Alpha and I64 only)

  TTY_DEFCHAR3 allows a user to set a bit so that the OpenVMS terminal driver remaps Ctrl/H to Delete. HP recommends that you *not* set this bit as a systemwide default.

Characteristic	Value (Hex)	Function
TT3$M_BS	10	When this bit is set, the OpenVMS terminal console remaps CTRL/H to Delete.

- VHPT_SIZE (I64 only)

  VHPT_SIZE is the number of kilobytes to allocate for the Virtual Hash Page Table (VHPT) on each CPU in the system:

  - 0 indicates that no VHPT is allocated.
  - 1 indicates that OpenVMS chooses a default size that is appropriate for your system configuration.

3.8.2 Changed System Parameters

Definitions of the following system parameters have been changed in OpenVMS Version 8.2:

- BALSETCNT
- CHANNELCNT
- CRD_CONTROL
- DEVICE_NAMING
- ERLBUFFERPAGES
- ERRORLOGBUFFERS
- FAST_PATH_PORTS
- GALAXY
- GLX_SHM_REG
- LAN_FLAGS
- LCKMGR_MODE

- MMG_CTLFLAGS
- MULTITHREAD
- NISCS_MAX_PKTSZ
- PQL_MASTLM
- PQL_MENQLM
- SECURITY_POLICY
- SHADOW_MBR_TMO
- VCC_FLAGS

In addition to the parameters in this list, many parameters that were formerly designated as *Alpha only* are now *Alpha and I64*.

For more detailed information, refer to the *HP OpenVMS Version 8.2 Release Notes* and *HP OpenVMS System Management Utilities Reference Manual*.

3.9 Additional Time Zones Added to Database

In OpenVMS Version 8.2, 540 time zones are provided based on the time-zone public database, named tzdata2003e, placed in `ftp://elsie.nci.nih.gov/pub/`. Existing time zones are updated, and there are 204 new time zones added to the database. For a list of the new time-zone names, see *HP OpenVMS System Manager's Manual*.

3.10 Volume Shadowing for OpenVMS New Features

Volume Shadowing for OpenVMS introduces the following new features in this release:

- Host-Based Minimerge (HBMM)

  HBMM is designed to improve shadow set merge operations. HBMM uses a bitmap to track the blocks that changed. HBMM compares and reconciles only those sections of the shadow set where changes occurred since the bitmap was reset.

  An HBMM policy keyword, RESET_THRESHOLD, is used to specify the number of blocks that can be changed before the bitmap is cleared. A new system parameter, SHADOW_HBMM_RTC, is used to specify how frequently the bitmap is checked to determine if the number of changed blocks exceeds the RESET_THRESHOLD setting. If it does, the bitmap is cleared. HBMM operations can be significantly faster than full merges, depending on the value of RESET_THRESHOLD.

- Prioritizing merge and copy operations

  You can now control the order in which merge and copy operations occur on the remaining systems in a cluster after a system failure. The new /PRIORITY=*n* qualifier to the SET SHADOW command can be used to assign different priorities to all mounted shadow sets. Assigning higher priorities to the most important volumes ensures that they are merged or copied before less important volumes, which are assigned a lower priority.

  You can dynamically increase or decrease the SHADOW_MAX_COPY setting as circumstances change in the cluster. Without having to dismount the shadow sets, you can use the new EVALUATE_RESOURCES qualifier to the SET SHADOW command to cause a system to adjust its workload to a new SHADOW_MAX_COPY value or to new priorities specified with the SET SHADOW/PRIORITY=*n* DSA*n* command.

A new system parameter, SHADOW_REC_DLY, enables you to predict the order that each system attempts to manage copy or merge operations after a system failure. You can assign the shortest delays to the systems that are best able to perform recovery operations.

- Licensing Volume Shadowing for OpenVMS

  You can purchase capacity licenses, per CPU, for OpenVMS I64 systems. OpenVMS Alpha Version 8.2 continues to provide capacity and per-disk licenses.

  For OpenVMS I64 computers, a volume shadowing license is available either as a separate product or as part of the collection of OpenVMS products known as the Enterprise Operating Environment. For more information about the Enterprise Operating Environment, refer to the *HP Operating Environments for OpenVMS Industry Standard 64 Version 8.2 for Integrity Servers Software Product Description* (SPD 82.34.xx).

  For more information about volume shadowing licensing, refer to the *HP Volume Shadowing for OpenVMS Software Product Description* (SPD 27.29.xx). For more information about the License Management Facility, refer to the OpenVMS operating system SPD or the *HP OpenVMS License Management Utility Manual*.

For more information about the HBMM new feature, refer to Chapter 6.

4
Programming Features

This chapter describes new features relating to application and system programming in this version of the HP OpenVMS operating system.

4.1 Analyze Utility Enhancements (I64 Only)

The Analyze utility has been enhanced on OpenVMS I64 systems to analyze Executable and Linkable Format (ELF) object and image files. Additional information describing these enhancements has been added to the *HP OpenVMS DCL Dictionary* for the ANALYZE/IMAGE and ANALYZE/OBJECT commands.

Using record formats and landmark values in the file, the ANALYZE utility on OpenVMS I64 can determine the architecture type and whether the file is an object or image file. Although ANALYZE accepts the /OBJECT and /IMAGE qualifiers, these qualifiers do not restrict the analysis to the specified file type.

For more information about the enhancements to the Analyze utility, see the *HP OpenVMS DCL Dictionary*.

4.2 OpenVMS Calling Standard Changes for OpenVMS I64

The OpenVMS Calling Standard has been adapted for use on systems using Intel Itanium processors and running OpenVMS I64.

The OpenVMS Calling Standard on the Intel Itanium processor family is designed to follow the Itanium software conventions as much as possible while avoiding user-visible differences from the OpenVMS VAX and Alpha conventions. Changes to the Itanium convention were made only where necessary to maintain compatibility with the historical OpenVMS design. The goal was to minimize the cost and difficulty of porting applications and OpenVMS itself to the Itanium architecture.

Refer to the *HP OpenVMS Calling Standard* for more information.

4.3 Checksum Utility

The Checksum utility calculates file, image, or object checksums for an OpenVMS file. The CHECKSUM command invokes the utility, which can now be run on I64, Alpha, or VAX platforms. The result, or checksum, is available in the DCL symbol CHECKSUM$CHECKSUM.

For information about this utility, see the CHECKSUM command in the *HP OpenVMS DCL Dictionary*.

4.3.1 CHECKSUM/OBJECT Enhanced for I64 Objects

CHECKSUM/OBJECT for I64 objects (ELF objects) provides new information into the calculation for a checksum:

- EIDC (entity identification consistency check) information
- FPMODE (whole programming floating-point mode) information

If the EIDC or FPMODE fields are present in an object file, then the I64 object checksum calculated by an earlier version of the Checksum utility is different than the one calculated by Checksum on OpenVMS Version 8.2.

The difference can be seen in the checksum of the ".note" section.

The previous version did not calculate any checksum for this section while the new version calculates the checksum if the information is present. Use the /SHOW=SECTIONS qualifier to view this behavior.

CHECKSUM/IMAGE is not affected; an image does not have EIDC or FPMODE information in the ".note" section.

The file checksums (CHECKSUM without /IMAGE or /OBJECT) are also not affected. File checksums use the whole file or the record structured data to calculate the checksum.

4.4 C Run-Time Library Enhancements

The following sections describe the C Run-Time Library (RTL) enhancements included in OpenVMS Version 8.2. These enhancements provide improved UNIX portability, standards compliance, and the flexibility of additional user-controlled feature selections. New C RTL functions are also included. For more details, refer to the *HP C Run-Time Library Reference Manual for OpenVMS Systems*.

4.4.1 File-Locking Functions

The following X/Open file-locking functions have been added. They allow a user to lock and unlock files and provide access synchronization across threaded programs:

```
flockfile
ftrylockfile
funlockfile
clearerr_unlocked
getc_unlocked
getchar_unlocked
feof_unlocked
ferror_unlocked
fgetc_unlocked
fputc_unlocked
putc_unlocked
putchar_unlocked
```

4.4.2 Standard-Compliant stat Structure

An X/Open standard-compliant definition of the `stat` structure and associated definitions are added. To use these new definitions, applications must compile with a new feature macro defined:

_USE_STD_STAT

The following new macros have also been added to support the standard-compliant `stat` structure:

S_INO_NUM(ino)
S_INO_SEQ(ino)
S_INO_RVN(ino)
S_INO_RVN_RVN(ino)
S_INO_RVN_NMX(ino)

4.4.3 File-System Statistics Support

The following X/Open functions that return file system information have been added in support of UNIX portability:

```
statvfs
fstatvfs
```

4.4.4 fcntl File Status Flags

The F_SETFL and F_GETFL command options are added to the `fcntl` function to set and get file status flags.

4.4.5 UNIX Style Pipe Support

For added UNIX portability, the C RTL pipe implementation now uses stream I/O as well as record I/O, under control of a new feature logical, DECC$STREAM_PIPE.

The legacy behavior, record I/O, is the default behavior.

To enable stream I/O pipe support, define the DECC$STREAM_PIPE feature logical name to ENABLE:

```
$ DEFINE DECC$STREAM_PIPE ENABLE
```

4.4.6 DECC$POPEN_NO_CRLF_REC_ATTR

A pipe opened with the `popen` function has its record attributes set to CR/LF carriage control (fab$b_rat |= FAB$M_CR). UNIX systems do not insert CR/LF for pipes.

A new feature logical has been added to provide UNIX compatible behavior. To override the default legacy behavior and thereby prevent CR/LF carriage control from being added to the pipe records, define DECC$POPEN_NO_CRLF_REC_ATTR to ENABLE:

```
$ DEFINE DECC$POPEN_NO_CRLF_REC_ATTR ENABLE
```

Be aware that enabling this feature might result in undesired behavior from other functions, such as `gets`, that rely on the carriage-return character.

4.4.7 glob and globfree 64-Bit Support

64-bit support is added for the `glob` and `globfree` functions. As a result, the following additional function entry points are now available for use with 32-bit and 64-bit pointer sizes, respectively:

```
_glob32         _glob64
_globfree32     _globfree64
```

4.4.8 socketpair

The `socketpair` TCP/IP socket routine has been added.

This routine is used for creating a pair of connected sockets and requires the underlying TCP/IP product to have the TCPIP$SOCKETPAIR function available.

4.5 DCE RPC Now Supports IEEE Floating-Point Type

DCE RPC for OpenVMS now supports both G_FLOAT and IEEE floating-point types on OpenVMS Alpha and OpenVMS I64 platforms. The default floating-point type on the Alpha platform remains G_FLOAT. The default floating-point type on I64 platform is IEEE_FLOAT.

DCE RPC Application developers need to use the rpc_set_local_float_drep call in their applications when using the non-default floating-point type

Note

DCE RPC on OpenVMS VAX platforms supports only G_FLOAT type.

4.6 OpenVMS Debugger

The following sections describe new features of the OpenVMS Debugger on OpenVMS I64 systems.

4.6.1 Intel® Itanium® Hardware Support

OpenVMS I64 Debugger supports the following hardware registers:

- General registers R0 through R127
- Floating registers F0 through F127
- Branch registers B0 through B7
- Predicate registers P0 through P63. You can examine all predicate register values using the symbol named PR.
- Application registers: AR16 (RSC), AR17 (BSP), AR18 (BSPSTORE), AR19 (RNAT), AR25 (CSD), AR26 (SSD), AR32 (CCV), AR36 (UNAT), AR64 (PFS), AR65 (LC), AR66 (EC)
- A program counter named PC, synthesized from the hardware IP register and the ri field of the PSR register
- Miscellaneous registers: CFM (current frame marker), UM (user mask), PSP (previous stack pointer), and IIPA (previously executed bundle address)
- Output register names OUT0 through OUT7. These names are provided for convenience to make it easy to identify the registers that are used to pass integer arguments from the current routine to a called routine. For more information, refer to the *HP OpenVMS Calling Standard*.

4.6.2 OpenVMS I64 Language Support

OpenVMS I64 Debugger supports programs written in the following languages:

BASIC
BLISS
C
C++
COBOL
IMACRO
Fortran
Intel assembler (IAS)
Pascal

4.6.3 Heap Analyzer Available on OpenVMS I64 Systems

The Heap Analyzer is now available on OpenVMS I64 systems and is activated differently than on Alpha systems. A new start command, START HEAP_ANALYZER, is available from the debugger command-line prompt (DBG>) to start the Heap Analyzer from the OpenVMS I64 Debugger. For complete information, refer to the Heap Analyzer chapter in the *HP OpenVMS Debugger Manual*.

4.7 Extended Lock Value Block

A lock value block is a feature of the OpenVMS Lock Manager that provides a type of synchronized interprocess/intracluster communication. In the past, OpenVMS applications using the Lock Manager have been able to store and retrieve up to 16 bytes of lock value block data during locking operations. In OpenVMS Version 8.2, applications may now optionally store up to 64 bytes of data in the lock value block.

The $GETLKI system service has been modified to add two new item codes that this new feature requires. The purpose of the items codes is:

- To retrieve the extended value block (LKI$_XVALBLK) into a 64-byte buffer.
- To indicate whether the last writer wrote a short value block (LKI$_XVALNOTVALID) into a boolean value in a quadword buffer.

For more information, refer to *HP OpenVMS Programming Concepts Manual* and to the *HP OpenVMS System Services Reference Manual*.

4.8 Librarian Utility and Library Routines (I64 only)

The following sections provide information about the Librarian utility and Library routines on OpenVMS I64 systems. On OpenVMS Alpha systems, the Librarian utility and Library routines are unchanged.

Note

The Librarian utility is also referred to as the Librarian. Librarian routines are sometimes called library services, LBR routines, or LBR$ routines.

4.8.1 Librarian Usage Summary

You can use the DCL command LIBRARY to invoke the Librarian utility (or use Library [LBR] routines) to create the following library types:

- I64 (ELF) object library
- I64 (ELF) shareable image library
- Macro library
- Help library
- Text library

The LIBRARY command invokes the OpenVMS Librarian utility from which you can maintain library modules or simply display information about a library and its modules. The I64 Librarian utility provides the same features provided by the Alpha Librarian, except for the changes and restrictions specified in *HP OpenVMS Version 8.2 Release Notes*.

Table 4–1 shows the libraries that are created by the Librarian utility for each OpenVMS platform.

Table 4–1 Libraries Created by OpenVMS Platforms

OpenVMS VAX	OpenVMS Alpha	OpenVMS I64
VAX object	Alpha object	I64 object
VAX sharable image	Alpha sharable image	I64 sharable image
Alpha object	VAX object	
Alpha sharable image	VAX sharable image	
Macro	Macro	Macro
Text	Text	Text
Help	Help	Help

4.8.2 Changes to the Librarian Utility

This section describes changes and restrictions to the Librarian utility on OpenVMS I64 systems.

4.8.2.1 Librarian Defaults to Intel® Itanium® Architecture

There is no architecture switch for the OpenVMS I64 Librarian utility. When used with the following qualifiers, the Librarian works on OpenVMS ELF object and image libraries:

/OBJECT—Work on OpenVMS ELF object libraries (default)
/SHARE—Work on OpenVMS ELF shareable image libraries

The default library type created is an object library if no OBJECT and SHARE qualifiers are specified.

4.8.2.2 No Support for /ALPHA and /VAX Qualifiers

The /ALPHA and /VAX qualifiers are not supported in the I64 Librarian utility. The Librarian utility works on the following library types only:

MACRO
TEXT
HELP
ELF OBJECT
ELF SHARABLE IMAGE

4.8.2.3 Enhanced /REMOVE Qualifier

The /REMOVE qualifier has been enhanced for the I64 Librarian. The format now allows you to specify the module of the instance of the symbol to be removed.

Requests the LIBRARY command to delete one or more entries from the global symbol table in an object library.

/REMOVE=(symbol[:module] [, ...])

symbol

The symbol to be deleted from the global symbol table.

module

The module whose instance of the symbol is to be removed from the global symbol table. With the support of UNIX-style weak symbols and ELF group symbols, a symbol can have definitions from more than one module. This extended syntax allows you to remove from the index a symbol of a specific module without removing the symbol instances of all other modules from the index.

Description

If you specify more than one symbol, separate the symbols with commas and enclose the list in parentheses. Wildcard characters are allowed in the symbol specification. To display the names of the deleted global symbols, you must also specify the /LOG qualifier.

4.8.3 Changes to the Library (LBR) Routines

The following sections describe changes and restrictions to the Library routines for OpenVMS I64 systems. All routines not listed here remain the same as on Alpha systems (and as documented in the *OpenVMS Utility Routines Manual*).

4.8.3.1 New Library Types Added

Two new library types for the LBR$OPEN routine have been added:

LBR$C_TYP_ELFOBJ (9)—Represents an ELF object library
LBR$C_TYP_ELFSHSTB (10)—Represents an ELF shareable image library

In addition, the following library types for the LBR$OPEN Library routine are not supported on OpenVMS I64:

LBR$C_TYP_OBJ (1)—Represents a VAX object library
LBR$C_TYP_SHSTB (5)—Represents a VAX shareable image library
LBR$C_TYP_EOBJ (7)—Represents an Alpha object library

LBR$C_TYP_ESHSTB (8)—Represents an Alpha shareable image library

Note

You cannot use these library types to create or open OpenVMS Alpha or VAX object and shareable image libraries.

4.8.3.2 Accessing ELF Object Libraries

ELF object modules are inherently random access modules, whereas OpenVMS Alpha objects, text modules, and so on, are sequential. To allow random access, a new library routine was created to map the ELF object modules into process P2 space so that applications can make random access queries. To recover virtual address space from this mapping another library routine was created to remove this mapping. These new routines (LBR$MAP_MODULE and LBR$UNMAP_MODULE) work only with ELF object libraries. These entry points are 64-bit interfaces since they refer to P2 space.

Because of the random-access nature of ELF object files, the following operations are not be allowed on ELF object libraries:

LBR$GET_RECORD
LBR$SET_LOCATE
LBR$SET_MOVE

Because inserting modules into the library is a sequential operation, LBR$PUT_RECORD is allowed on ELF object libraries. Since the ELF object modules are not segmented into records, the module's on-disk size needs to be provided upon the first call to LBR$PUT_RECORD when writing a module into the library.

The C code fragment in the following example illustrates how to use LBR$PUT_RECORD to insert an object module:

```
bufdesc->dsc$a_pointer = &p0_buffer ;
   bytes_to_transfer = module_size ;

   while ( bytes_to_transfer )  {
      transfer = MIN ( bytes_to_transfer ,
                       ELBR$C_MAXRECSIZ ) ;

      bufdesc->dsc$w_length = transfer ;

      status = lbr$put_record (  library_index ,
                               & bufdesc ,
                               & txtrfa ,
                                 module_size ) ;
      if ( (status & 1) == 0 )
         break ;

      bytes_to_transfer      -= transfer ;
      bufdesc->dsc$a_pointer += transfer ;
      } ;

   if ( (status & 1) == 1 )
      status = lbr$put_end ( library_index ) ;
```

To avoid making several calls to LBR$PUT_RECORD, a new library routine, LBR$PUT_MODULE, has been created (see Section 4.8.4).

4.8.4 New Librarian (LBR) Routines for ELF Object Libraries

This section describes the following four new Library routines for ELF object libraries:

- LBR$LOOKUP_TYPE
- LBR$MAP_MODULE
- LBR$PUT_MODULE
- LBR$UNMAP_MODULE

LBR$LOOKUP_TYPE—Search the index for the key from a module (RFA)

The LBR$LOOK_TYPE routine searches the index for the key from a particular module (RFA) and returns that key's type for that module.

Format

LBR$LOOKUP_TYPE library_index, key_name, txtrfa, ret_types

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL routine. The **library_index** argument is the address of the longword that contains the index.

key_name

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

The **key_name** argument is the address of the string descriptor pointing to the key with the following argument characteristics:

Argument Characteristics	Entry
OpenVMS usage	char_string
type	character string
access	read only
mechanism	by descriptor

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

The module's record file address (RFA) of the library module header. The **txtrfa** argument is the address of the 2-longword array that specifies the RFA of the module header.

ret_types

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	write only
mechanism:	by reference

The address of a longword to receive the symbol types found for the specified module (**txtrfa**). The return type bits are:

LBR$M_SYM_NGG = 1
LBR$M_SYM_UXWK = 2
LBR$M_SYM_GG = 4
LBR$M_SYM_GUXWK = 8

Description

This routine searches the index for the key from a particular module (RFA) and returns that key's type for that module, if present. Otherwise, it returns LBR$_KEYNOTFND.

LBR$MAP_MODULE—Map a module

The LBR$MAP_MODULE routine maps a module into process P2 space.

Format

LBR$MAP_MODULE library_index, ret_va_addr, ret_mod_len, txtrfa

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL library routine. The **library_index** argument is the address of the longword that contains the index.

ret_va_addr

OpenVMS usage:	address
type:	quadword address
access:	write only
mechanism:	by 32-bit or 64-bit reference

The 32-bit or 64-bit virtual address of a naturally aligned quadword into which the routine returns the virtual address at which the routine mapped the library module.

ret_mod_len

OpenVMS usage:	byte_count
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

The address of a naturally aligned quadword into which the library routine returns the module length.

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

The module's record file address (RFA) of the library module header. The **txtrfa** argument is the address of the 2-longword array that specifies the RFA of the module header.

Description

This routine maps a module, with the given **txtrfa**, into process P2 memory space and returns the virtual address where the module is mapped and the module size.

Unlike other LBR services that use RMS services, LBR$MAP_MODULE also uses system services. Because of this, the secondary status for error returns is placed in LBR$$GL_SUBSTS. Use this to find further status when an error is returned.

LBR$PUT_MODULE—Put module from memory into current library

The LBR$PUT_MODULE routine puts an entire module, with the module's record file address (RFA), from memory space into the current library.

Format

LBR$PUT_MODULE library_index, mod_addr, mod_len, txtrfa

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index return by the LBR$INI_CONTROL library service. The **library_index** argument is the address of the longword that contains the index.

mod_addr

OpenVMS usage:	address
type:	quadword address
access:	read only
mechanism:	by 32- or 64-bit reference

The address from which the Library service will obtain the 64-bit address of where the module is mapped in memory. The **mod_addr** argument is the 32- or 64-bit virtual address of a naturally aligned quadword containing the virtual address location of the module to write to the library.

mod_len

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	read only
mechanism:	by 32- or 64-bit reference

The 64-bit virtual address of a naturally aligned quadword containing the length of the module that the Library service is to write into the library.

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	write only
mechanism:	by reference

The module's record file address (RFA) of the library module header. The **txtrfa** argument is the address of a 2-longword array receiving the RFA of the newly created module header.

Description

The LBR$PUT_MODULE routine puts an entire module, with the module's record file address (RFA), from memory space into the current library. LBR$PUT_END is not required when you write an entire module to the current library.

LBR$UNMAP_MODULE—Unmap a module from process P2 space

The LBR$UNMAP_MODULE routine unmaps a module from process P2 space.

Format

LBR$UNMAP_MODULE library_index, txtrfa

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL routine. The **library_ index** argument is the address of the longword that contains the index.

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

The module's record file address (RFA) of the library module header. The **txtrfa** argument is the address of the 2-longword array that specifies the RFA of the module header.

Description

Unmaps the module, with the record file address in **txtrfa**, from process P2 space. This releases the resources used to map the module.

Unlike other LBR services that use RMS services, LBR$UNMAP_MODULE also uses system services. Because of this, the secondary status for error returns is placed in LBR$GL_SUBSTS. Use this to find further status when an error is returned.

4.8.5 Extended Library (LBR) Routines for ELF Object Libraries

The Library routines described in the following section have changed for OpenVMS I64 systems. Because of extra information that needs to be passed due to the addition of ELF group symbols and UNIX-style weak definitions, the Library routines have been extended to take additional parameters. Extensions to the Library routines only affect ELF object and ELF shareable image libraries.

The following Library routines are listed in this section:

- LBR$DELETE_DATA
- LBR$DELETE_KEY
- LBR$GET_INDEX
- LBR$INSERT_KEY
- LBR$LOOKUP_KEY
- LBR$PUT_RECORD
- LBR$REPLACE_KEY
- LBR$SEARCH

Library routines not listed here remain the same for both OpenVMS I64 systems and OpenVMS Alpha systems and are documented in the *OpenVMS Utility Routines Manual*.

LBR$DELETE_DATA—Delete module data

The LBR$DELETE_DATA routine deletes module data from the library.

Format

LBR$DELETE_DATA   library_index, txtrfa, [flags]

Returns

OpenVMS usage:	cond_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Longword condition value. Most utility routines return a condition value. Condition values that this routine can return are listed under Condition Values Returned.

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL library routine. The **library_index** argument is the address of the longword that contains the index.

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Record's file address (RFA) of the module header for the module you want to delete. The **txtrfa** argument is the address of the 2-longword array that contains the RFA. You can obtain the RFA of a module header by calling LBR$LOOKUP_KEY or LBR$PUT_RECORD.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

The contents of the flag are ignored. Its purpose is to indicate to this routine that the application knows about the new index structure for ELF object and ELF shareable image libraries.

Description

If you want to delete a library module, you must first call LBR$DELETE_KEY to delete all keys that point to it. If no library index keys are pointing to the module header, LBR$DELETE_DATA deletes the module header and associated data records; otherwise, this routine returns the error LBR$_STILLKEYS. Note that other library routines may reuse data blocks that contain no data.

Condition Values Returned

LBR$_ILLCTL	Specified library control index not valid.
LBR$_INVRFA	Specified RFA not valid.
LBR$_LIBNOTOPN	Specified library not open.
LBR$_STILLKEYS	Keys in indexes still point at the module header. Therefore, the specified module was not deleted.

LBR$DELETE_KEY—Delete a key from the current index

The LBR$DELETE_KEY routine removes a key from the current index.

Format

LBR$DELETE_KEY library_index, key_name [, txtrfa] [, flags]

Returns

OpenVMS usage:	cond_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Longword condition value. Most utility routines return a condition value. Condition values that this routine can return are listed under Condition Values Returned.

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL library routine. The **library_index** argument is the address of the longword that contains the index.

key_name

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Key to be deleted from the library index. For libraries with binary keys, the **key_name** argument is the address of an unsigned longword containing the key number.

For libraries with ASCII keys, the **key_name** argument is the address of the string descriptor pointing to the key with the following argument characteristics:

Argument Characteristics	Entry
OpenVMS usage	char_string
type	character string
access	read only
mechanism	by descriptor

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

If present *and* if the **flags** argument is not present, the routine will scan for all types of the key for the specified **txtrfa** and delete those entries.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

If present, this argument indicates that a particular type of the key or all types of the key is to be deleted. The flags bits are:

Flag Bits	Description
LBR$M_SYM_WEAK = 0x1	UNIX-style weak symbol attribute
LBR$M_SYM_GROUP = 0x2	Group symbol attribute
LBR$M_SYM_ALL = 0x80000000	All symbols

If the **txtrfa** is not present or 0 (zero), the type indicated by **flags** is deleted. If the **txtrfa** specifies a non-zero value, the entry of the type indicated, with the **txtrfa** supplied, is removed. Note that only one type or all types may be specified.

Description

If LBR$DELETE_KEY finds the key specified by **key_name** in the current index, it deletes the key. Note that if you want to delete a library module, you should first use LBR$DELETE_KEY to delete all keys that point to it, then use LBR$DELETE_DATA to delete the module's header and associated data. You cannot call LBR$DELETE_KEY from within the user-supplied routine specified in LBR$SEARCH or LBR$GET_INDEX.

Condition Values Returned

LBR$_ILLCTL	Specified library control index not valid.
LBR$_KEYNOTFND	Specified key not found.
LBR$_LIBNOTOPN	Specified library not open.
LBR$_UPDIRTRAV	Specified index update not valid in a user-supplied routine specified in LBR$SEARCH or LBR$GET_INDEX.

LBR$GET_INDEX—Call a routine for selected index keys

The LBR$GET_INDEX routine calls a user-supplied routine for selected keys in an index.

Format

LBR$GET_INDEX library_index,index_number, routine_name [, match_desc] [, flags]

Returns

OpenVMS usage:	cond_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Longword condition value. Most utility routines return a condition value. Condition values that this routine can return are listed under Condition Values Returned.

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL routine. The **library_index** argument is the address of the longword that contains the index.

index_number

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Number of the library index. The **index_number** argument is the address of a longword containing the index number. This is the index number associated with the keys you want to use as input to the user-supplied routine.

routine_name

OpenVMS usage:	procedure
type:	procedure value
access:	read only
mechanism:	by reference

User-supplied routine called for each of the specified index keys. The **routine_name** argument is the address of the procedure value for this user-supplied routine.

LBR$GET_INDEX passes three arguments to the routine:

- A key name.
  - For libraries with ASCII keys, the **key_name** argument is the address of a string descriptor pointing to the key. Note that the string and the

string descriptor passed to the user routine are valid only for the duration of that call. The string must be copied privately if you need it again for more processing.

  – For libraries with binary keys, the **key_name** argument is the address of an unsigned longword containing the key number.

- The record's file address (RFA) of the module's header for this key name. The RFA argument is the address of a 2-longword array that contains the RFA.
- The key's type whose bits are:

Flag Bits	Description
LBR$M_SYM_WEAK = 1	UNIX-style weak symbol attributes
LBR$M_SYM_GROUP = 2	Group symbol attribute

The user routine must return a value to indicate success or failure. If the user routine returns a false value (low bit = 0), LBR$GET_INDEX stops searching the index and returns the status value of the user-specified routine to the calling program.

The user routine cannot contain calls to either LBR$DELETE_KEY or LBR$INSERT_KEY.

match_desc

OpenVMS usage:	char_string
type:	character string
access:	read only
mechanism:	by descriptor

Key matching identifier. The **match_desc** argument is the address of a string descriptor pointing to a string used to identify which keys result in calls to the user-supplied routine. Wildcard characters are allowed in this string. If you omit this argument, the user routine is called for every key in the index. The **match_desc** argument is valid only for libraries that have ASCII keys.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

If present and non-zero, specifies the type, or all types, of the key provided. The flag bits are:

Flag Bits	Description
LBR$M_SYM_WEAK = 0x1	UNIX-style weak symbol attribute
LBR$M_SYM_GROUP = 0x2	Group symbol attribute
LBR$M_SYM_ALL = 0x80000000	All symbols

The user routine will be provided the key's type through an additional third parameter.

Description

LBR$GET_INDEX searches through the specified index for keys that match the argument **match_desc**. Each time it finds a match, it calls the user routine specified by the **routine_name** argument. If you do not specify the **match_desc** argument, it calls the user routine for every key in the index.

For example, if you call LBR$GET_INDEX on an object library with **match_desc** equal to TR* and **index_number** set to 1 (module name table), then LBR$GET_INDEX calls **routine_name** for each module whose name begins with TR.

Condition Values Returned

LBR$_ILLCTL	Specified library control index not valid.
LBR$_ILLIDXNUM	Specified index number not valid.
LBR$_LIBNOTOPN	Specified library not open.
LBR$_NULIDX	Specified library empty.

LBR$INSERT_KEY—Insert a new key

The LBR$INSERT_KEY routine inserts a new key in the current library index.

Format

LBR$INSERT_KEY library_index ,key_name ,txtrfa, [flags]

Returns

OpenVMS usage:	cond_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Longword condition value. Most utility routines return a condition value. Condition values that this routine can return are listed under Condition Values Returned.

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL library routine. The **library_index** argument is the address of the longword that contains the index.

key_name

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Name of the new key you are inserting. If the library uses binary keys, the **key_name** argument is the address of an unsigned longword containing the value of the key.

If the library uses ASCII keys, the **key_name** argument is the address of a string descriptor of the key with the following argument characteristics:

Argument Characteristics	Entry
OpenVMS usage	char_string
type	character string
access	read only
mechanism	by descriptor

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	modify
mechanism:	by reference

The record's file address (RFA) of the module associated with the new key you are inserting. The **txtrfa** argument is the address of a 2-longword array containing the RFA. You can use the RFA returned by the first call to LBR$PUT_RECORD.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

If present, specifies the key's type. The flag bits are:

Flag Bits	Description
LBR$M_SYM_WEAK = 0x1	UNIX-style weak symbol attribute
LBR$M_SYM_GROUP = 0x2	Group symbol attribute

If this parameter is not present, the normal NonGroup-Global type is the assumed type.

Description

You cannot call LBR$INSERT_KEY within the user-supplied routine specified in LBR$SEARCH or LBR$GET_INDEX.

Condition Values Returned

LBR$_DUPKEY	Index already contains the specified key.
LBR$_ILLCTL	Specified library control index not valid.
LBR$_INVRFA	Specified RFA does not point to valid data.
LBR$_LIBNOTOPN	Specified library not open.
LBR$_UPDURTRAV	LBR$INSERT_KEY was called by the user-defined routine specified in LBR$SEARCH or LBR$GET_INDEX.

LBR$LOOKUP_KEY—Look up a library key

The LBR$LOOKUP_KEY routine looks up a key in the library's current index and prepares to access the data in the module associated with the key.

Format

LBR$LOOKUP_KEY   library_index ,key_name ,txtrfa, [flags]

Returns

OpenVMS usage:	cond_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Longword condition value. Most utility routines return a condition value. Condition values that this routine can return are listed under Condition Values Returned.

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL routine. The **library_index** argument is the address of the longword that contains the index.

key_name

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Name of the library key. If the library uses binary keys, the **key_name** argument is the address of the unsigned longword value of the key.

If the library uses ASCII keys, the **key_name** argument is the address of a string descriptor for the key with the following argument characteristics:

Argument Characteristics	Entry
OpenVMS usage	char_string
type	character string
access	read only
mechanism	by descriptor

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	write only
mechanism:	by reference

The record's file address (RFA) of the library module header. The **txtrfa** argument is the address of the 2-longword array that receives the RFA of the module header.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	write only
mechanism:	by reference

If present and not zero, receives the type of key returned. The flag bits are:

Flag Bits	Description
LBR$M_SYM_WEAK = 0x1	UNIX-style weak symbol attribute
LBR$M_SYM_GROUP = 0x2	Group symbol attribute

The key returned is the highest precedent definition type present.

Description

If LBR$LOOKUP_KEY finds the specified key, it initializes internal tables so you can access the associated data.

This routine returns the RFA to the 2-longword array referenced by **txtrfa**.

Condition Values Returned

LBR$_ILLCTL	Specified library control index not valid.
LBR$_INVRFA	RFA obtained not valid.
LBR$_KEYNOTFND	Specified key not found.
LBR$_LIBNOTOPN	Specified library not open.

LBR$PUT_RECORD—Write a data record

The LBR$PUT_RECORD routine writes a data record beginning at the next free location in the library.

Format

LBR$PUT_RECORD library_index ,bufdes ,txtrfa [,mod_size]

Returns

OpenVMS usage:	cond_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Longword condition value. Most utility routines return a condition value. Condition values that this routine can return are listed under Condition Values Returned.

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL routine. The **library_index** argument is the address of the longword that contains the index.

bufdes

OpenVMS usage:	char_string
type:	character string
access:	read only
mechanism:	by descriptor

Record to be written to the library. The **bufdes** argument is the address of a string descriptor pointing to the buffer containing the record. The maximum record size for VAX libraries is symbolically defined as LBR$C_MAXRECSIZ; for Alpha and I64 libraries, the symbolic maximum record size is ELBR$_MAXRECSIZ.

txtrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	write only
mechanism:	by reference

Record's file address (RFA) of the module header. The **txtrfa** argument is the address of a 2-longword array receiving the RFA of the newly created module header upon the first call to LBR$PUT_RECORD.

mod_size

OpenVMS usage:	byte count
type:	longword (unsigned)
access:	read only
mechanism:	by value

The value from **mod_size** is read on the first call to this routine only and ignored otherwise. This value specifies the size of the module to be entered so that contiguous space is allocated within the library for that module. This argument is ignored for non-ELF object libraries and for data-reduced ELF object libraries. The LBR$PUT_END routine is still required to terminate the byte stream and close off the module.

Description

If this is the first call to LBR$PUT_RECORD, this routine first writes a module header and returns its RFA to the 2-longword array pointed to by **txtrfa**. LBR$PUT_RECORD then writes the supplied data record to the library. On subsequent calls to LBR$PUT_RECORD, this routine writes the data record beginning at the next free location in the library (after the previous record). The last record written for the module should be followed by a call to LBR$PUT_END.

Condition Values Returned

LBR$_ILLCTL	Specified library control index not valid.
LBR$_LIBNOTOPN	Specified library not open.

LBR$REPLACE_KEY—Replace a library key

The LBR$REPLACE_KEY routine modifies or inserts a key into the library.

Format

LBR$REPLACE_KEY library_index ,key_name ,oldrfa ,newrfa [,flags]

Returns

OpenVMS usage:	cond_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Longword condition value. Most utility routines return a condition value. Condition values that this routine can return are listed under Condition Values Returned.

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL routine. The **library_ index** argument is the address of the longword that contains the index.

key_name

OpenVMS usage:	char_string
type:	character string
access:	read only
mechanism:	by reference

For libraries with ASCII keys, the **key_name** argument is the address of a string descriptor for the key.

For libraries with binary keys, the **key_name** argument is the address of an unsigned longword value for the key.

oldrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Old record file address (RFA). The **oldrfa** argument is the address of a 2-longword array containing the original RFA (returned by LBR$LOOKUP_KEY) of the module header associated with the key you are replacing.

newrfa

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

New RFA. The **newrfa** argument is the address of a 2-longword array containing the RFA (returned by LBR$PUT_RECORD) of the module header associated with the new key.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by reference

If present, specifies the type of the key being replaced. The flag bits are:

Flag Bits	Description
LBR$M_SYM_WEAK = 0x1	UNIX-style weak symbol attribute
LBR$M_SYM_GROUP = 0x2	Group symbol attribute

If this parameter is not present, NonGroup-Global is the assumed type. In this case, all type lists are searched and the entries removed. The new symbol is placed as the new NonGroup-Global definition with **newrfa** as the defining module.

If this parameter is present, it represents the flags for the type of symbol being replaced. The replacement is done in place without losing its position in the type list. If the symbol does not exist when the call to this routine is made, the new definition is placed at the end of the type list for the specified type.

Because there are now different symbol definition types, HP advises using the LBR$DELETE_KEY routine followed by the LBR$INSERT_KEY routine, when the old key and new key differ in definition type.

Description

If LBR$REPLACE_KEY does not find the key in the current index, it calls the LBR$INSERT_KEY routine to insert the key. If LBR$REPLACE_KEY does find the key, it modifies the key entry in the index so that it points to the new module header.

Condition Values Returned

LBR$_ILLCTL	Specified library control index not valid.
LBR$_INVRFA	Specified RFA not valid.
LBR$_LIBNOTOPN	Specified library not open.

LBR$SEARCH—Search an Index

The LBR$SEARCH routine finds index keys that point to specified data.

Format

LBR$SEARCH library_index ,index_number ,rfa_to_find ,routine_name [, flags]

Returns

OpenVMS usage:	cond_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Longword condition value. Most utility routines return a condition value. Condition values that this routine can return are listed under Condition Values Returned.

Arguments

library_index

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library control index returned by the LBR$INI_CONTROL routine. The **library_index** argument is the address of the longword that contains the index.

index_number

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Library index number. The **index_number** argument is the address of a longword containing the number of the index you want to search.

rfa_to_find

OpenVMS usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	write only
mechanism:	by reference

Record's file address (RFA) of the module whose keys you are searching for. The **rfa_to_find** argument is the address of a 2-longword array containing the RFA (returned earlier by LBR$LOOKUP_KEY or LBR$PUT_RECORD) of the module header.

routine_name

OpenVMS usage:	procedure
type:	procedure value
access:	read only
mechanism:	by reference

Name of a user-supplied routine to process the keys. The **routine_name** argument is the address of the procedure value of a user-supplied routine to call for each key entry containing the RFA (in other words, for each key that points to the same module header).

This user-supplied routine cannot contain any calls to LBR$DELETE_KEY or LBR$INSERT_KEY.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by reference

If present and nonzero, specifies the type, or all types, of the key provided. The flag bits are:

Flag Bits	Description
LBR$M_SYM_WEAK = 0x1	UNIX-style weak symbol attribute
LBR$M_SYM_GROUP = 0x2	Group symbol attribute
LBR$M_SYM_ALL = 0x80000000	All symbols

The user routine is provided the symbol's type through an additional third parameter.

Description

Searches the library index for symbols with the given RFA and calls the supplied routine with those symbols.

Use LBR$SEARCH to find index keys that point to the same module header. Generally, in index number 1 (the module name table), just one key points to any particular module; thus, you would probably use this routine only to search library indexes where more than one key points to a module. For example, you might call LBR$SEARCH to find all the symbols in the symbol index that are associated with an object module in an object library.

If LBR$SEARCH finds an index key associated with the specified RFA, it calls a user-supplied routine with three arguments:

- The key argument, which is the address of either of the following:
  - A string descriptor for the key name (libraries with ASCII key names)
  - An unsigned longword for the key value (libraries with binary keys)
- The RFA argument, which is the address of a 2-longword array containing the RFA of the module header
- The key's type whose flag bits are:

Flag Bits	Description
LBR$M_SYM_WEAK = 1	UNIX-style weak symbol attribute
LBR$M_SYM_GROUP = 2	Group symbol attribute

The user routine must return a value to indicate success or failure. If the specified user routine returns a false value (low bit = 0), then the index search terminates.

Note that the key found by LBR$SEARCH is valid only during the call to the user-supplied routine. If you want to use the key later, you must copy it.

Condition Values Returned

LBR$_ILLCTL	Specified library control index not valid.
LBR$_ILLIDXNUM	Specified library index number not valid.
LBR$_KEYNOTFND	Library routine did not find any keys with the specified RFA.
LBR$_LIBNOTOPN	Specified library not open.

4.8.6 Library Format Changed due to New UNIX-Style Weak Symbols

Because of the requirements of the Intel C++ compiler, the library format has been expanded to accommodate new UNIX-style weak symbols. Multiple modules matching key names of new UNIX-style weak symbols can now exist in the same library. The Librarian utility ignores the OpenVMS-style weak symbol definitions as it did in the past.

UNIX-style weak symbol definitions behave in the same manner as weak transfer addresses on OpenVMS; that is, their definitions are tentative. If a definition of a stronger binding type is not seen during a link operation, a tentative definition is designated as the definitive definition.

4.8.6.1 New ELF Type for Weak Symbols

A new Executable and Linkable Format (ELF) type was generated to distinguish between the two types of weak symbol definitions.

For modules with ABI versions greater than or equal to 2:

- Type STB_WEAK represents the UNIX-style weak symbol (formerly, the OpenVMS-style weak symbol definition for ABI Version 1 ELF format)
- Type STB_VMS_WEAK represents the OpenVMS-style weak symbol definition.

The Librarian supports both the ELF ABI versions 1 and 2 of the object and image file formats within the same library.

Note

The new library format (version 6.0) applies only to ELF object and shareable image libraries. Other libraries will remain at the version 3.0 format.

4.8.6.2 Version 6.0 Library Index Format

HP recommends using the new Version 6.0 libraries.

However, note that with the new library index format, the LBR routines open Version 3.0 and Version 4.0 ELF object and shareable image libraries in read-only mode. You cannot modify the library with the LBR routines or with the Librarian utility. Instead, you can convert your older version libraries to version 6.0 libraries using the following LIBRARY command:

```
$  LIBRARY/COMPRESS library-name
```

Using the Librarian utility to modify a Version 3.0, 4.0 or 5.0 ELF object or shareable image library causes the Librarian to automatically convert your library to a version 6.0 library.

Note

Once your library is at least version level 6.0 format, you cannot access the library with some older versions of the Library routines or Librarian utility. Doing so returns the LBR$_UNSUPLVL status, and causes the Librarian utility to display the following message:

```
%LIBRAR-F-OPENIN, error opening library-name as input
 -LBR-E-UNSUPLVL, unsupported library format level
```

4.8.6.3 New Group Section Symbols

Symbols may be relative to sections contained in an ELF entity called a GROUP. These groups, and the symbols associated with them, behave in a similar fashion as the new UNIX-style weak symbol definitions; that is, they are tentative definitions. The library must now allow multiple symbol definitions in the library's symbol name index for these types of symbols.

4.8.6.4 Precedence Ordering Rules

The following list describes the symbol types for the combinations of GROUP and UNIX-style WEAK attributes in order of their precedence from highest to lowest. Within these symbol types, the ordering of symbol associations is by time of insertion, from earliest to latest. This ordering is important because when the OpenVMS I64 Linker asks for a symbol resolution from the library, it is given the earliest inserted module instance of the symbol with the highest precedence.

- *NonGroup-Global symbols* –Symbols that are not members of an ELF group and are not UNIX-style weak definitions. These symbols have the highest precedence. There may be only one definition of this type. If the symbol of this type already exists in the library, a subsequent symbol definition of this type returns an error.
- *Group Global symbols*–Symbols that belong to an ELF section associated with a group and are not UNIX-style weak definitions. Multiple definitions of this type may exist in the library. These definitions are stored in a list, ordered by insertion time. Note that if a module definition in the list is replaced, it does not lose its current position in the list.
- *UNIX-style weak symbols*–Symbols that have a UNIX-style weak binding and do not belong to a group. Multiple definitions of this type may exist in the library. These definitions are stored in a list, ordered by insertion time. Note that if a module definition in the list is replaced, it does not lose its current position in the list.
- *Group weak symbols*—Symbols that behave as combined Group Global and UNIX-style Weak attributes. Multiple definitions of this type may exist in the library. These definitions are stored in a list, ordered by insertion time. Note that if a module definition in the list is replaced, it does not lose its current position in the list.

4.9 Linker Utility

Refer to Chapter 7 for information on the Linker utility.

4.10 HP OpenVMS Migration Software

HP OpenVMS Migration Software for Alpha to Integrity servers (OMSAI), also known as the binary translator, facilitates migrating OpenVMS Alpha applications to OpenVMS I64 systems by allowing you to translate the OpenVMS Alpha images into equivalent OpenVMS I64 images. When the translated image runs, the OpenVMS I64 transparently supports the image with an environment that allows it to run as if it were on an OpenVMS Alpha system. OMSAI consists of the Alpha Environment Software Translator (AEST) utility and a collection of programs and command files designed to ease the translation process.

For information on the availability of this feature, check the following web site:

`http://h71000.www7.hp.com/openvms/products/omsva/omsva.html`

4.11 POSIX Threads Features

The following sections describe the new features added to the POSIX Threads Library.

4.11.1 Lowercase Symbol Names for /NAMES=AS_IS Compilation

The POSIX threads library PTHREAD$RTL.EXE exports a number of symbol names (for the pthread API routines, predefined exception objects, and so forth). In prior releases, all letters in these symbols have been in uppercase. In OpenVMS Version 8.2, new lowercase variants of all the symbols are provided in addition to the existing uppercase symbols. These symbols facilitate use of the /NAMES=AS_IS qualifier that is present in a number of HP compiler products.

4.11.2 Support for Process-Shared Mutexes and Condition Variables

The POSIX Threads Library now supports process-shared mutexes and condition variables. Until now, these were supported only on UNIX systems. (Note, process-shared read-write locks are still supported only on UNIX systems.) The following pthread routines are now supported on OpenVMS Version 8.2:

- `pthread_condattr_getpshared`
- `pthread_condattr_setpshared`
- `pthread_mutexattr_getpshared`
- `pthread_mutexattr_setpshared`

4.11.3 SET and SHOW Commands Enhanced (I64 Only)

The DCL commands SET and SHOW have been enhanced to allow a user to query and change two main-image header flags, UPCALLS and MKTHREADS on OpenVMS I64 systems. In addition, you can use the SET IMAGE and SHOW IMAGE commands on both Alpha and I64 systems to set and show only I64 images.

The DCL command THREADCP continues to be available on OpenVMS Alpha systems.

4.11.4 New Routine Added to Thread Independent Services API

A new routine, `tis_mutex_init_type`, has been added to the Thread Independent Services API (TIS). This routine is similar to the existing routine named `tis_mutex_init`.

tis_mutex_init_type

Initializes the specified mutex object, establishing the mutex's type and name as specified by the caller.

Format

tis_mutex_init_type (mutex,type,name):

Argument	Data Type	Access
mutex	opaque pthread_mutex_t	write
type	integer	read
name	char	read

C Binding

```
#include <tis.h>

int
tis_mutex_init_type (
    pthread_mutex_t    *mutex,
    int   type,
    const char    *name );
```

Arguments

mutex
Pointer to a mutex object (passed by reference) to be initialized.

type
Value for the mutex type attribute. The **type** argument specifies the type of mutex that will be created. Valid values are:

- PTHREAD_MUTEX_NORMAL
- PTHREAD_MUTEX_DEFAULT
- PTHREAD_MUTEX_RECURSIVE
- PTHREAD_MUTEX_ERRORCHECK

name
Textual name to be associated with the mutex.

Description

This routine initializes a mutex object with the DECthreads default mutex attributes, except for mutex *type* and *name*, which are specified by the caller. A mutex is a synchronization object that allows multiple threads to serialize their access to shared data.

The mutex object is initialized and set to the unlocked state. Refer to the *Guide to POSIX Threads Library* for information on the types of mutexes.

The mutex **name** argument is a C language string and provides an identifier that is a meaningful to a person debugging a DECthreads multithreaded application. The name string should be a string literal, or other storage that will remain for the life of the mutex. The contents of the string are not copied as part of mutex initialization.

Return Values

If an error condition occurs, this routine returns an integer value indicating the type of error, the mutex is not initialized, and the contents of *mutex* are undefined. Possible return values are as follows:

0	Successful completion.
[EAGAIN]	The system lacks the necessary resources to initialize a mutex.
[EBUSY]	The implementation has detected an attempt to reinitialize *mutex* (a previously initialized, but not yet destroyed, mutex).
[EINVAL]	The value specified by *mutex* is not a valid mutex, or the type specified by *type* is not a valid mutex type.
[ENOMEM]	Insufficient memory exists to initialize the mutex.
[EPERM]	The caller does not have privileges to perform this.

Associated Routines

```
tis_mutex_destroy()
tis_mutex_init()
tis_mutex_lock()
tis_mutex_trylock()
tis_mutex_unlock()
```

4.12 New RTL LIB Routines

Table 4–2 lists the new routines provided with OpenVMS Version 8.2. The routines apply to both OpenVMS Alpha and OpenVMS I64 systems unless otherwise specified.

Table 4–2 RTL LIB Routines

Routine	Description
LIB$CVTS_FROM_INTERNAL_TIME	Convert internal time to external time (S-floating value).
LIB$CVTS_TO_INTERNAL_TIME	Convert external time to internal time (S-floating value).
LIB$EMODS	Perform extended multiply and integerize for S-floating values.
LIB$EMODT	Perform extended multiply and integerize for T-floating values.
LIB$I64_CREATE_INVO_CONTEXT	Allocate and initialize an invocation context block. (I64 only)
LIB$I64_FREE_INVO_CONTEXT	Deallocate an invocation context block. (I64 only)
LIB$I64_GET_INVO_CONTEXT	Get the invocation context of any active procedure. (I64 only)
LIB$I64_GET_CURR_INVO_CONTEXT	Gets the current invocation context of any active procedure. (I64 only)
LIB$I64_GET_CURR_INVO_HANDLE	Get current invocation handle. (I64 only)
LIB$I64_GET_FR	Get floating-point register value. (I64 only)
LIB$I64_GET_GR	Get general register value. (I64 only)
LIB$I64_GET_INVO_HANDLE	Get invocation handle. (I64 only)
LIB$I64_GET_PREV_INVO_CONTEXT	Get previous invocation context. (I64 only)
LIB$I64_GET_PREV_INVO_HANDLE	Get previous invocation handle. (I64 only)
LIB$I64_GET_UNWIND_HANDLER_FV	Given a pc_value, find the function value (address of the procedure descriptor) for the condition handler, if present, and write it to handler_fv. (I64 only)
LIB$I64_INIT_INVO_CONTEXT	Initialize an invocation context block that has already been allocated. (I64 only)
LIB$GET_UIB_INFO	Return information from the unwind information block.
LIB$I64_GET_GR	Get general register value. (I64 only)
LIB$I64_GET_UNWIND_LSDA	Find Address of Unwind Information Block Language-Specific Data. (I64 only)
LIB$I64_GET_UNWIND_OSSD	Find address of the unwind information block operating system-specific data area. (I64 only)
LIB$I64_IS_AST_DISPATCH_FRAME	Determine whether a given PC value represents an AST. (I64 only)

(continued on next page)

Table 4–2 (Cont.) RTL LIB Routines

Routine	Description
LIB$I64_IS_EXC_DISPATCH_FRAME	Determine whether a given PC value represents an exception dispatch frame. I64 only.
LIB$I64_PREV_INVO_END	Free memory used to process unwind descriptors. (I64 only)
LIB$I64_PUT_INVO_REGISTERS	Put invocation registers.
LIB$I64_SET_FR	Write context of invocation context block. (I64 only)
LIB$I64_SET_GR	Write invocation block general register value. (I64 only)
LIB$I64_SET_PC	Write pc_copy value of invocation context block. (I64 only)
LIB$LOCK_IMAGE	Lock an image in the process working set.
LIB$MULTS_DELTA_TIME	Multiply a delta time by an S-floating scalar.
LIB$POLYS	Evaluate Polynomials routine (S-floating values).
LIB$POLYT	Evaluate polynomials (T-floating values).
LIB$UNLOCK_IMAGE	Unlock an image from the process working set.

See the *HP OpenVMS RTL Library (LIB$) Manual* for more information.

4.12.1 Change to LIB$GETDVI routine (I64 only)

A new argument, *pathname*, has been added to the RTL LIB routine LIB$GETDVI for use on OpenVMS I64 systems. See the *HP OpenVMS RTL Library (LIB$) Manual* for more information.

4.13 New RTL OTS Routines

Table 4–3 lists the new OTS routines. All routines are provided for both OpenVMS Alpha and OpenVMS I64 systems unless otherwise specified.

Table 4–3 RTL OTS Routines

Routine	Description
OTS$CNVOUT_S	Convert an S-floating value to a character string.
OTS$CNVOUT_T	Convert a T-floating value to a character string.
OTS$CVT_T_S	Convert numeric text to an S-floating value.
OTS$CVT_T_T	Convert numeric text to a T-floating value.
OTS$DIVCS_R3	Return an S-floating complex result of a division on complex numbers.
OTS$DIVCT_R3	Return a T-floating complex result of a division on complex numbers.

(continued on next page)

Table 4–3 (Cont.) RTL OTS Routines

Routine	Description
OTS$MULCT_R3	Calculate the complex product of two complex values; returns a T-floating complex number.
OTS$POWCSCS_R3	Raise a complex base to an S-floating complex exponent.
OTS$POWCSCT_R3	Raise a complex base to a T-floating complex exponent.
OTS$POWCSJ	return the complex result of raising an S-floating complex base to an integer exponent.
OTS$POWCTJ	return the complex result of raising a T-floating complex base to an integer exponent.
OTS$POWSJ	Raise an S-floating base to a longword exponent.
OTS$POWSLU	Raise an S-floating-point base to an unsigned longword.
OTS$POWSS	Raise an S-floating base to an S-floating or longword integer exponent.
OTS$POWTLU	Raise a T-floating-point base to an unsigned longword.
OTS$POWTJ	Raise an T-floating base to a longword integer exponent.
OTS$POWTT	Raise a T-floating base to a T-floating or longword integer exponent.

For more information, see the *HP OpenVMS RTL General Purpose (OTS$) Manual*.

4.14 Patch Utility Now Available on OpenVMS Alpha and OpenVMS I64

The Patch utility, formerly only on OpenVMS VAX systems, is now available on OpenVMS Alpha and OpenVMS I64 systems. By default, PATCH/ABSOLUTE is involved, which patches a file at absolute virtual addresses. For more information about PATCH/ABSOLUTE and related parameters, see the *HP OpenVMS DCL Dictionary*. Additional documentation on the Patch utility can be found in the *OpenVMS VAX Patch Utility Manual*, available on the OpenVMS Documentation web site under Archived Manuals, or in online Help inside the Patch utility.

4.15 New and Revised System Services

Table 4–4 summarizes system services that are new in OpenVMS Version 8.2.

Table 4–4 New System Services

System Service	Description
SYS$CLEAR_UNWIND_TABLE	Clears unwind table (UT) information.
SYS$GET_UNWIND_ENTRY_INFO	On I64 systems, gets fixed-up unwind entry information.
SYS$GOTO_UNWIND_64	On Alpha and I64 systems, unwinds the call stack.

(continued on next page)

Table 4–4 (Cont.) New System Services

System Service	Description
SYS$IEEE_SET_PRECISION_MODE	On I64 systems, modifies the IEEE precision mode and, optionally, returns the previous value.
SYS$IEEE_SET_ROUNDING_MODE	On I64 systems, modifies the IEEE rounding mode and, optionally, returns the previous value.
SYS$RPCC_64	On Alpha and I64 systems, returns a 64-bit, process-based, high-resolution time counter.
SYS$SET_RETURN_VALUE	On Alpha and I64 systems, sets the return values or condition codes in the Mechanism Array, independent of the architecture.
SYS$SET_UNWIND_TABLE	On I64 systems, registers or extends unwind table (UT) information.

For more detailed information, refer to the *HP OpenVMS System Services Reference Manual*. For additional information about the UNWIND system service routines, refer to the *HP OpenVMS Calling Standard*.

Table 4–5 summarizes system services that have been revised in OpenVMS Version 8.2.

Table 4–5 Revised System Services

System Service	Description
SYS$CHECK_FEN	On I64 systems, the bitmask has two bits: bit 0 for the low floating-point bank and bit 1 for the high floating-point bank.
SYS$CREATE_GPFN	SEC$M_UNCACHED flag applies only to I64 systems.
SYS$CRELNT	LNM$M_NO_ALIAS does not apply to clusterwide logical name tables.
SYS$CREMBX	Value of prmflg changed.
SYS$CRMPSC_GDZRO_64	SS$_INSF_SHM_REG condition value added.
SYS$CRMPSC_GPFN_64	Revised to reflect new behaviors of Alpha and I64 systems.
SYS$CRMPSC_PFN_64	SEC$M_UNCACHED flag applies only to I64 systems.
SYS$DEQ	Extended Lock Value Block information added.
SYS$ENQ	Extended Lock Value Block information added.
SYS$GETLKI	Extended Lock Value Block information added.

(continued on next page)

Table 4–5 (Cont.) Revised System Services

System Service	Description
SYS$GETDVI	The following new item codes have been added: ACCESSTIMES_RECORDED AVAILABLE_PATH_COUNT ERASE_ON_DELETE ERROR_RESET_TIME HARDLINKS_SUPPORTED MOUNT_TIME MOUNTVER_ELIGIBLE MPDEV_AUTO_PATH_SW_CNT MPDEV_MAN_PATH_SW_CNT MVSUPMSG NOCACHE_ON_VOLUME NOHIGHWATER NOSHARE_MOUNTED ODS2_SUBSET0 ODS5 PATH_AVAILABLE PATH_NOT_RESPONDING PATH_POLL_ENABLED PATH_SWITCH_FROM_TIME PATH_SWITCH_TO_TIME PATH_USER_DISABLED PROT_SUBSYSTEM_ENABLED SCSI_DEVICE_FIRMWARE_REV TOTAL_PATH_COUNT VOLUME_EXTEND_QUANTITY VOLUME_MOUNT_GROUP VOLUME_MOUNT_SYS WRITETHRU_CACHE_ENABLED
SYS$GETRMI	Many buffer length fields changed to 8 bytes.
SYS$IEEE_SET_FP_CONTROL	Revised to reflect new behaviors of Alpha and I64 systems.
SYS$INIT_VOL	New item codes added: INIT$_ERASE_ON_DELETE and INIT$_ERASE_ON_INIT, INIT$_VOLUME_LIMIT
SYS$LKWSET	Revised to reflect new behaviors of Alpha and I64 systems.
SYS$LKWSET_64	Revised to reflect new behaviors of Alpha and I64 systems.
SYS$MGBLSC_GPFN_64	SEC$M_UNCACHED flag ignored on I64 systems.
SYS$MOUNT	Item code MNT_DENSITY revised.
SYS$SETFLT	Small change made to FLT$M_EXECUTABLE.
SYS$SETFLT_64	Small change made to FLT$M_EXECUTABLE.
SYS$SETRWN	Added system resources and process quotas are affected by resource wait mode.
SYS$SET_DEVICE	Added new condition values returned for SDV$_MP_SWITCH_PATH.
SYS$ULWSET	Revised to reflect new behaviors of Alpha and I64 systems.

(continued on next page)

Table 4–5 (Cont.) Revised System Services

System Service	Description
SYS$ULWSET_64	Revised to reflect new behaviors of Alpha and I64 systems.

Refer to the descriptions in the *HP OpenVMS System Services Reference Manual* for more information.

4.16 Time Zone Information Compiler (zic) Updates

New time indicators are added for AT field in Rule line. The letter "u" (or "g" or "z") indicates the time in AT field as UTC. The following is a detailed description of zic:

AT — Gives the time of day at which the rule takes effect. Recognized forms include:

2 — time in hours
2:00 — time in hours and minutes
15:00 — 24-hour format time (for times after noon)
1:28:14 — time in hours, minutes, and seconds
minus sign (-) — equivalent to 0

Hour 0 is midnight at the start of the day, and hour 24 is midnight at the end of the day. Any of these forms may be followed by the letter w if the given time is local "wall clock" time, the letter s if the given time is local "standard" time, or the letter u (or g or z) if the given time is universal time. In the absence of an indicator, wall clock time is assumed.

4.17 Traceback Facility

The Traceback facility for I64 systems includes a new symbolize routine, TBK$I64_SYMBOLIZE. This routine, which can be invoked at any time, allows an application to process a condition invoked by an exception.

The following information describes this routine.

TBK$I64_SYMBOLIZE

The Traceback symbolize routine, TBK$I64_SYMBOLIZE, allows an application program to process an exception condition.

Calling Convention

#pragma pointer_size save
#pragma pointer_size 64

```
int32 tbk$i64_symbolize(
   uint64 const pc,
   struct dsc64$descriptor * const filename_desc,
   struct dsc64$descriptor * const library_module_desc,
   uint64 * const record_number,
   struct dsc64$descriptor * const image_desc,
   struct dsc64$descriptor * const module_desc,
   struct dsc64$descriptor * const routine_desc,
   uint64 * const listing_lineno,
   uint64 * const rel_pc);
#pragma pointer_size restore
```

Input

pc	Executable instruction to be "tracebacked" by value in process space.

Output

filename_desc	String descriptor in which to return the name of the file containing the code, by reference.
library_module_desc	String descriptor in which to return the name of the text library file containing the code, by reference. Returned only if applicable.
record_number	A 64-bit unsigned integer in which to return the record number (that is, record number n specifies the nth line of the file specified by filename_desc), by reference.
image_desc	String descriptor in which to return the image name, by reference.
module_desc	String descriptor in which to return the module name, by reference.
routine_desc	String descriptor in which to return the routine name, by reference.
listing_lineno	A 64-bit unsigned integer in which to write the compiler listing line number, by reference.
rel_pc	A 64-bit unsigned integer in which to write the relative PC value, by reference.

Return Value

TBK$_NORMAL is returned on successful completion Other unsuccessful completion codes may be returned if an error occurs.

Description

The Traceback facility for I64 systems includes a new symbolize routine, TBK$I64_SYMBOLIZE. This routine, which can be invoked at any time, allows an application to process a condition invoked by an exception in an alternate way. Normal traceback processing generates "traceback stack" information onto SYS$OUTPUT (that is, a series of pc values, one for each stack level). If an application wants to independently process traceback information, it can invoke the traceback handling directly.

An application can specify any executable pc value within its image address space and be returned any or all of the return arguments described in the Output section, above. The return information specifies the location of listing line and/or record number of the source code line that generated the object code that includes the pc value specified. This information can also specify the image name, the module name, and the routine name. A relative pc value, which specifies either the image relative or module relative value, is also returned.

The ability for Traceback to return the values requested is dependent upon whether traceback information was requested during the compilation and linker steps of image generation.

Notes

Specifying any output argument (value) as zero causes the argument to be ignored.

To link an application that refers to TBK$I64_SYMBOLIZE, include the following within a linker option file:

```
SYS$SHARE:TRACE.EXE/shareable
```

4.18 XDELTA New Features

The following list summarizes new features of the OpenVMS XDELTA Debugger running on OpenVMS Alpha and I64 systems:

- New commands: ;D and ;T
- Better symbolization
- XDELTA supports access to the following Itanium® registers:
  - General registers: R0 through R127
  - Floating registers: FP0 through FP127
  - Application registers: AR0 through AR127
  - Branch registers: BR0 through BR7
  - Control registers: CR0 through CR63
  - Predicate registers: P0 through P63
  - Miscellaneous registers: PC, PS, CFM
  - Software implementation of Alpha hardware registers

For more information, see the *HP OpenVMS Delta/XDelta Debugger Manual*.

5 Associated Products Features

This chapter describes significant new features of OpenVMS operating system associated products. For a listing and directory information about the OpenVMS associated products, refer to the *Read Before Installing* letter appropriate for your operating system.

5.1 ATI RADEON 7500 Graphics

OpenVMS Version 8.2 will provide support for 2D multi-head and 3D graphics on Integrity servers after the initial release of the operating system. Support is planned to be available in the first half of 2005 and will be announced on the OpenVMS Web site:

```
http://h71000.www7.hp.com/new/index.html
```

5.2 Common Data Security Architecture (CDSA) Now Supports a New Encryption Type

CDSA now supports a new encryption type as of CDSA Version 2.1 (OpenVMS Version 8.2). The Advanced Encryption Standard (AES) is now available as one of the standard Encryption types supported by the CDSA SSLeay Cryptographic Service Provider.

The following describes a simple AES encryption/decryption program that uses CDSA, along with the necessary files to build it on OpenVMS. The program consists of two source files (AES.C and DO_AES.C), and two build files (BUILD_AES.COM and AES.OPT). These files can be found in the CDSA AES example directory in `SYS$COMMON:[SYSHLP.EXAMPLES.CDSA.AES]`.

You must initialize CDSA before this program is run. This needs to be done on a one-time basis, by executing the following command:

```
$ @SYS$STARTUP:CDSA$INITIALIZE
```

To build the AES example program, copy the example files into a local build area and run the BUILD_AES command file, as follows:

```
$ copy SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.AES]*.* local_build_area
$ SET DEF local_build_area
$ @AES_BUILD
```

You can run the resulting AES.EXE file as a foreign command by specifying the following:

```
$ AES :== $ local_build_area AES.EXE
```

The program can then be run with the following options:

-e : encrypt with supplied key (requires -k switch)
-d : decrypt with supplied key (requires -k switch)

-h : specifies that the supplied key is a 32,48, or 64 typed character hexadecimal number
-k key : use key "key" (single quotes are necessary if used with -h)

To encrypt MYFILE.TXT using an ASCII key with the AES example program, issue the following command:

```
$ aes -e -k "xyzzy" MYFILE.TXT MYFILE.AES
```

To decrypt the same file, you issue this command:

```
$ aes -d -k "xyzzy" MYFILE.AES MYFILE.TXT
```

To encrypt/decrypt using a hexadecimal (hex) key, use a key length of exactly 64 typed characters (32 hex bytes) and the -h switch, as follows:

```
$ aes -e -k 0123456789abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmnopqr -h MYFILE.TXT MYFILE.AES
$ aes -d -k 0123456789abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmnopqr -h MYFILE.AES MYFILE.TXT
```

Note

For a hexadecimal key:

- 64 characters are used for 256-bit AES (32 hex bytes, 256 bits). (This example is included in CDSA.)
- 48 characters are used for 192-bit AES (24 hex bytes, 192 bits).
- 32 characters are used for 128-bit AES (16 hex bytes, 128 bits).

To change this example to a 128- or 192-bit AES example, do following:

1. Edit `aes.c`
2. Change the key size from key[32] to:

   key[24] for 192-bit AES
   key[16] for 128-bit AES

3. Edit `do_aes`
4. Change `key.KeyHeader.AlgorithmId = CSSM_ALGID_EVP_AES_256;` to:

   `key.KeyHeader.AlgorithmId = CSSM_ALGID_EVP_AES_192;` for 192-bit AES
   `key.KeyHeader.AlgorithmId = CSSM_ALGID_EVP_AES_128;` for 128-bit AES

5. Rebuild

5.3 Kerberos for OpenVMS

Kerberos Version 2.1 for OpenVMS is based on MIT Kerberos V5 Release 1.2.6 with CERT patches up to Release 1.2.8. Support for both Kerberos clients and servers is provided on OpenVMS I64, OpenVMS Alpha, and OpenVMS VAX.

New features in Kerberos for OpenVMS Version 2.1 include the `ktutil` command, which invokes a menu from which an administrator can read, write, or edit entries in a Kerberos V5 `keytab` or V4 `srvtab` file.

Kerberos performs authentication as a trusted third-party authentication service by using conventional (shared secret key) cryptography. Kerberos provides a means of verifying the identities of principals, without relying on authentication by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will. After a client and server have used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity.

For more detailed information, refer to *HP Open Source Security for OpenVMS, Volume 3: Kerberos*.

For information about downloading the latest version of Kerberos for OpenVMS, see the following World Wide Web address:

```
http://h71000.www7.hp.com/openvms/products/kerberos/
```

For additional information about Kerberos, see the MIT Kerberos web site at the following World Wide Web address:

```
http://web.mit.edu/kerberos/www/
```

5.4 HP SSL for OpenVMS

HP SSL Version 1.2 is based on OpenSSL 0.9.7d. (Previous versions of HP SSL were based on OpenSSL 0.9.6g.) This release includes fixes to security vulnerabilities reported on September 30 and November 4, 2003, and March 17, 2004 at `http://www.openssl.org/news/`.

Support for HP SSL is provided on OpenVMS I64, OpenVMS Alpha, and OpenVMS VAX.

New features in HP SSL Version 1.2 include OCSP (Online Certificate Status Protocol), AES (Advanced Encryption Standard), and Elliptic Curve cryptography. These features are described in the following list:

- OCSP (Online Certificate Status Protocol)

  The Online Certificate Status Protocol allows an application to more quickly determine the status of a certificate than it can by using Certificate Revocation Lists (CRLs). This is achieved by allowing the server or client application to request certificate status information from a Validation Authority (VA) in real time, rather than relying on CRL information that is issued from a Certificate Authority (CA) on a periodic basis (weekly or monthly). The VA and CA can be the same entity, but are not required to be.

- AES (Advanced Encryption Standard)

  The Advanced Encryption Standard (AES) is a new Federal Information Processing Standard (FIPS) Publication that specifies a cryptographic algorithm for use by U.S. Government organizations to protect sensitive (unclassified) information. The AES is also widely used on a voluntary basis by organizations, institutions, and individuals outside of the U.S. Government and outside of the United States. Rijndael has been selected as the AES algorithm.

  The AES was developed to replace DES, but Triple DES will remain an approved algorithm (for U.S. Government use) for the foreseeable future. Single DES is being phased out of use.

  The AES will specify three key sizes: 128, 192 and 256 bits. Assuming that one could build a machine that could recover a 56-bit DES key in a second, it would take that machine approximately 149 trillion years to crack a 128-bit AES key.

- Elliptic Curve cryptography

  Elliptic curves are simple functions that can be drawn as gently looping lines in the (x,y) plane. Elliptic curves can provide versions of public-key methods that, in some cases, are faster and use smaller keys, while providing an equivalent level of security. Their advantage comes from using a different kind of mathematical group for public-key arithmetic.

  RSA, SPEKE, Diffie-Hellman, and many other public-key methods can easily work with elliptic curves.

Secure Sockets Layer (SSL) is the open standard security protocol for the secure transfer of sensitive information over the Internet. HP SSL addresses these three fundamental security concerns about communication over the Internet and other TCP/IP networks:

- SSL server authentication

  Allows a user to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check whether a server's certificate and public ID are valid and have been issued by a Certificate Authority (CA) listed in the client's list of trusted CAs. Server authentication is used, for example, when a PC user is sending a credit card number to make a purchase on the web and wants to check the receiving server's identity.

- SSL client authentication

  Allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check whether a client's certificate and public ID are valid and have been issued by a Certificate Authority (CA) listed in the server's list of trusted CAs. Client authentication is used, for example, when a bank is sending confidential financial information to a customer and wants to check the recipient's identity.

- An encrypted SSL connection

  Requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thereby providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism that automatically detects whether data has been altered in transit.

For more detailed information, refer to *HP Open Source Security for OpenVMS, Volume 2: HP SSL for OpenVMS*.

For information about downloading the latest version of HP SSL for OpenVMS, see the following World Wide Web address:

```
http://h71000.www7.hp.com/openvms/products/ssl/
```

For additional information about OpenSSL, see the OpenSSL web site at the following World Wide Web address:

```
http://www.openssl.org/
```

5.5 HP TCP/IP Services for OpenVMS Version 5.5

This release of OpenVMS includes a new version of TCP/IP Services for OpenVMS that supports I64 systems.

TCP/IP Services Version 5.5 is supported only with this release of OpenVMS and runs on both Alpha and I64 systems. VAX systems continue to be supported by TCP/IP Services Version 5.3. Note, in a cluster, use the corresponding version of TCP/IP that is appropriate for your version of the operating system. For information on cluster configurations and warranted pair support, refer to the *Guidelines for OpenVMS Cluster Configurations*.

The following new features are included in Version 5.5:

Libpcap Library and TCPDUMP updates	– Support for libpcap API – Support for TCPDUMP Version 3.8.3
IPv6 Configuration Support and Enhancements	– The IPv6 configuration procedure has been enhanced to provide more configuration options. – FailSAFE IP support for IPv6, including new ifconfig commands – SSH support for IPv6 – PATHWORKS Internet Protocol (PWIP) driver support for IPv6
Support for Network Time Protocol (NTP)	– Supports NTP Version 4.2.0. Retains backward compatibility with NTP Version 3 and NTP Version 2, but not with NTP Version 1. – Provides increased security over NTP Version 1.

For further information about the enhancements and corrections in TCP/IP Services V5.5, refer to the TCP/IP Services V5.5 Release Notes.

6 Host-Based Minimerge (HBMM) in Volume Shadowing for OpenVMS

This chapter provides the following information:

- HBMM configuration requirements
- HBMM restrictions
- HBMM in a mixed-version or mixed-architecture OpenVMS Cluster system
- Overview of full merge and minimerge operations
- Overview of host-based minimerge (HBMM)
- HBMM policy specification syntax
- Rules governing HBMM policies
- Guidelines for establishing HBMM policies
- Configuring and managing HBMM
- New system parameters that affect HBMM
- Use of /DEMAND_MERGE when HBMM is enabled
- Prioritizing merge and copy operations
- Visible impact of transient state events

In addition to the HBMM features, new capabilities for prioritizing merge and copy operations, as described in Section 6.12, are provided.

6.1 HBMM Configuration Requirements

The configuration requirements for enabling HBMM on an OpenVMS Cluster system are:

- In a cluster of Alpha and HP Integrity server systems, all HP Integrity server systems must be running OpenVMS I64 Version 8.2 and all OpenVMS Alpha systems must be running OpenVMS Version 7.3-2 or Version 8.2.
- In a cluster of Alpha and VAX systems, Alpha systems must be running OpenVMS Alpha Version 7.3-2 or higher and all VAX systems must be running OpenVMS VAX Version 7.3.

  For supported OpenVMS Cluster configurations of Alpha and HP Integrity server systems, refer to *HP OpenVMS Version 8.2 New Features and Documentation Overview*.
- Sufficient available memory to support bitmaps, as described in Section 6.5.1.

6.2 HBMM Restrictions

The following restrictions pertain to the configuration and operation of HBMM in an OpenVMS Cluster system.

6.2.1 Cluster Configuration Restrictions

An HBMM enabled shadow set can only be mounted on HBMM capable systems. However, systems running versions of OpenVMS that support write bitmaps can coexist in a cluster with systems that support HBMM, but these systems cannot mount an HBMM enabled shadow set. The following OpenVMS versions support write bitmaps but do not include HBMM support:

- OpenVMS VAX Version 7.3 systems
- OpenVMS Alpha Versions 7.2-2 through Version 7.3-2. (Version 7.3-2 supports HBMM if the Volume Shadowing HBMM kit is installed.)

For OpenVMS Version 8.2, the earliest version of OpenVMS Alpha that is supported in a migration or warranted configuration is OpenVMS Alpha Version 7.3-2.

Caution

The inclusion in a cluster of a system that does not support write bitmaps turns off HBMM in the cluster and deletes all existing HBMM and minicopy bitmaps.

6.2.2 Shadow Set Member Restrictions

HBMM can be used with all disks that are supported by Volume Shadowing for OpenVMS except disks on HSJ, HSC, and HSD controllers.

6.2.3 System Parameter Restrictions

Host-based minimerge operations can only take place on a system that has an HBMM master bitmap for that shadow set. If you set the system parameter SHADOW_MAX_COPY to zero on all the systems that have a master bitmap for that shadow set, HBMM cannot occur on any of those systems.

Furthermore, full merges will not occur on any of the other systems (that lack a master bitmap) on which the shadow set is mounted, even if SHADOW_MAX_COPY is set to 1 or higher.

If a merge is required on a shadow set that is mounted on some systems that have HBMM master bitmaps and on some systems that do not, then the systems that do not have an HBMM master bitmap will not perform the merge as long as the shadow set is mounted on a system with an HBMM master bitmap. See Section 6.12.8 for information on how to recover from this situation.

6.3 HBMM in a Mixed-Version or Mixed-Architecture OpenVMS Cluster System

HBMM is supported by OpenVMS Alpha Version 8.2 and OpenVMS I64 Version 8.2. HBMM is also supported by OpenVMS Alpha Version 7.3-2 with an HBMM kit.

HBMM does not require that all cluster members have HBMM support, but does require that *all* cluster members support write bitmaps.

Earlier versions of OpenVMS that support write bitmaps are:

- OpenVMS Alpha Version 7.2-2 and higher
- OpenVMS VAX Version 7.3

If a system in the cluster does not support write bitmaps, neither HBMM nor minicopy functionality can be used in that cluster. Furthermore, if a system that does not support write bitmaps joins a cluster whose members do support them, all existing HBMM and minicopy bitmaps are deleted.

Once an HBMM-capable system mounts a shadow set, and HBMM is enabled for use, only the cluster members that are HBMM capable can mount that shadow set.

Whereas minicopy requires that all cluster members have minicopy support, HBMM requires only that all cluster members support write bitmaps but does not require that they all support HBMM.

Enhanced Shadowing Features

To enforce this restriction (and to provide for future enhancements), shadow sets using the HBMM feature are marked as having Enhanced Shadowing Features. This designation is included in the SHOW SHADOW DSA*n* display, as are the particular features that are in use, as shown in the following example:

```
$ SHOW SHADOW DSA0
_DSA0:    Volume Label: TST0
  Virtual Unit State:   Steady State
  Enhanced Shadowing Features in use:
    Host-Based Minimerge (HBMM)

  VU Timeout Value      3600    VU Site Value          0
  Copy/Merge Priority   5000    Mini Merge       Enabled
  Served Path Delay     30

  HBMM Policy
    HBMM Reset Threshold: 50000
    HBMM Master lists:
      Any 1 of the nodes: RAIN,SNOW
    HBMM bitmaps are active on RAIN
    Modified blocks since bitmap creation: 254

  Device $252$DKA0
    Read Cost            2     Site 0
    Member Timeout       10

 Device $252$DKA100            Master Member
   Read Cost            501   Site 0
   Member Timeout       10
$
```

Once a shadow set is marked as using Enhanced Shadowing Features, it remains so until it is dismounted on all systems in the cluster. When you remount the shadow set, the features being requested will be reevaluated. If the shadow set is no longer using any enhanced features, then it will be noted on the display and this shadow set will be available for mounting even on nodes that do not support the enhanced features.

Systems that are not HBMM capable will fail to mount HBMM shadow sets. However, if HBMM is not used by the specified shadow set, the shadow set can be mounted on prior versions of OpenVMS that are not HBMM capable.

Mount Utility Messages

If a MOUNT command for an HBMM shadow set is issued on a system that supports bitmaps but is not HBMM capable, an error message is displayed. (As noted in Section 6.2, systems running versions of Volume Shadowing for OpenVMS that support bitmaps but are not HBMM capable can be members of the cluster with systems that support HBMM, but they cannot mount HBMM shadow sets.)

The message varies, depending on the number of members in the shadow set and the manner in which the mount is attempted. The mount may appear to hang (for 30 seconds or so) while the Mount utility attempts to retry the command, and then fails.

The errors that can be generated by the Mount utility failing to mount an HBMM shadow set include the following:

```
%MOUNT-F-DEVBUSY, mount or dismount in progress on device
%MOUNT-F-XSMBRS, maximum number of shadow members exceeded
```

A Mount utility remedial kit that eliminates the delay and displays a more useful message may be available in the future for earlier versions of OpenVMS that support bitmaps.

Once a shadow set is marked as an HBMM shadow set, it remains so marked until it is dismounted from all systems in the cluster. When you remount a shadow set, if it is no longer using HBMM, it can be mounted on prior versions of OpenVMS that are not HBMM capable.

6.4 Overview of Full Merge and Minimerge Operations

The purpose of either a full merge or minimerge recovery operation is to compare data on shadow set members to ensure that all of them contain identical data on every logical block. Each block is identified by its logical block number (LBN). During the recovery operation, application I/O continues but at a slower rate. A full merge or minimerge operation is managed by one of the OpenVMS systems that has the shadow set mounted. Throughout this manual, **minimerge operation** and **merge operation** refer to a minimerge recovery operation and a merge recovery operation, respectively.

A full merge or minimerge operation is initiated by any of the following events:

- A system failure results in the possibility of incomplete application writes.
- A shadow set enters mount verification and then times out or aborts mount verification, under certain conditions (as described in Section 6.4.2).
- A system manager issues a SET SHADOW/DEMAND_MERGE command.

6.4.1 Merge Resulting from a System Failure

When a system with a mounted shadow set fails, if a write request is made to a shadow set and the system fails before a completion status is returned to the application, the data *might* be inconsistent on the shadow set members:

- All members might contain the new data.
- All members might contain the old data.
- Some members might contain new data and others might contain old data.

The exact timing of the failure during the original write request determines the outcome. Volume Shadowing for OpenVMS ensures that corresponding LBNs on each shadow set member contain the *same* data (old or new), when the application issues a read to the virtual unit.

Note

Volume Shadowing for OpenVMS guarantees that data is the same on all members of the shadow set, but it cannot guarantee that a write request that was in flight when a system failed is recorded on the shadow set. The volume *might* contain the data from the last write request, depending on when the failure occurred. In this regard, the shadow set does not differ from a nonshadowed device. The application should be designed to function properly in either case.

6.4.2 Merge Resulting from Mount Verification Timeout

A shadow set that enters mount verification and either times out or aborts mount verification will enter a merge state if the following conditions are true:

- There are outstanding write I/O requests in the shadow driver's internal queues on the system or systems on which it has timed out.
- The shadow set is mounted on other systems in the cluster.

The system on which the mount verification timed out (or aborted mount verification) notifies the other systems on which the shadow set is mounted that a merge operation is needed, and then it will disable the shadow set. (It does not dismount it.)

For example, if a shadow set is mounted on eight systems and mount verification times out on two of them, those two systems check their internal queues for write I/O. If any write I/O is found, the shadow set will need to be merged.

6.4.3 Merge Resulting from Use of SET SHADOW/DEMAND_MERGE

The SET SHADOW/DEMAND_MERGE command initiates a merge of a specified shadow set or of all shadow sets. This qualifier is useful if the shadow set was created with the INITIALIZE/SHADOW command without the use of the /ERASE qualifier.

The SET SHADOW command was introduced in OpenVMS Alpha Version 7.3-2. For more information about using SET SHADOW/DEMAND_MERGE, refer to *HP OpenVMS DCL Dictionary* and to the *HP Volume Shadowing for OpenVMS* manual.

6.4.4 Comparison of Merge and Minimerge Operations

In a full merge operation, the members of a shadow set are compared with each other to ensure that they contain the same data. This is done by performing a block-by-block comparison of the entire volume. This can be a very lengthy procedure.

A minimerge operation can be significantly faster. By using information about write operations that were logged in volatile controller storage or in a write bitmap on an OpenVMS system, volume shadowing merges only those areas of the shadow set where write activity occurred. This avoids the need for the entire volume scan that is required by full merge operations, thus reducing consumption of system I/O resources.

Prior to the introduction of HBMM, minimerge was controller-based and available only on the HSJ, HSC, and HSD controllers.

6.5 Overview of HBMM

HBMM depends on bitmaps and policies to provide the information required for minimerge operations. Depending on your computing environment, one HBMM policy, a DEFAULT policy that you specify, might be sufficient.

Before you can use HBMM for recovery of a shadow set, the following conditions must be true:

- An HBMM policy exists.
- An HBMM policy is associated with a shadow set.
- The shadow set is mounted on one or more systems that are specified in the HBMM policy.

When a policy is associated with a shadow set and the shadow set is mounted on several systems, bitmaps specific to that shadow set are created.

The systems selected from the master list, as specified in the HBMM policy definition, can perform a minimerge operation because they possess the master bitmaps. All other systems on which the shadow set is mounted possess a local bitmap for each master bitmap.

6.5.1 Bitmaps: Master and Local

For a given bitmap, there is exactly one master version on some system in the cluster and a local version on every other system that has the associated shadow set mounted. A minimerge operation can occur only on a system with a master bitmap. A shadow set can have up to six HBMM master bitmaps. Multiple master bitmaps for the same shadow set are equivalent but they do have different bitmap IDs.

The following example shows two master bitmaps for DSA12, one on RAIN and one on SNOW, each with a unique bitmap ID:

```
$ SHOW DEVICE/BITMAP DSA12

Device         BitMap    Size     Percent      Type of   Master  Active
Name             ID     (Bytes)   Populated     Bitmap     Node
DSA12:         00020007     8364         0%     Minimerge  RAIN      Yes
               00010008     8364         0%     Minimerge  SNOW     Yes
```

If only one master bitmap exists for the shadow set, and the system with the master bitmap fails or is shut down, the bitmap is gone; that is, the remaining local versions are automatically deleted. Local bitmaps cannot be used for recovery.

If multiple master bitmaps were created for the shadow set and at least one remains, that master bitmap can be used for recovery. HP recommends the use of multiple master bitmaps, especially for multiple-site cluster systems. Multiple master bitmaps increase the likelihood of an HBMM operation rather than a full merge in the event of a system failure.

Bitmaps require additional memory. The calculation is based on the shadow set volume size. For every gigabyte of storage of a shadow set mounted on a system, 2 KB of bitmap memory is required on that system for each bitmap. For example, a shadow set with a volume size of 200 GB of storage and 2 bitmaps uses 800 KB of memory on every system on which it is mounted.

6.5.2 HBMM Policies

A policy specifies the following attributes for one or more shadow sets:

- Names of systems that are eligible to host a master bitmap.
- Number of systems that will host a master bitmap (not to exceed six). If this number is omitted, the first available six systems of the systems you specified are selected.
- Threshold (in 512-byte blocks) at which the bitmaps are reset. If omitted, the threshold defaults to 50,000 blocks.

You can assign almost any name to a policy. However, the reserved names DEFAULT and NODEFAULT have specific properties that are described in Section 6.7. You can also create a policy without a name and assign it to a specific shadow set. An advantage of a named policy is that it can be reused by specifying only its name.

Multiple policies can be created to customize the minimerge operations in a cluster.

You use the SET SHADOW/POLICY command with HBMM specific qualifiers to define, assign, deassign, and delete policies and to enable and disable HBMM on a shadow set. SET SHADOW/POLICY is the only user interface for specifying HBMM policies. You cannot use the MOUNT command to define a policy. You can define a policy before the shadow set is mounted. (Policies can be associated with shadow sets in other ways as well, as described in Section 6.7.)

6.6 HBMM Policy Specification Syntax

An HBMM policy specification consists of a list of HBMM policy keywords enclosed with parentheses. The HBMM policy keywords are MASTER_LIST, COUNT, and RESET_THRESHOLD. Of the three keywords, only MASTER_LIST must be specified. If COUNT and RESET_THRESHOLD are omitted, default values are supplied. (For examples of policy specifications, see Section 6.9.1 and *HP OpenVMS DCL Dictionary*.)

The use of these keywords and the rules for specifying them are described in this section.

MASTER_LIST=*system-list*

The MASTER_LIST keyword is used to identify a set of systems as candidates for a master bitmap. The *system-list* value can be a single system name; a parenthesized, comma-separated list of system names; or the asterisk (*) wildcard character. For example:

MASTER_LIST=*node1*
MASTER_LIST=(*node1*,*node2*,*node3*)
MASTER_LIST=*

When the system list consists of a single system name or the wildcard character, parentheses are optional.

An HBMM policy must include at least one MASTER_LIST. Multiple master lists are optional. If a policy has multiple master lists, the entire policy must be enclosed with parentheses, and each constituent master list must be separated by a comma as shown in the following example:

```
(MASTER_LIST=(node1,node2), MASTER_LIST=(node3,node4))
```

There is no significance to the position of a system name in a master list.

COUNT=*n*

The COUNT keyword specifies the number of the systems, which are named in the master list, that can have a master bitmap. Therefore, the COUNT keyword must be associated with a specific master list by enclosing both with parentheses.

A COUNT value of *n* means that you want master bitmaps on any *n* systems in the associated master list. It does not necessarily mean that the first *n* systems in the list are chosen.

The COUNT keyword is optional. When omitted, the default value is the the number of systems in the master list or the value of 6, whichever is less. You cannot specify more than one COUNT keyword for any one master list.

The following two examples are valid policies:

```
(MASTER_LIST=(node1,node2,node3), COUNT=2)
```

```
(MASTER_LIST=(node1,node2,node3),COUNT=2),(COUNT=2, MASTER_LIST=(system4,system5,system6)
```

In contrast, the following example is not valid because the COUNT keyword is not grouped with a specific master list:

```
(MASTER_LIST=(node1,node2), MASTER_LIST=(node4,node5), COUNT=1)
```

RESET_THRESHOLD=*n*

The RESET_THRESHOLD keyword specifies the number of blocks that can be set before the bitmap is eligible to be cleared. Each bit that is set in a master bitmap corresponds to a set of blocks that needs to be merged. Therefore, the merge time can be influenced by this value.

Bitmaps are eligible to be cleared when the RESET_THRESHOLD is exceeded. However, the reset is not guaranteed to occur immediately when the threshold is crossed. For additional information about choosing a value for this attribute, see Section 6.8.2 and Section 6.10.2.

A single reset threshold value is associated with any given HBMM policy. Therefore, the RESET_THRESHOLD keyword cannot be specified more than once in a given policy specification. Because its scope is the entire policy, the RESET_THRESHOLD keyword cannot be specified inside a constituent master list when the policy uses multiple master lists.

When the RESET_THRESHOLD keyword is omitted, the value of 50000 is used by default.

The following policy example includes an explicit reset threshold value:

```
(MASTER_LIST=*, COUNT=4, RESET_THRESHOLD=100000)
```

6.7 Rules Governing HBMM Policies

The following rules govern the creation and management of HBMM policies. The rules are based on the assumption that a shadow set is mounted on a system that supports HBMM.

Policies and Their Attributes

- A policy can be assigned to a shadow set by specifying only its attributes. The number of policies that you can assign in this way is limited only by the number of shadow sets that are supported on a system.
- A shadow set can have only one HBMM policy associated with it at a time.
- Policies are in effect clusterwide.

- Policy names must conform to the following rules:
  - A policy name can be from 1 to 64 characters in length and is case insensitive.
  - Only letters, numbers, the dollar sign ($), and the underscore (_) are allowed.
- A policy name must be specified in full; abbreviations are not allowed.
- A named policy can be assigned to a shadow set only by the SET SHADOW/POLICY=HBMM=*policy-name* command.
- The limit on user-defined, named policies is 128.

DEFAULT and NODEFAULT Policies

The named policies DEFAULT and NODEFAULT have special properties, as summarized in the following sections:

- DEFAULT
  - A DEFAULT policy is useful if the majority of the shadow sets in a cluster are expected to use an identical policy.
  - You can create a DEFAULT policy by defining a named policy with the reserved name DEFAULT. No predetermined DEFAULT policy is provided by HP.
  - When a policy with the reserved name of DEFAULT is defined, this policy is associated with a shadow set by any of the following operations:
    + Mount of a shadow set without an associated policy

      The DEFAULT policy, if defined, is applied to a shadow set in the absence of an assigned policy (including the NODEFAULT policy). For example, when shadow set DSA1 is mounted on an HBMM-capable system, an attempt is made to apply an HBMM policy, if one exists, that is specific to DSA1. (To verify whether a device-specific policy exists and to display specific policies, see Section 6.9.9.)

      If a policy has not been defined specifically for DSA1, an attempt is made to apply the DEFAULT policy. If the DEFAULT policy exists, the attributes of that policy are applied to DSA1.
    + End of merge of a shadow set without an associated policy
    + Use of SET SHADOW/ENABLE=HBMM command
  - If a shadow set has a policy association and that policy association is deleted, it is then eligible for the DEFAULT policy, if one was established for your cluster.
- NODEFAULT Policy
  - The NODEFAULT policy specifies that the shadow set to which it is applied will not use HBMM; no HBMM bitmaps are created anywhere in the cluster for this shadow set.
  - In a cluster where a DEFAULT policy has been defined, the NODEFAULT policy can be used to prevent specific shadow sets from receiving the default policy.
  - The NODEFAULT policy cannot be deleted or redefined.

Assignment and Activation of a Policy

- A policy can be assigned to a shadow set before the shadow set is mounted on any system in the cluster.
- If a policy has been assigned, it is activated by the first mount of a shadow set on a system capable of having a master bitmap.
- Assigning a policy implicitly enables HBMM on a mounted shadow set if it is mounted on a system that can create a master bitmap. Consider DSA1 that is mounted on system MAPLE. When DSA1 was mounted, no HBMM policy was set for DSA1, nor was there a DEFAULT policy that would be applied. Later, the following command is used:

  ```
  $ SET SHADOW DSA1:/POLICY=HBMM=(MASTER=(MAPLE), COUNT=1)
  ```

  Because DSA1 is already mounted on system MAPLE, HBMM becomes enabled as a result of the policy assignment (see Section 6.9.2).
- Any attempt to enable HBMM by means of the SET SHADOW DSA*n* /ENABLE=HBMM returns a failure if a shadow set is not mounted on a system that has a master bitmap, or if the policy has not been defined.
- As new systems join the cluster, they inherit the policies in existence in that cluster.

Changes to Policies

- Named policies can be created, changed, and deleted at will. Changes made to a named policy are *not* inherited by any mounted shadow set that was assigned the previous version of that named policy.
- The association of a policy with a mounted shadow set cannot be changed if HBMM is enabled for that shadow set. HBMM must first be disabled on that shadow set, and then a different policy can be assigned to it.
- Any policy change is clusterwide.

Life of a Policy

- All policies remain in effect in a cluster as long as at least one system remains active. However, if all systems are shut down, all policy definitions and associations are deleted. The policies must be defined and assigned again when the systems form the cluster. Therefore, HP recommends that you define your desired HBMM policies in your system startup procedures before you mount your shadow sets.
- Policy assignments persist across the disabling of HBMM or the dismounting of the shadow set as long as at least one system in the cluster remains active.

6.8 Guidelines for Establishing HBMM Policies

Establishing HBMM policies is likely to be an ongoing process as configurations change and as you learn more about how HBMM works and how it affects various operations on your systems. This section describes a number of considerations to help you determine what policies are appropriate for your configuration.

The settings depend on your hardware and software configuration, the computing load, and your operational requirements. These guidelines should assist you with choosing the initial settings for your configuration. As you observe the results in your configuration, you can make further adjustments to suit your computing environment.

6.8.1 Selecting the Systems to Host Master Bitmaps

There are several factors to consider when choosing the number of master bitmaps to specify in a policy and the systems that will host the master bitmaps. The first issue is how many master bitmaps should be used in the configuration. Six is the maximum per shadow set. The use of each additional master bitmap has a slight impact on write performance and also consumes memory on each system (as described in Section 6.5.1).

Using only one master bitmap creates a single point of failure; if the system hosting the master bitmap fails, then this shadow sets undergoes a full merge. Therefore, the memory consumption must be weighed against the adverse effects of a full merge. Using six master bitmaps provides the greatest defense against performing full merges.

Another issue when selecting a system to host the master bitmap is the I/O bandwidth of the various systems. Keep in mind that minimerges are always performed on a system that has a master bitmap. Therefore, low-bandwidth systems, such as satellite cluster members, are not good candidates.

The disaster tolerance of the configuration is also important in the decision process. Specifying systems to host master bitmaps at multiple sites helps ensure that a minimerge is performed if connectivity to an entire site is lost. A two-site configuration should ensure that half the master bitmaps are at each site, and a three-site configuration should ensure that one third of the master bitmaps are at each of the three sites.

6.8.2 Setting the Bitmap RESET_THRESHOLD Value

HBMM bitmaps keep track of writes to a shadow set. The more bits that are set in the bitmap, the greater the amount of merging that is required in the event of a minimerge. HBMM clears the bitmap (after ensuring that all outstanding writes have completed so that the members are consistent) when certain conditions are met (see Section 6.10.2). A freshly cleared bitmap, with few bits set, performs a minimerge much more quickly.

The bitmap reset, however, can be costly to I/O performance. Before a bitmap reset can occur, all write I/O to the shadow set must be paused and any write I/O that is in flight must be completed. Then the bitmap is cleared. This is done on all systems on a per shadow set basis. Therefore, avoid a reset threshold setting that causes frequent resets.

You can view the number of resets performed by using the SHOW SHADOW command, as shown in the following example:

```
$ SHOW SHADOW DSA1031
_DSA1031: Volume Label: HBMM1031
  Virtual Unit State:   Steady State
  Enhanced Shadowing Features in use:
        Host-Based Minimerge (HBMM)

  VU Timeout Value      3600    VU Site Value          0
  Copy/Merge Priority   5000    Mini Merge       Enabled
  Served Path Delay     30
  HBMM Policy
    HBMM Reset Threshold: 50000
    HBMM Master lists:
      Up to any 2 of the systems: LEMON, ORANGE
      Any 1 of the systems: MELON, PEACH
    HBMM bitmaps are active on LEMON, MELON, ORANGE
  HBMM Reset Count      76    Last Reset    29-JAN-2004 10:13:53.90
    Modified blocks since last bitmap reset: 40132

 .
 .
 .
$
```

Writes that need to set bits in the bitmap are slightly slower than writes to areas that are already marked as having been written. Therefore, if many of the writes to a particular shadow set are concentrated in certain "hot" files, then the reset threshold should be made large enough so that the same bits are not constantly set and then cleared.

On the other hand, if the reset threshold is too large, then the advantages of HBMM are reduced. For example, if 50% of the bitmap is populated (that is, 50% of the shadow set has been written to since the last reset), then the HBMM merge will take approximately 50% of the time of a full merge.

When selecting a threshold reset value, you need to balance the effects of bitmap resets on I/O performance with the time it takes to perform HBMM minimerges. The goal is to set the reset value as low as possible (thus decreasing merge times) while not affecting application I/O performance. Too low a value will degrade I/O performance. Too high a value causes HBMM merges to take extra time.

Note

You can change the reset threshold while a policy is in effect.

6.8.3 Using Multiple Policies

HBMM policies are defined to implement the decisions regarding master bitmaps. Some sites might find that a single policy can effectively implement the decisions. Other sites might need greater granularity and therefore implement multiple policies.

The most likely need for multiple policies is when the cluster includes enough high-bandwidth systems that you want to ensure that the merge load is spread out. Remember, minimerges occur only on systems that host a master bitmap. So, if 12 systems with high bandwidth are set up to perform minimerge or merge operations (the system parameter SHADOW_MAX_COPY is greater than zero on all systems), then you should ensure that the master bitmaps are spread out among these high-bandwidth systems.

Multiple HBMM policies are also useful when shadow sets need different bitmap reset thresholds. The master list can be the same for each policy, but the threshold can differ.

6.9 Configuring and Managing HBMM

This section describes the major tasks for configuring and managing HBMM.

6.9.1 How to Define an HBMM Policy

The SET SHADOW/POLICY=HBMM command is used to define HBMM policies. You can define multiple policies for your environment. The following examples show how to define two policies, a DEFAULT policy and POLICY_1, a named policy.

To define the policy named DEFAULT:

```
$ SET SHADOW/POLICY=HBMM=(MASTER_LIST=*)/NAME=DEFAULT
```

In this example, a DEFAULT policy is created for the cluster. The use of the asterisk wildcard (*) means that any system can host a master bitmap. The omission of the keyword COUNT=*n* means that up to six systems (the default and the current maximum supported) can host a master bitmap. The DEFAULT policy is inherited at mount time by shadow sets that have not been assigned a named policy.

The following example defines a named policy (POLICY_1), specifies the systems that are eligible to host a master bitmap, limits to two the number of systems that can host a master bitmap, and specifies a higher threshold (default is 50,000 blocks) to be reached before clearing the bitmap.

```
$ SET SHADOW /POLICY=HBMM=( -
_$ (MASTER_LIST=(NODE1,NODE2,NODE3), COUNT=2), -
_$ RESET_THRESHOLD=100000) -
_$ /NAME=POLICY_1
```

For the full DCL syntax for the SET SHADOW/POLICY=HBMM command, see *HP OpenVMS DCL Dictionary*.

6.9.2 How to Assign an HBMM Policy to a Shadow Set

You can assign a policy, named or unnamed, to a shadow set. To assign an existing named policy, use the following command:

```
$ SET SHADOW DSAn:/POLICY=HBMM=policy-name
```

To assign an unnamed policy to a shadow set, use the same command, but in place of the policy name, specify the attributes of the policy you want to use. For example:

```
$ SET SHADOW DSA1:/POLICY=HBMM=(MASTER_LIST=(NODE1, NODE2, NODE3), COUNT=2)
```

In this example, the default bitmap reset value of 50,000 blocks takes effect because the RESET_THRESHOLD keyword was omitted.

6.9.3 How to Activate HBMM on a Shadow Set

HBMM is automatically activated on a shadow set under the following conditions:

- An HBMM policy exists for a given shadow set and that shadow set is then mounted on one or more systems defined in the master list.
- An HBMM policy is created for a mounted shadow set and at least one system that has it mounted is defined in the master list.

You can also activate HBMM with the SET SHADOW/ENABLE=HBMM command, provided a policy exists and the shadow set is mounted on a system defined in the master list of the shadow set policy, and the count has not been exceeded.

6.9.4 How to Disable HBMM on a Shadow Set

To disable HBMM on a shadow set, use the following command:

```
$ SET SHADOW DSAn:/DISABLE=HBMM
```

Reasons for disabling HBMM on a shadow set include:

- To change the policy associated with it.
- To delete the policy associated with it.
- To mount the shadow set on a system that does not support HBMM. You must disable HBMM first and then dismount it from all the HBMM capable systems on which it is mounted before you can mount it on a system that does not support HBMM.

HBMM remains disabled until you either re-enable it or define a new policy for the shadow set.

6.9.5 How to Remove a Policy Association from a Shadow Set

Before removing a policy association from a shadow set, HBMM must be disabled, if active. Then you can remove a policy association from a shadow set by entering the following command:

```
$ SET SHADOW DSAn:/POLICY=HBMM/DELETE
```

This command removes any policy that had been set for this shadow set, making the shadow set eligible for the DEFAULT policy. If a DEFAULT policy exists, it will be assigned the next time the shadow set becomes eligible for a policy, for example, at the end of a merge or when you issue the SET SHADOW/ENABLE=HBMM command.

6.9.6 How to Change a Policy Assignment of a Shadow Set

To change a policy assigned to a shadow set, you must first disable HBMM, as described in Section 6.9.4, and then assign another policy to the shadow set. To apply a different policy, specify one by name or specify policy attributes (thereby creating an “unnamed” policy), as described in Section 6.9.2. Specifying a new policy (or policy attributes) for a shadow set replaces the prior policy. The use of the command shown in Section 6.9.5 is not required when you are changing the policy assignment.

6.9.7 How to Delete a Named Policy from the Cluster

You can delete a named policy with the /DELETE qualifier, as shown in the following example:

```
$ SET SHADOW /POLICY=HBMM/NAME=policy-name/DELETE
```

This command deletes the policy whose name you specified and takes effect across the cluster. It does not delete the policy from any shadow set to which it was already assigned.

Note

You cannot delete the NODEFAULT policy.

6.9.8 How to Apply a Changed DEFAULT Policy

The DEFAULT policy can be changed at any time. However, if a previous definition of the DEFAULT was associated with a shadow set, a subsequent change to the definition of the DEFAULT policy is not retroactively applied to that shadow set. In this regard, the DEFAULT policy behaves just like any other named policy.

This section shows how to apply a changed DEFAULT policy.

Initially, the following DEFAULT policy was associated with DSA20 when it was mounted:

```
$ SET SHADOW/POLICY=HBMM=(MASTER=(NODE1,NODE2,NODE3),COUNT=2)/NAME=DEFAULT
$ MOUNT/SYSTEM DSA20:/SHADOW=($1$DGA20,$1$DGA21) VOL_20
```

Subsequently, the DEFAULT policy is redefined by the following command. This redefined policy allows any node in the cluster to be eligible for an HBMM master bitmap:

```
$ SET SHADOW/POLICY=HBMM=(MASTER=*,COUNT=2)/NAME=DEFAULT
```

You can apply the redefined DEFAULT policy to DSA20 by using the following commands:

```
$ SET SHADOW DSA20:/DISABLE=HBMM
$ SET SHADOW DSA20:/POLICY=HBMM/DELETE
$ SET SHADOW DSA20:/ENABLE=HBMM
```

Note that you must explicitly delete the HBMM policy associated with DSA20 in order for DSA20 to become eligible for the current DEFAULT policy. This step is required because, when HBMM is disabled on DSA20, the policy (MASTER=(NODE1,NODE2,NODE3),COUNT=2) remains associated with DSA20.

An alternative way to apply the updated DEFAULT policy to DSA20 is to take advantage of the fact that the DEFAULT policy is a named policy. This method requires only two commands, as shown next:

```
$ SET SHADOW DSA20:/DISABLE=HBMM
$ SET SHADOW DSA20:/POLICY=HBMM=DEFAULT
```

6.9.9 How to Display Policies

You can display policies with the SHOW SHADOW command. You can display:

- The policy associated with a specified shadow set
- The definition of a named policy
- All shadow sets in a cluster with policy assignments, together with the definition of each policy
- All named policies and their definitions that exist on a cluster

Displaying the Policy of a Specific Shadow Set

To display the policy associated with a specific shadow set, issue the following command:

```
$ SHOW SHADOW DSAn:/POLICY=HBMM
```

An example of the resulting output follows:

```
$ SHOW SHADOW DSA999:/POLICY=HBMM
HBMM Policy for device _DSA999:
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: NODE1,NODE2,NODE3
  Any 1 of the nodes: NODE4,NODE5
  Up to any 2 of the nodes: NODE6,NODE7,NODE8
```

Displaying the Definition of a Named Policy

To display the definition of a named policy, issue the following command:

```
$ SHOW SHADOW/POLICY=HBMM/NAME=policy-name
```

The following display shows the definition of the PEAKS_ISLAND policy:

```
$ SHOW SHADOW/POLICY=HBMM/NAME=PEAKS_ISLAND
HBMM Policy PEAKS_ISLAND
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: NODE1,NODE2,NODE3
  Any 1 of the nodes: NODE4,NODE5
  Up to any 2 of the nodes: NODE6,NODE7,NODE8
```

Displaying All Shadow Sets with Policy Assignments

To display all shadow sets in a cluster with policy assignments, along with the definition of each policy, use the following command:

```
$ SHOW SHADOW/POLICY=HBMM
```

The following display results from this command:

```
$ SHOW SHADOW/POLICY=HBMM
HBMM Policy for device _DSA12:
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: NODE1,NODE2
 HBMM bitmaps are active on NODE1,NODE2
 Modified blocks since bitmap creation: 254

HBMM Policy for device _DSA30:
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: FLURRY,FREEZE,HOTTUB

HBMM Policy for device _DSA99:
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: NODE1,NODE2,NODE3
  Any 1 of the nodes: NODE4,NODE5
  Up to any 2 of the nodes: NODE6,NODE7,NODE8

HBMM Policy for device _DSA999:
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: NODE1,NODE2,NODE3
  Any 1 of the nodes: NODE4,NODE5
  Up to any 2 of the nodes: NODE6,NODE7,NODE8
```

Displaying All Named Policies on a Cluster

To display the named policies that exist on a cluster, along with their definitions, issue the following command:

```
$ SHOW SHADOW/POLICY=HBMM/NAME
```

The named policies are displayed in the order in which they were created. The following display results from this command:

```
$ SHOW SHADOW/POLICY=HBMM/NAME
HBMM Policy DEFAULT
    HBMM Reset Threshold: 50000
    HBMM Master lists:
      Up to any 6 nodes in the cluster

HBMM Policy PEAKS_ISLAND
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: NODE1,NODE2,NODE3
  Any 1 of the nodes: NODE4,NODE5
  Up to any 2 of the nodes: NODE6,NODE7,NODE8

HBMM Policy POLICY_1
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: NODE1,NODE2,NODE3
  Any 1 of the nodes: NODE4,NODE5

HBMM Policy ICE_HOTELS
 HBMM Reset Threshold: 50000
 HBMM Master lists:
  Up to any 2 of the nodes: QUEBEC,ICELND,SWEDEN
  Any 1 of the nodes: ALASKA,GRNLND
```

6.9.10 How to Display the Merge Status of Shadow Sets

You can check the merge status of each shadow set member by issuing the SHOW SHADOW/MERGE DSA*n* command. The /MERGE qualifier returns one of the following messages:

- Merge is not required.
- Merge is pending.
- Merge is in progress on node *node-name*.

An example of the display produced by the SHOW SHADOW/MERGE DSA*n* command follows:

```
$ SHOW SHADOW/MERGE
  Device      Volume Name   Status
  _DSA1010      FOOBAR        Merging (10%)
```

If a copy operation (instead of the merge operation) is currently active, the display shows the percentage of the merge that has completed and the percentage of the copy that has completed with the designation “Copy Active,” as follows:

```
$ SHOW SHADOW/MERGE
  Device      Volume Name   Status
  _DSA1010    FOOBAR        Merging (23%), Copy Active  (77%) on CSGF1
```

6.9.11 How to Prevent Merge Operations on a System

You can prevent merge operations on a system in two ways:

- Set SHADOW_MAX_COPY to 0
- Set the priority for merge and copy operations to zero for every shadow set mounted on the system using the SET SHADOW/PRIORITY=0 DSA*n* for each shadow set.

6.9.12 Considerations for Multiple-Site OpenVMS Cluster Systems

Only the systems that have an HBMM master bitmap for a particular shadow set are able to perform HBMM recovery on that shadow set. If a merge recovery is required on a shadow set and no systems in the cluster have an HBMM master bitmap for that shadow set, a full merge will be performed.

Therefore, to minimize the need to perform a full merge, you should use policies that attempt to maintain at least one HBMM master bitmap at each site in a multiple-site OpenVMS Cluster system. The ability to specify multiple master lists in an HBMM policy is expressly designed for this purpose. You should specify a separate MASTER_LIST for each site.

For example, consider a 3-site OpenVMS Cluster system with 12 cluster members:

Site 1:	Member systems NYN1, NYN2, NYN3, and NYN4
Site 2:	Member systems CTN1, CTN2, CTN3, and CTN4
Site 3:	Member systems NJN1, NJN2, NJN3, and NJN4

The following definition of a DEFAULT policy would provide for up to two HBMM master bitmaps at each site:

```
$ SET SHADOW/NAME=DEFAULT/POLICY=HBMM=( -
_$ (MASTER_LIST=(NYN1,NYN2,NYN3,NYN4), COUNT=2), -
_$ (MASTER_LIST=(CTN1,CTN2,CTN3,CTN4), COUNT=2), -
_$ (MASTER_LIST=(NJN1,NJN2,NJN3,NJN4), COUNT=2))
```

Specifically, this policy requests master bitmaps at any two of the systems in the first master list, any two of the systems in the second master list, and any two of the systems in the third master list.

Note that you cannot accomplish this type of distribution by listing the systems in a particular order within a single MASTER_LIST. This is because the order in which the systems are specified in a master list does not affect the order in which the systems are considered when HBMM master bitmaps are created. When an event occurs that warrants the creation of an HBMM master bitmap, the creation of these bitmaps is done in random order by the systems that have the shadow set mounted. In the following example, the likelihood of system NYN1 getting a master bitmap is the same for either POLICY_A or POLICY_B:

```
$ SET SHADOW/NAME=POLICY_A/POLICY=HBMM=( -
_$ (MASTER_LIST=(NYN1,CTN1,NJN1,NYN2,CTN2,NJN2),COUNT=3))

$ SET SHADOW/NAME=POLICY_B/POLICY=HBMM=( -
_$ (MASTER_LIST=(NJN2,CTN2,NYN2,NJN1,CTN1,NYN1),COUNT=3))
```

6.10 New System Parameters That Affect HBMM

Two new system parameters are introduced in this version: SHADOW_REC_DLY and SHADOW_HBMM_RTC.

6.10.1 SHADOW_REC_DLY Parameter

SHADOW_REC_DLY governs system behavior after a system failure or after a shadow set is aborted. The value of the SHADOW_REC_DLY parameter is added to the value of the RECNXINTERVAL parameter to determine how long a system waits before it attempts to manage a merge or copy operation on any shadow sets that it has mounted.

SHADOW_REC_DLY can be used to better predict which systems in an OpenVMS Cluster will perform recovery operations. This is done by setting lower values of SHADOW_REC_DLY on systems that are preferred to handle recovery operations and higher values of SHADOW_REC_DLY on the remaining systems.

SHADOW_REC_DLY is a dynamic parameter; its range is 0 to 65535 seconds. The default value is 20 seconds.

For more information about controlling which systems perform the merge or copy operations, see Section 6.12.5.

6.10.2 SHADOW_HBMM_RTC Parameter

SHADOW_HBMM_RTC is used to specify, in seconds, how frequently the modified block count of the HBMM bitmap is compared with the reset threshold. If the modified block count exceeds the reset threshold, the bitmap is zeroed.

The reset threshold is specified by the RESET_THRESHOLD keyword in the /POLICY qualifier of the SET SHADOW command. This comparison is performed for all shadow sets mounted on the system that have HBMM bitmaps.

When the comparison is made, the modified block count might exceed the reset threshold by a small increment or by a much larger amount. The difference depends on the write activity to the volume and the setting of SHADOW_HBMM_RTC.

SHADOW_HBMM_RTC is a dynamic parameter; its range is 60 to 65535 seconds. The default value is 150 seconds.

You can view the reset threshold setting and the modified block count, since the last reset, in the SHOW SHADOW display. For guidelines for setting reset threshold values and a sample SHOW SHADOW display, see Section 6.8.2. For a SHOW SHADOW display that includes a modified block count greater than the reset threshold value, see Example 9 for SHOW SHADOW in *HP OpenVMS DCL Dictionary*.

6.11 Use of /DEMAND_MERGE When HBMM Is Enabled

If a shadow set is HBMM enabled and is actively using HBMM, then the SET SHADOW/DEMAND_MERGE DSA*n*: command causes a minimerge operation to occur. To force a full merge instead of a minimerge operation, you must disable HBMM on the shadow set before issuing the SET SHADOW/DEMAND_MERGE DSA*n*: command. For information about disabling HBMM, see Section 6.9.4.

The /DEMAND_MERGE qualifier of the SET SHADOW command is used primarily to force a merge operation on shadow sets that were created with the INITIALIZE/SHADOW command without specifying the /ERASE qualifier. The /DEMAND_MERGE qualifier ensures that all blocks on the shadow set are the same, including those blocks that are not currently allocated to files. System managers can use this command at their convenience during off-peak demands on their computing environment.

If the /ERASE qualifier was not used when the shadow set was created with the INITIALIZE/SHADOW command, and the SET SHADOW/DEMAND_MERGE DSA*n*: command has not been executed, then the overhead of a full merge operation on this shadow set is even higher than is normally encountered after a system failure.

System managers can also use the SET SHADOW/DEMAND_MERGE DSA*n*: command for the following reasons:

- If the ANALYZE/DISK/SHADOW command found differences between the members of the shadow set. For more information, refer to the ANALYZE/DISK/SHADOW command description in the *HP Volume Shadowing for OpenVMS* manual.
- If they want to measure the impact that a minimerge or a full merge has on their I/O throughput.

6.12 Prioritizing Merge and Copy Operations

Starting with this version of Volume Shadowing, greater control is available to system managers for managing merge and copy operations. This greater control is made possible by two new qualifiers to the SET SHADOW command, /PRIORITY=*n* and /EVALUATE=RESOURCES, and a new system parameter, SHADOW_REC_DLY. With these parameters, system managers can:

- Prioritize shadow sets for merge and copy operations on a per-system basis.
- Control which system performs a merge or copy operation of a particular shadow set.
- Make changes to the SHADOW_MAX_COPY system parameter, which take effect immediately.

6.12.1 Default Management of Merge and Copy Operations

If a system fails or if it aborts a shadow set, most commonly through mount verification, such an action is termed a significant event. When one of these significant events occurs, all systems in the cluster are notified automatically. This notification causes all shadow server processes to stop any full merge or full copy operations and release all the resources performing these operations. Then every system can reallocate its resources to newer, higher-priority work.

After a predetermined delay, each system with a nonzero SHADOW_MAX_COPY setting begins to process the shadow sets that are in a transient state, according to their priority. The predetermined delay is governed by the new system parameter SHADOW_REC_DLY. (For more information about SHADOW_REC_DLY, see Section 6.12.5.) Every system allocates available SHADOW_MAX_COPY resources based on a shadow set's priority.

A shadow set is said to be in a **steady state** when none of the following operations is pending or active:

- Minimerge
- Minicopy
- Full copy
- Full merge

If a shadow set has one or more of these operations pending, or one operation active, it is said to be in a **transient state**. While a combination of these transient states is valid, only one operation at a time can be performed.

For example, suppose HBMM is not enabled. After a device is added to a shadow set, it is marked as being in a full copy transient state. If the system on which this shadow set is mounted fails, the shadow set is further marked as being in a full merge state. In this example, the full copy operation is performed before the full merge is started.

Note

The priority assigned to a shadow set does not affect the hierarchy of transient state operations.

6.12.2 Hierarchy of Transient State Operations

Shadow set operations for a specific shadow set are performed in the following order:

1. Minimerge
2. Copy (either minicopy or full copy)
3. Full merge

6.12.3 Assigning Priorities

When first mounted on a system, every shadow set is assigned a default priority of 5000. You can assign a unique priority to every mounted shadow set on a per-system basis using the SET SHADOW/PRIORITY=*n* DSA*n* command. Every shadow set can have a unique priority per system, or shadow sets can be assigned the same priority. Shadow sets with the same priority are managed in a consistent way for each release. However, the order in which shadow sets with the same priority are managed may change from release to release because of changes to the algorithm. Therefore, if the order is important, assign them different priorities.

The valid range for priority values is 0 through 10,000. The higher the assigned value, the higher the priority. To ensure that high-priority volumes are merged (or copied) before less important volumes, use this command to override the default priority assignment on a system.

A priority level of 0 has a unique meaning. It means the shadow set is not considered for merge or copy operations on this system.

Note

After the notification of a significant event and the allocation of a system's resources, it is not possible to directly affect any of the current merge or copy operations on the system by assigning a different priority level to one or more shadow sets. If you need to reprioritize one or more shadow sets, you must use another technique, as described in Section 6.12.8.

6.12.4 Displaying Priority Values

You can display the priority of a shadow set on a specific system by issuing the following command:

```
$ SHOW SHADOW/BY_PRIORITY DSAn:
```

This command displays the current priority and status of the specified shadow set. If any copy or merge operations are in progress, the node on which the operation is progressing is displayed, along with its progress. For example:

```
$ SHOW SHADOW DSA1104/BY_PRIORITY
Device    Mbr                                                   Active
 Name     Cnt  Priority     Virtual Unit State                 on Node
_DSA1104:  2     5000       Merge Active (29%)                    MAX
```

You can use the following command to display the priority level and the status for all of the shadow sets that exist on the system. The status indicates whether the shadow set is currently undergoing a copy or merge operation or whether one is required. If either or both operations are underway, the systems on which they are occurring are identified in the display, as shown in the following example:

```
$ SHOW SHADOW/BY_PRIORITY
Device    Mbr                                                   Active
 Name     Cnt  Priority     Virtual Unit State                 on Node
_DSA106:   2    10000       Steady State
_DSA108:   3     8000       Steady State
_DSA110:   3     8000       Steady State
_DSA112:   3     8000       Steady State
_DSA114:   1     7000       Steady State
_DSA116:   1     7000       Steady State
_DSA150:   2     7000       Steady State
_DSA152:         7000       Not Mounted on this node
_DSA154:   3     6000       Steady State
_DSA156:   1     6000       Steady State
_DSA159:   2     5000       Steady State
_DSA74:    3     5000       Merge Active (47%)                  CASSID
_DSA304:  2+1    5000       Merge Active (30%), Copy Active (3%)     MAX
_DSA1104:  2     5000       Merge Active (29%)                  MAX
_DSA300:  2+1    5000       Merge Active (59%), Copy Active (0%)     MAX
_DSA0:    1+2    5000       Copy Active (83%)                   CASSID
_DSA3:     2     3000       Steady State
_DSA100:   2     3000       Steady State
_DSA102:   1     3000       Steady State
_DSA104:   3     3000       Steady State
Total of 19 Operational shadow sets; 0 in Mount Verification; 1 not mounted
$
```

In this example, the 20 shadow sets on this system are displayed in their priority order. In the event of a failure of another system in the cluster that has these shadow sets mounted, the shadow sets are merged in this order on the system.

The Mbr Cnt field shows how many source members are in each shadow set. If members are being added via a copy operation, this is indicated by +1 or +2. So, 2+1 indicates two source members and one member being added. The notation 1+2 indicates only one source member and two members being copied into the set.

The summary line provides the total number of shadow sets that were found to be in the various conditions. “Operational shadow sets” are those shadow sets that are mounted with one or more members and may or may not have copy or merge operations occurring. These shadow sets are available to applications for reads and writes. “Mount Verification” indicates the number of shadow sets that are in some mount verification state. Shadow sets that have exceeded their mount verification timeout times are also included in this total.

For additional examples, see the *HP OpenVMS DCL Dictionary*.

6.12.5 Controlling Which Systems Manage Merge and Copy Operations

When a system fails or aborts a shadow set, this significant event causes every shadow set to be reassessed by all other systems with that shadow set mounted. All active minimerge, full merge, or copy operations cease at this time, returning their resources to those systems. (However, if a system is performing a *minicopy* operation, that operation continues to completion.)

Those systems wait a predetermined amount of time, measured in seconds, before each attempts to manage any shadow set in a transient state. This pause is called a significant-event recovery delay. It is the total of the values specified for two system parameters, SHADOW_REC_DLY and RECNXINTERVAL. (The default value for each is 20 seconds.)

If the value of the significant-event recovery delay is the same on all systems, it is not possible to predict which systems will manage which shadow set. However, by making the value of the significant-event recovery delay different on all systems, you can predict when a specific system will begin to manage transient-state operations.

6.12.6 Managing Merge Operations

A merge transient state is an event that cannot be predicted. The management of merge activity, on a specific system for multiple shadow sets, can be predicted if the priority level settings for the shadow sets differ.

In the following example, there are four shadow sets, and the SHADOW_MAX_COPY parameter on this system is equal to 1. The value of 1 means that only one merge or copy operation can occur at the same time. This example illustrates how the priority level is used to select shadow sets when only merge operations are involved.

Two shadow sets are assigned a priority level and two have the default priority level of 5000.

The four shadow sets DSA1, DSA20, DSA22, and DSA42, are mounted on two systems. DSA20 and DSA42 are minimerge enabled.

```
$ SET SHADOW/PRIORITY=7000 DSA1:
$ SET SHADOW/PRIORITY=3000 DSA42:
! DSA20: and DSA22: are at the default priority level of 5000
```

In this example, when one of the systems fails, all shadow sets are put into a merge-required state. After the significant-event recovery delay time elapses, this system evaluates the shadow sets, and the operations are performed in the following order:

1. A minimerge operation starts first on DSA20, even though its priority of 5000 is lower than DSA1's priority of 7000. A minimerge operation always takes precedence over other operations. DSA20 and DSA42 are both minimerge enabled, but DSA20's higher priority causes its minimerge operation to start first.
2. A minimerge operation starts on DSA42. Its priority of 3000 is the lowest of all the shadow sets, but a minimerge operation takes precedence over other operations.

3. Because there are no other minimerge capable units, DSA1, with a priority level of 7000, is selected to start a merge operation, and it runs to completion.
4. A merge operation starts on DSA22, the one remaining shadow set whose priority is the default value of 5000, and runs to completion.

6.12.7 Managing Copy Operations

A copy transient state can be predicted by the user because it is the result of direct user action. Therefore, a full copy operation caused by adding a device to a shadow set is not considered a significant event in the cluster. The copy operation is managed by the first system that has an available resource.

In the following example, assume there are four shadow sets, and the SHADOW_MAX COPY parameter on this system is equal to 1. Recall that the shadow sets that are not assigned a specific level will have a default priority level assigned.

For the following example, assume that:

- DSA1, DSA20, DSA22, and DSA42 are mounted on multiple systems.
- Only DSA42 is minimerge enabled.
- DSA22 is already in a full copy state being managed on this system.
- DSA1 has a priority level of 7000.
- DSA42 has a priority level of 3000.
- DSA20 has a priority level of 3000.
- DSA22 has a default priority level of 5000.

The user adds a device to DSA1. This is not a significant event, and this system will not interrupt the full copy operation of the DSA22 in favor of performing the DSA1 full copy operation.

To expand on this example, assume that a system fails (a significant event) before the copy operations have completed. All shadow sets will be put into a merge required state. Specifically, DSA1, DSA20, and DSA22 are put into a full merge state, and DSA42 is put into a minimerge state.

After the significant event recovery delay expires, this system begins to evaluate all the shadow sets in a transient state. The operations take place in the following order:

1. A minimerge operation starts on DSA42 and continues until completion. This operation takes priority over other operations, regardless of its priority level.
2. A copy operation starts on DSA1. The full merge operation is not started because a copy operation takes precedence over a full merge operation.
3. A merge operation is started and completed on DSA1.
4. A copy operation is started and completed on DSA22.
5. A merge operation is started and completed on DSA22.
6. A merge operation is started and completed on DSA20.

Thus, in this example, the priority level is used to direct the priority of merge and copy operations on this system.

6.12.8 Managing Transient States in Progress

SHADOW_MAX_COPY is a dynamic system parameter that governs the use of system resources by shadowing. Shadowing can be directed to immediately respond to changes in this parameter setting with the following DCL command:

```
$ SET SHADOW/EVALUATE=RESOURCES
```

This command stops all the current merge and copy operations on the system on which it is issued. It then restarts the work using the new value of SHADOW_MAX_COPY.

This command is also useful in other circumstances. For example, if a shadow set had a priority level of 0 or another low value, the SET SHADOW /PRIORITY=*n* command can be used to increase the value. Then, by using the /EVALUATE=RESOURCES qualifier, the priority of shadow sets in a transient state is reevaluated.

The /PRIORITY and /EVALUATE=RESOURCES command qualifiers can be used on the same command line.

When a significant event occurs, all of the SHADOW_MAX_COPY resources are applied. If the value of SHADOW_MAX_COPY is modified using the SYSGEN SET and WRITE ACTIVE commands, and then a SET SHADOW /EVALUATE=RESOURCES is issued, the new value of SHADOW_MAX_COPY has a direct and immediate affect.

To determine which system is controlling a transient operation, enter the following command:

```
$ SHOW SHADOW/ACTIVE DSAn:
```

To determine the priority values assigned to each shadow set, enter the following command:

```
$ SHOW SHADOW/BY_PRIORITY DSAn:
```

6.13 Visible Impact of Transient State Events

Table 6–1 summarizes the user-visible impact of transient state events from the viewpoint of a shadow set on one system in an OpenVMS Cluster system. For each type of transient state event, the effects on the shadow set, when a merge (full merge, HBMM, or controller minimerge) or copy (full or minicopy) operation was already underway, are listed. The terms Canceled, Restarted, Continued, and Suspended, described in the table key have the same meaning in this table as in Volume Shadowing for OpenVMS messages.

Note also the following characteristics of merge and copy operations:

- If both a merge and a copy are pending on a shadow set, the merge is done before the copy if and only if the merge is a minimerge. This applies to controller-based minimerges, host-based minimerges, full copies, and minicopies.
- If an event that specifies a delay occurs during the delay for some prior event, no additional delay is incurred. The merges or copies required for the current event are considered when the prior delay expires.
- If an event that specifies no delay occurs during the delay for some prior event, the merges or copies required for the “no delay” event are not considered until the prior delay expires.

Table 6–1 Visible Impact of Transient State Events

Event[1]	Shadow Set (SS) Focus	New work required	What happens to prior merge/copy on SS?				Delay[2]
			Prior full merge or HBMM	Prior controller minimerge	Prior full copy	Prior minicopy	
Failure of other system that had at least one mounted SS in common with this system.	All SSs that were mounted on failed system.	Merge required.	Canceled and is restarted.	If on failed system, it restarts. Otherwise, minimerge continues with added work.	Canceled but is eventually continued.	Canceled but is eventually continued. Continued as full copy if minicopy master bitmap was on failed system.	Yes
	All other SSs with prior merge or copy state.	No new work.	Canceled but is continued.	No change.	Canceled but is continued.	Canceled but is continued.	Yes
Failure of a system that did not have any SS mounted in common with this system.	All other SSs with prior merge or copy state.	No new work.	No change.	No change.	No change.	No Change.	Yes
SS aborted on another system, with writes in restart queue on the aborting system; SS is also mounted on this system.	Aborted SS.	Merge required.	Canceled and is restarted.	If on failed system, it restarts. Otherwise, minimerge continues with added work.	Canceled but is continued.	Canceled but is continued.	Yes
	All other SSs with prior merge or copy state.	No new work.	Canceled but is continued.	No change.	Canceled but is continued.	Canceled but is continued.	Yes
Other system dismounts a SS that has a merge or copy in progress on its system; SS is also mounted on this system.	SS dismounted on other system.	No new work.	Continued.	Restarted.	Continued.	Continued.	No

[1]Each event is described from the perspective of one system in a cluster.

[2]Delay represents a predetermined length of time that elapses before the operation begins. It is the total of the values specified for the SHADOW_REC_DLY and RECNXINTERVAL system parameters.

Key to Merge or Copy Operation Actions (same definitions as in shadowing messages)

Canceled—Operation is stopped so that it can be restarted or continued on any system that is eligible.
Restarted—Operation must start over again on the same system at LBN 0 when the operation is resumed.
Continued—Operation continues at the LBN where it left off when it was canceled or suspended.
Suspended—Operation is stopped such that the operation for that SS can be initiated, restarted, or continued only on the same system where the suspended operation was active.

(continued on next page)

Table 6–1 (Cont.) Visible Impact of Transient State Events

Event[1]	Shadow Set (SS) Focus	New work required	What happens to prior merge/copy on SS?				Delay[2]
			Prior full merge or HBMM	Prior controller minimerge	Prior full copy	Prior minicopy	
	All other SSs with prior merge or copy state.	No change.	No change.	No change.	No change.	No change.	No
A member is added to a SS that is mounted on this system.	Specified SS.	Copy required.	Canceled but is continued.	No change.	No change.	No change.	No
	All other SSs with prior merge or copy state.	No new work.	No change.	No change.	No change.	No change.	No
Mount a SS that requires a merge or copy; SS is not mounted on any other system.	Specified SS.	Copy and/or merge required.	Restarted as full merge.	Restarted as full merge.	Restarted.	Restarted as full copy.	No
SET SHADOW /DEMAND_ MERGE command issued on any system for SS mounted on this system.	Specified SS does not use controller minimerge.	Merge required.	Restarted.	Not applicable.	Canceled but is continued.	Canceled but is continued.	No
	Specified SS uses controller minimerge.	Full merge required.	Restarted	Suspended and restarted as full merge.	Canceled but is continued.	Canceled but is continued.	No
	All other SSs with prior merge or copy state.	No change.	No change.	No change.	No change.	No change.	No

[1]Each event is described from the perspective of one system in a cluster.

[2]Delay represents a predetermined length of time that elapses before the operation begins. It is the total of the values specified for the SHADOW_REC_DLY and RECNXINTERVAL system parameters.

Key to Merge or Copy Operation Actions (same definitions as in shadowing messages)

Canceled—Operation is stopped so that it can be restarted or continued on any system that is eligible.
Restarted—Operation must start over again on the same system at LBN 0 when the operation is resumed.
Continued—Operation continues at the LBN where it left off when it was canceled or suspended.
Suspended—Operation is stopped such that the operation for that SS can be initiated, restarted, or continued only on the same system where the suspended operation was active.

(continued on next page)

Table 6–1 (Cont.) Visible Impact of Transient State Events

Event[1]	Shadow Set (SS) Focus	New work required	What happens to prior merge/copy on SS?				Delay[2]
			Prior full merge or HBMM	Prior controller minimerge	Prior full copy	Prior minicopy	
SET SHADOW /EVAL=RESOURCES command issued on this system.	All SSs mounted on this system with prior merge or copy state.	No change.	Canceled but is continued.	No change.	Canceled but is continued.	Canceled but is continued.	Yes

[1]Each event is described from the perspective of one system in a cluster.

[2]Delay represents a predetermined length of time that elapses before the operation begins. It is the total of the values specified for the SHADOW_REC_DLY and RECNXINTERVAL system parameters.

Key to Merge or Copy Operation Actions (same definitions as in shadowing messages)

Canceled—Operation is stopped so that it can be restarted or continued on any system that is eligible.
Restarted—Operation must start over again on the same system at LBN 0 when the operation is resumed.
Continued—Operation continues at the LBN where it left off when it was canceled or suspended.
Suspended—Operation is stopped such that the operation for that SS can be initiated, restarted, or continued only on the same system where the suspended operation was active.

7 Linker Utility

This chapter provides an overview of the differences as well as considerations you should review before linking programs on OpenVMS I64 systems. Major topics include:

- Differences when linking on I64 systems as compared with linking on VAX and Alpha systems (Section 7.1.1)
- New aspects of linking an image on OpenVMS I64 (Section 7.1.2)
- New linker qualifiers and options (Section 7.1.3, Section 7.1.4)
- New ways to use existing linker qualifiers and options (Section 7.1.5)
- Expanded linker map file information for I64 (Section 7.1.6)

For release notes on the linker, refer to the *HP OpenVMS Version 8.2 Release Notes*.

7.1 Linker Utility New Features

The purpose of the linker is to create images (that is, files that contain binary code and data). The linker ported to OpenVMS I64 is different from the linker on OpenVMS VAX and Alpha systems because it accepts OpenVMS I64 object files and produces OpenVMS I64 images. As on OpenVMS VAX and Alpha systems, the primary type of image created is an **executable image**. This image can be activated at the DCL command line by entering the RUN command.

The OpenVMS I64 Linker also creates **shareable images**. A shareable image is a collection of procedures and data that are exported via the SYMBOL_VECTOR option that can be called by executable images or other shareable images. Shareable images are included in an input file via a link operation that creates an executable or shareable image.

When linking modules, the primary type of input file to the OpenVMS I64 Linker is the **object file**. Object files are produced by language processors such as compilers or assemblers. Because the object files produced by these compilers are unique to the Intel® Itanium® architecture, some aspects of linking modules on OpenVMS I64 systems are different from linking on VAX and Alpha systems. Overall, however, the OpenVMS I64 Linker interface as well as functional capabilities (for example, symbol resolution, virtual memory allocation, and image initialization) are similar to those on OpenVMS VAX and Alpha systems.

7.1.1 Differences When Linking on OpenVMS I64 Systems

Although linking on OpenVMS I64 systems is similar to linking on OpenVMS Alpha systems, some differences exist. The following qualifiers or options are ignored by the OpenVMS I64 linker:

- /REPLACE
- /SECTION_BINDING
- /HEADER
- PER_PAGE—The per_page keyword in /DEMAND_ZERO=PER_PAGE (the per_page keyword will be supported in a future release, although with a slightly different meaning)
- DZRO_MIN
- ISD_MAX

The following qualifiers and options are not allowed by the OpenVMS I64 Linker:

- /SYSTEM
- A file spec keyword with the /DEBUG= qualifier
- The base address must be null in CLUSTER=cluster_name,base_address . . .
- BASE=
- UNIVERSAL=

The following list contains new qualifiers that are supported by the OpenVMS I64 Linker:

- /BASE_ADDRESS
- /SEGMENT_ATTRIBUTE
- /FP_MODE
- /EXPORT_SYMBOL_VECTOR
- /PUBLISH_GLOBAL_SYMBOLS
- The GROUP_SECTIONS and SECTION_DETAILS keywords for the /FULL qualifier

These qualifiers and options are described in the following sections.

7.1.1.1 Data Types of Symbols must Match on I64

On OpenVMS Alpha, there can be a symbol that is defined as a piece of data but referenced as if it were a function. There is no way, using the Alpha object language, that this situation can be detected by the linker.

But on OpenVMS I64, the linker can detect a symbol's data type mismatch. The OpenVMS I64 linker receives information which marks a symbol as a function (FUNC), a piece of data (OBJECT) or unknown (NOTYPE). It also receives relocations which tell the linker when to create a function descriptor for symbols that are defined or referenced as a function.

If the types of a symbol are not unknown (not NOTYPE), they must match. If the definer sets the type as unknown, the referencer type cannot be a function.

For example, take the two modules FIRST.C and SECOND.C:

```
FIRST.C

#include <stdio.h>

int a;
int aa;
extern int second ();

void main () {

        printf ("The address of 'a'  is %x\n", &a);
        printf ("The address of 'aa' is %x\n", &aa);
        second ();
}
```

```
SECOND.C

#include <stdio.h>

extern int a();
extern int aa();

void second () {

        printf ("The address of 'a'  is %x\n", &a);
        printf ("The address of 'aa' is %x\n", &aa);
}
```

When you link FIRST and SECOND, you get the following informational messages (based on the symbol descriptors) and warning messages (based on the relocations) from the linker:

```
$ link first,second
%ILINK-I-DIFTYPE, symbol AA of type OBJECT cannot be referenced as type FUNC
        module: SECOND
        file: DISK$USER:[JOE]SECOND.OBJ;5
%ILINK-I-DIFTYPE, symbol A of type OBJECT cannot be referenced as type FUNC
        module: SECOND
        file: DISK$USER:[JOE]SECOND.OBJ;5
%ILINK-W-RELODIFTYPE, relocation requests the linker to build a function
descriptor for a non-function type of symbol
        symbol: A
        relocation section: .rela$CODE$ (section header entry: 19)
        relocation type: RELA$K_R_IA_64_LTOFF_FPTR22
        relocation entry: 1
        module: SECOND
        file: DISK$USER:[JOE]SECOND.OBJ;5
%ILINK-W-RELODIFTYPE, relocation requests the linker to build a function
descriptor for a non-function type of symbol
        symbol: AA
        relocation section: .rela$CODE$ (section header entry: 19)
        relocation type: RELA$K_R_IA_64_LTOFF_FPTR22
        relocation entry: 4
        module: SECOND
        file: DISK$USER:[JOE]SECOND.OBJ;5
```

To eliminate these informational and warning messages, change the type of the symbols "A" and "AA" to be a function or a piece of data. For example, change the declarations in FIRST.C to functions:

```
#include <stdio.h>

int a();
int aa();
extern int second ();

void main () {

        printf ("The address of 'a'  is %x\n", &a);
        printf ("The address of 'aa' is %x\n", &aa);
        second ();
}

int a () {
        return 1;
}

int aa () {
        return 1;
}
```

After this change, your link will be free of informationals and warnings:

```
$ link first,second
$
```

Another alternative is to change the references to "A" and "AA" in SECOND to be references to data.

7.1.1.2 Specifying Based Clusters Vary by OpenVMS Platform

Specifying a base address in the CLUSTER option is permitted only on VAX and Alpha systems. On Alpha systems, only main images are allowed to contain based clusters. VAX systems are more flexible; even shareable images are allowed to contain based clusters. On I64 systems, specifying a base address in a CLUSTER option is illegal for all images.

7.1.1.3 Handling of Initialized Overlaid Program Sections on OpenVMS I64 Systems

On Alpha and VAX systems, initializations can be done to portions of an overlaid program section. Subsequent initializations to the same portions overwrite initializations from previous modules. The last initialization performed on any byte is used as the final one of that byte for the image being linked.

On I64 systems, the ELF object language currently does not implement the feature of the Alpha and VAX object language which allows the initialization of portions of the program sections. When an initialization is made, the entire section is initialized. Subsequent initializations of this section may be performed only if the nonzero portions match in value. This is called a compatible initialization.

For example, the following condition produces different results on OpenVMS I64 systems than on Alpha systems:

Two program sections, each declaring two longwords, are overlaid. The first program section initializes the first longword and the second program section initializes the second longword with a nonzero value.

On Alpha systems, the linker is able to produce an image section with the first and second longword initialized. The VAX and Alpha object languages give the linker the section size and the Text Information Relocation (TIR) commands to initialize each longword.

On I64 systems, the linker gets sections that contain initialization data from the compilers. The linker does not perform or know about the initializations, because there are no TIR commands in ELF. The linker then produces an image segment using the last processed program section for initialization. That is, in the image either the first or second longword has a nonzero value, depending on the order in which they were linked. Although an image is produced, the linker regards this as an error and issues a message.

On I64 systems, the linker reads all the applicable initializations and checks for compatibility. Any two initializations are compatible if they are identical in the nonzero values. If they are not compatible, the linker issues the following error message:

```
%ILINK-E-INVOVRINI, incompatible multiple initializations for
 overlaid section
        section:  <section name>
        module:   <module name for first overlaid section>
        file:  <file name for first overlaid section>
        module:  <module name for second overlaid section>
        file:  <file name for second overlaid section>
```

In this error message, the linker lists the first module, which contributes a nonzero initialization, and the first module with an incompatible initialization. Note that this is not a full list of all incompatible initializations; it is just the first one the linker encounters.

In the Program Section Synopsis of the linker map, each module with a nonzero initialization is flagged as "Initializing Contribution". Use this information to identify and resolve all the incompatible initializations.

The following example shows the additional information in the map file (see Example 7–1):

```
$ cre one.c
int common_data[]={0,0,47,11};
[Exit]
$ cc /extern=common one
$
$ cre two.c
int common_data[]={0,0,47,11};
[Exit]
$ cc /extern=common two
$
$ cre three.c
int common_data[]={0,0,0,0,0,0,0,0};
[Exit]
$ cc /extern=common three
$
$ link/map one,two,three
.
.
.
```

Example 7–1 shows the program section synopsis of the linker map for the above example. Note that the Align and Attributes fields normally continue after the Length field but were modified to fit on the page.

Example 7–1 Linker Map Showing Program Section Synopsis

```
                                    +-------------------------+
                                    ! Program Section Synopsis !
                                    +-------------------------+

Psect Name      Module/Image     Base     End          Length
----------      ------------     ----     ---          ------
COMMON_DATA                    00010000 0001001F 00000020 (      32.)
                ONE            00010000 0001000F 00000010 (      16.)
                TWO            00010000 0001000F 00000010 (      16.)
                THREE          00010000 0001001F 00000020 (      32.)

Align                Attributes
-----               ------------
OCTA  4 OVR,REL,GBL,NOSHR,NOEXE,  WRT,NOVEC,  MOD
OCTA  4 Initializing Contribution
OCTA  4 Initializing Contribution
OCTA  4
```

Another example shows an incompatible initialization and the resulting linker message:

```
$ cre four.c
int common_data[]={0,0,47,11,0,0,17,4};
[Exit]
$ cc /extern=common four
$
$link one,two,three,four
%ILINK-E-INVOVRINI, incompatible multiple initializations for
 overlaid section
        section: COMMON_DATA
        module: ONE
        file: DISK$USER:[JOE]ONE.OBJ;1
        module: FOUR
        file: DISK$USER:[JOE]FOUR.OBJ;1
```

The examples were compiled with the extern common model. For VMS the default extern model is the relaxed refdef model. In that model, there is only one explicit initialization that is allowed. That is, even identical initializations result in a linker warning message, the multiply defined message. Here is an example, which uses the same source files as above:

```
$ cc one
$ cc two
$ cc three
$
$ link one, two, three
%ILINK-W-MULDEF, symbol COMMON_DATA multiply defined
        module: TWO
        file: DISK$USER:[JOE]TWO.OBJ;2
%ILINK-W-MULDEF, symbol COMMON_DATA multiply defined
        module: THREE
        file: DISK$USER:[JOE]THREE.OBJ;2
```

For an additional incompatible initialization the linker shows both messages, the INVOVRINI error and the MULDEF warning. And, for such a module, the INVOVRINI, precedes the MULDEF message. In this case, the user has to fix the MULDEF warning to get rid of the INVOVRINI error.

```
$ cc four
$
$ link one,two,three,four
%ILINK-W-MULDEF, symbol COMMON_DATA multiply defined
        module: TWO
        file: DISK$USER:[JOE]TWO.OBJ;2
%ILINK-W-MULDEF, symbol COMMON_DATA multiply defined
        module: THREE
        file: DISK$USER:[JOE]THREE.OBJ;2
%ILINK-E-INVOVRINI, incompatible multiple initializations for
 overlaid section
        section: COMMON_DATA
        module: ONE
        file: DISK$USER:[JOE]ONE.OBJ;2
        module: FOUR
        file: DISK$USER:[JOE]FOUR.OBJ;2
%ILINK-W-MULDEF, symbol COMMON_DATA multiply defined
        module: FOUR
        file: DISK$USER:[JOE]FOUR.OBJ;2
```

7.1.1.4 Behavior Difference When Linking ELF Common Symbols

The I64 linker behaves differently when ELF common symbols (or, on Alpha, relaxed refdef symbols) are linked selectively against an image that contains a definition for the same symbol. On Alpha, the linker incorrectly takes the definition from the relaxed refdef symbol in your module. The I64 linker takes the definition from the shareable image.

For example, a shareable image is linked from the following module (my_int.c):

```
#include ints
uint64 my_int ;
$cc/extern=relaxed my_int.c
$link/map/full/cross/share my_int,sys$input/opt
symbol_vector=(my_int=data)
```

Next, another C module (x.c) is selectively linked against my_$int.exe. Note that the object is compiled with the relaxed extern model. The result is a conditional reference/definition generated for my_int.

```
#include ints
uint64 my_int;
main()
{
my_int = 1;
return;
}
```

```
$cc/extern=relaxed x.c
$link/map/full/cross sys$input/opt
cluster=myclu,,,x.obj
my_int.exe/share/select
```

The Alpha linker incorrectly defines my_int from the module's conditional definition of my_int. The I64 linker correctly retrieves the definition from my_int.exe.

The Alpha linker is not being changed to preserve those cases that rely upon this behavior. The I64 linker performs the selection operation correctly.

7.1.1.5 Flags Set When /TRACEBACK, /DEBUG, and /DSF are Used

When the /TRACEBACK, /DEBUG and /DSF linker debug qualifiers are specified, the following flags are set for the image activator. These flags are set in the dynamic segment under the tag value DT_VMS_LNKFLAGS. Note that the meanings of the /TRACEBACK, /DEBUG and /DSF qualifiers have not changed from Alpha, but the meanings and the names of the flags have changed.

Flag	Meaning
VMS_LF_IMGSTA	Image execution is to begin by calling SYS$IMGSTA. The image activator includes SYS$IMGSTA as the first address in the (traditional VMS style) transfer vector.
VMS_LF_CALL_DEBUG	SYS$IMGSTA checks this flag to determine whether it calls the debugger.
VMS_LF_TBK_IN_IMG	Traceback records are present in the image file.
VMS_LF_DBG_IN_IMG	Debug information is present in the image file.
VMS_LF_TBK_IN_DSF	Traceback records are present in the DSF file.
VMS_LF_DBG_IN_DSF	Debug information is present in the DSF file.

The flags will be set according to the following table:

Qualifier	IMGSTA	CALL_ DEBUG	TBK_IN _IMG	DBG_IN _IMG	TBK_IN _DSF	DBG_IN _DSF
/NoTrace/NoDebug /NoDSF	0	0	0	0	0	0
/Trace/NoDebug /NoDSF	1	0	1	0	0	0
/NoTrace **/Debug** /NoDSF	1	1	1	1	0	0
/Trace /Debug /NoDSF	1	1	1	1	0	0
/NoTrace /NoDebug **/DSF**	0	0	0	0	1	1
/Trace /NoDebug **/DSF**	1	0	1	0	1	1
/NoTrace **/Debug** **/DSF**	1	1	1	0	1	1
/Trace /Debug **/DSF**	1	1	1	0	1	1

Notes

- The value of SYS$IMGSTA is no longer included in the image's transfer array; only a flag that indicates it is to be called. The image activator knows the value of SYS$IMGSTA.
- These flags do not appear in a DSF file. DSF files are not activated by the image activator (they have no dynamic segment and therefore no DT_VMS_ LNKFLAGS field).
- The Linker no longer supports /DEBUG=*filename* on I64 systems. Link alternative debuggers as a separate image and then define LIB$DEBUG to point to that image. SYS$IMGSTA always calls whatever is pointed to by LIB$DEBUG.

- When /DSF is specified, along with /TRACEBACK or /DEBUG, the VMS_LF_TBK_IN_IMG (traceback in image) flag will be set. This is a change in behavior from the behavior on Alpha, where traceback records were not included in the image when /TRACEBACK/DSF or /DEBUG/DSF was specified. Note that debugger records do not get copied to an image whenever /DEBUG/DSF is specified. Here, /DEBUG only causes the IMGSTA bit to be set in the image.

The following table indicates where global symbol definitions are written during a link operation for an image that does not use the SYMBOL_TABLE or SYMBOL_VECTOR option.

Qualifier	Global Symbols in Image	Global Symbols in DSF File
/NoTrace/NoDebug/NoDSF	0	0
/Trace/NoDebug/NoDSF	0	0
/NoTrace**/Debug**/NoDSF	1	0
/Trace /Debug/NoDSF	1	0
/NoTrace/NoDebug**/DSF**	0	1
/Trace/NoDebug**/DSF**	0	1
/NoTrace**/Debug/DSF**	0	1
/Trace/Debug/DSF	0	1

7.1.2 New Aspects for Linking on OpenVMS I64 Systems

The following sections describe aspects of linking images on I64 systems that are unique to the platform. Topics include:

- Understanding linkage messages (Section 7.1.2.1)
- Considerations for Images Compiled with Reduced Floating Point Model (Section 7.1.2.2)
- Considerations for Linking with ELF Groups and UNIX-style weak Symbols (Section 7.1.2.3)
- Use of the new /BASE_ADDRESS Qualifier (Section 7.1.3.1)
- Use of the new /SEGMENT_ATTRIBUTE Qualifier (Section 7.1.3.2)
- Use of the new /FP_MODE Qualifier (Section 7.1.3.3)
- Use of the new /EXPORT_SYMBOL_VECTOR and /PUBLISH_GLOBAL_SYMBOLS qualifiers (Section 7.1.3.4)
- Use of the new GROUP_SECTIONS and SECTION_DETAILS keywords for the /FULL Qualifier (Section 7.1.3.5)
- New Alignments for the PSECT_ATTRIBUTE Option (Section 7.1.4.1)

7.1.2.1 Understanding Linkage Messages

Some HP compilers produce call linkage information that verifies the linkage of a caller or a called routine to determine consistent use of the general registers across modules. Linkage information is supplied for only the general registers 0 through 31. Conflicts arise when the linkage information indicates a problem.

The following example shows a warning message that displays when the linker detects a linkage conflict:

```
%ILINK-W-LNKGERR, linkage to routine X is not compatible with
 linkage of caller
        calling module: MOD_SRC
                file: DISK:[DIR]SOURCE.OBJ;1
        target  module: MOD_TARG
                file: DISK:[DIR]TARGET.OBJ;1
        source type of JSB to target type of CALL
        register AI not provided (needed at target)
        register GP not provided (needed at target)
        IA64 register R19 (Alpha R21) -- call=PRESERVE, target=VOLATILE
```

Following the display of the source and target information are messages that indicate the nature of the conflict or conflicts that caused the warning message to be displayed. If a caller does not provide argument information in the argument information register (AI) when the target routine requires the AI, the linker will generate a message similar to the one in the above example.

In addition to missing required information, inconsistent or incompatible use of the general registers can also cause a linkage conflict. The linkage information contains register policies for the general registers, which are as follows:

- Volatile: A register with this policy may not be used to pass information between procedures, either as input or output.
- Scratch: A register with this policy may be modified by the called procedure.
- Output: A register with this policy may be used to pass information back to the calling procedure.
- Preserve: A register with this policy must have its contents saved and then restored, if it is to be used by the target routine.

The register policies of a standard linkage call are as follows:

Intel® Itanium® Register	Policy
R2	Volatile
R3	Scratch
R4 - R7	Preserve
R8 - R9	Output
R10 -R11	Scratch
R12 - R18	Volatile
R19 - R24	Scratch
R26 - R31	Scratch

A standard procedure call uses the CALL mechanism and provides the Global Pointer (GP) value and the AI register fill with the argument information.

The following table indicates compatible policies between the caller's register policies (row heading) and the target routine's register policies (column heading):

Caller	Target			
	Volatile	Scratch	Output	Preserve
Volatile	X			
Scratch		X		X
Output			X	X
Preserve				X

In the sample message:

```
IA64 register R19 (Alpha R21) - call-PRESERVE, target=VOLATILE
```

Intel® Itanium® register R19 has a caller register policy of "Preserve" but a target routine register policy of "Volatile". Based on the above table, R19 is not compatible between the caller and the target routines.

Additionally, the calling mechanism between the routines is not compatible. This occurs because the caller is performing a JumptoSuBroutine (JSB) to the target routine when the target routine is expecting a CALL mechanism to be used.

Certain Intel® Itanium® registers must be defined to have specific policies. If these registers do not contain the correct policies, the linker will issue a warning indicating that either the target linkage is invalid (1) or a message of the caller is invalid (2):

```
(1) %ILINK-W-INVLNKG, invalid target linkage for symbol INV_LNKG
```

```
(2) %ILINK-W-INVRLNKG, invalid call linkage for symbol INV_LNKG
```

The following table lists the Intel® Itanium® general registers that have specific policies:

Intel® Itanium® Register	Policy
R2	Volatile
R12 - R18	Volatile

Explanation of Potential Conflicts that Result in Linkage Messages

Some conflicts arise when performing cross language calls (for example, IMACRO to BLISS), where the calling standards for OpenVMS differ between Alpha and I64 systems. There are fewer registers preserved by default with the Intel® Itanium® version of the calling standard. Because of this, there may be assumptions about registers that are saved at the target routine and registers that are not saved.

Linkage statement mismatches also cause the linker to display warning messages. For example, if a calling routine is expecting register R12-R15 (Alpha register numbers) to be preserved but the target does not preserve them, the linker will list registers R20, R21, R30, and R31 as having a linkage problem but will include the Alpha register numbers in parentheses.

Additionally, a problem can exist when registers that are not properly declared for the function they are serving, for example, declaring registers to return values as NOPRESERVE rather than OUTPUT.

To correct these potential conflicts, examine the declaration and definition linkages for all cases the linker indicates and correct these problems to ensure a clean link operation. Because the linker considers these conflicts warnings, the completion code status of the image will not be set to SUCCESS. For a shareable image, this means that images linked against a shareable image with linkage issues will, themselves, receive a completion status other than SUCCESS.

7.1.2.2 Considerations for Images Compiled with Reduced Floating Point Model

On OpenVMS I64 systems, images using a reduced floating point model will run more quickly during interrupts because only a reduced number of floating point registers need to be saved and restored. Note that some integer arithmetic is done using floating point registers. The resulting image can be run in a reduced floating point mode only if all object modules that contribute to the image have been compiled with the reduced floating point model. The Object and Image Synopsis section of the linker map file indicates if the modules have the reduced floating point model.

7.1.2.3 Considerations for Linking with ELF Groups and UNIX-style Weak Symbols

The Intel C++ compiler introduced the use of Executable and Linking Format (ELF) Groups (that is, COMDATs) and UNIX-style Weak symbols that the Linker (and the Librarian) must correctly handle.

ELF groups in object modules can be seen as tentative code and data contributions with definitions for variables and procedures. The C++ compiler makes use of this feature for C++ templates. The compiler simply generates code and data for a template: a group. In such an environment multiple object modules will have multiple identical contributions with the same code and data. For the image, the linker selects one contribution from each group. So the result of linking multiple object modules with multiple identical groups is one instance of code and data per group in the image.

An ELF group has a name, also called group signature. Groups are as seen identical if they have the same name. Symbols can belong to a group; they are called group symbols.

As usual, group symbols can be used for definitions or references. The UNIX-style weak definition differs from the VMS-style weak definition (Alpha and VAX): there can be multiple UNIX-style weak definitions. UNIX-style weak references are similar to the Alpha and VAX style references: when they are unresolved, they do not cause an error or warning.

UNIX-style Weak symbols from an ELF Group take the first occurrence of the specific group as the defining instance. All subsequent definitions of these UNIX-style Weak symbols from other occurrences of that group are then seen as references to the first instance. UNIX-style Weak symbols are, by design, tentative. If present, and if there is no occurrence of a strong definition elsewhere among the other modules or images of the link, the first encountered UNIX-style Weak version of the symbol is regarded as the defining instance.

A strong definition overwrites a UNIX weak definition, which becomes a reference. The other UNIX weak symbols of the same group become references. This does not create a problem: all symbols, including strong definitions, are generated by the compiler or defined in the run time environment. Understanding this mechanism helps to understand the linker map.

UNIX weak symbols defined in a group can be referenced by non-weak symbols outside a group.

The cross reference list in the map has a UNIX weak tag in front of UNIX weak definitions and references. Typically, the UNIX weak tagged definitions are the result of selecting a group from an object module. The UNIX weak tagged references are usually a result of a secondary occurrence of the same group. An untagged definition with a UNIX weak tagged reference is usually a result of a strong definition overwriting a group symbol from an object module. The tagging in the cross reference table enables the user to see which module was selected as owning the defining instance of the symbol.

The cross reference does not list the group signatures (that is, their names). The ANALYZE utility can be used to find all symbols belonging to a group. Additionally, you can ask the linker for a list of all groups and their defining modules and files by using the /MAP/FULL=GROUP_SECTIONS qualifiers.

User-written templates can be linked as a shareable image. Symbols that are exported (made universal) by a shareable image are always strong definitions. With the precedence of strong definitions over UNIX weak ones, the code and data of the shareable image is the linker selected contribution. For OpenVMS I64, the grouping of symbols is preserved in the shareable image as well. In this case, where all the symbols are strong definitions, a group from a shareable image always takes precedence over all identical groups in object modules.

There is one difference between groups in a shareable image and groups in an object module. If the same group is seen in another shareable image, the linker warns of a second occurrence of the group. In this case, the resulting image may not behave as expected. Code and data from the first occurrence of the group will be used by all object modules within the defining shareable image, but code and data from the other occurrences will be used within the other defining shareable images. When linking a shareable image, make sure that all symbols of all groups making up the template implementation are exported. Otherwise, redundant code and data may appear in the resulting image, or - as mentioned above - wrong code or data may be used.

The original OpenVMS-style weak symbols continue to be processed in the same manner as before.

7.1.3 New Linker Qualifiers for OpenVMS I64

Some new linker qualifiers have been added to support linking on OpenVMS I64 systems. This section describes these features.

7.1.3.1 New /BASE_ADDRESS Qualifier

A new qualifier, /BASE_ADDRESS, is provided on I64 systems. The base address is the starting address that you want the linker to assign to an executable image. The purpose of this qualifier is to assign a virtual address for images which are not activated by the OpenVMS image activator, such as images used in the boot process. The OpenVMS image activator is free to ignore any linker-assigned starting address. This qualifier is primarily used by system developers.

The /BASE_ADDRESS qualifier does not replace the CLUSTER=,*[base-address]* option, which is illegal on OpenVMS I64. See Section 7.1.1.2.

7.1.3.2 New /SEGMENT_ATTRIBUTE Qualifier

The OpenVMS I64 Linker provides a new /SEGMENT_ATTRIBUTE qualifier that accepts two keywords: SHORT_DATA=WRITE and DYNAMIC_SEGMENT=P0 or P1. The /SEGMENT_ATTRIBUTE qualifier uses the following syntax:

```
/SEGMENT_ATTRIBUTE=([DYNAMIC_SEGMENT=(P0|P2) ], [SHORT_DATA=WRITE])
```

By default, the linker puts the dynamic segment, which contains information for the image activator, into P2 space. But by specifying DYNAMIC=P0, it forces the linker to put the dynamic segment into P0 space. As such, the DYNAMIC_ SEGMENT keyword is rarely needed.

The SHORT_DATA=WRITE keyword allows you to combine read-write and read/write short data sections into a single segment, reclaiming up to 65,535 bytes of unused read-only space. When setting SHORT_DATA to WRITE, your program may accidentally write to formerly read-only data. Therefore this qualifier is only recommended for users whose short data segment has reached the limit of 4 MB.

7.1.3.3 New /FP_MODE Qualifier

The OpenVMS I64 Linker determines the program's initial floating point mode using the floating point mode provided by the module that provides the main transfer address. Use the /FP_MODE qualifier to set an initial floating point mode only if the module that provides the main transfer address does not provide an initial floating point mode. The /FP_MODE qualifier will not override an initial floating point mode provided by the main transfer module.

The OpenVMS I64 Linker accepts the following keywords to set the floating point mode:

- D_FLOAT, G_FLOAT—Sets VAX floating point modes.
- IEEE_FLOAT[=ieee_behavior]—Sets the IEEE floating point mode to the default or a specific behavior.

The OpenVMS I64 Linker accepts the following IEEE behavior keywords:

- FAST
- UNDERFLOW_TO_ZERO
- DENORM_RESULTS (default)
- INEXACT

The OpenVMS I64 Linker also accepts a floating point mode behavior literal. For more information about the initial floating point mode, see the *HP OpenVMS Calling Standard*.

7.1.3.4 New Linker Qualifiers: /EXPORT_SYMBOL_VECTOR and /PUBLISH_GLOBAL_SYMBOLS

The /EXPORT_SYMBOL_VECTOR and /PUBLISH_GLOBAL_SYMBOLS qualifiers were added to the linker to aid users who are creating shareable images but do not know which symbols to export through the SYMBOL_VECTOR option. (Users may have been porting an application from UNIX and were unfamiliar with the application, such that they did not know which symbols to export. Another possible scenario is that because they were coding in C++, they were not able to know what the mangled names looked like.)

The new /PUBLISH_GLOBAL_SYMBOLS qualifier marks an object module so that all its global symbols can be exported in a symbol vector option. The new /EXPORT_SYMBOL_VECTOR qualifier writes a symbol vector option for each global symbol in modules marked with /PUBLISH_SYMBOL_VECTOR to the output file. When /EXPORT_SYMBOL_VECTOR is present, only the option file is written; no image file is generated. The generated option file needs to be completed with GSMATCH information by the developer for use in a future link.

Both qualifiers are only accepted if the /SHAREABLE qualifier is present. /EXPORT_SYMBOL_VECTOR is a command line only qualifier. /PUBLISH_GLOBAL_SYMBOLS can be used in option files as well. The linker will issue a warning if there is an /EXPORT_SYMBOL_VECTOR qualifier but no /PUBLISH_GLOBAL_SYMBOLS qualifier is seen.

/EXPORT_SYMBOL_VECTOR=[file-spec]

/EXPORT_SYMBOL_VECTOR instructs the I64 linker to create an options file with a symbol vector option filled in as directed by the /PUBLISH_GLOBAL_SYMBOLS qualifier, plus a template GSMATCH option.

Using the /EXPORT_SYMBOL_VECTOR qualifier prohibits image production.

If /EXPORT_SYMBOL_VECTOR is given, symbol_vector options as input are not allowed.

At least one /PUBLISH_GLOBAL_SYMBOLS must be on the command line or in an option.

The qualifier also specifies the character string you want the linker to use as the name of the options file that the link operation produces. If you do not specify a file type in the character string, the linker assigns the .OPT file type by default. If you do not specify a file name with the /EXPORT_SYMBOL_VECTOR qualifier, the linker creates an options file with the file name of the first input file.

If you append the /EXPORT_SYMBOL_VECTOR qualifier to an input file specification, the linker creates an options file with the file name of the file to which the qualifier is appended.

After filling in the GSMATCH template option, the generated options file should be used in another link operation to create the shareable image file.

/PUBLISH_GLOBAL_SYMBOLS

The /PUBLISH_GLOBAL_SYMBOLS qualifier instructs the I64 linker to mark an object module or object module library for exporting all its global symbols to a symbol vector options file as requested by the /EXPORT_SYMBOL_VECTOR qualifier.

The qualifier can only be used with the /SHAREABLE and /EXPORT_SYMBOL_VECTOR qualifiers.

The qualifier is compatible with the /INCLUDE and /SELECTIVE_SEARCH qualifiers. The qualifier can be used on the command line and in a linker option.

If the qualifier is applied to an object file, all global symbols from all modules in that file are candidates to be exported as a symbol vector option.

If the qualifier is applied to an object file in combination with the /SELECTIVE_SEARCH qualifier, only referenced global symbols from all modules in that file are candidates to be exported as a symbol vector option.

If the qualifier is applied to an object library, all global symbols from all modules of the object library, which were implicitly included in the image, are candidates to be exported as a symbol vector option.

If the qualifier is applied to an object library in combination with the /INCLUDE qualifier, all global symbols from the modules in the include list of the object library are candidates to be exported as a symbol vector option.

If the qualifier is applied to an object library built with the /SELECTIVE_SEARCH qualifier, only referenced global symbols from all modules of the object library, which were included in the image, are candidates to be exported as a symbol vector option. (See the *HP OpenVMS Command Definition, Librarian, and Message Utilities Manual* for how to insert modules in a library that can be selectively searched.)

If the global symbol is a tentative definition (that is, from the C relaxed refdef extern model) in several modules it is published if one of the modules is marked with /PUBLISH. Similarly, for UNIX weak symbols, any of the modules can make this a candidate to be exported as a symbol vector option. However, for both symbol types, if symbols are overwritten by strong definitions, the presence of /PUBLISH from the defining module determines if the symbols are candidates to be exported as a symbol vector option.

Examples:

```
$ link/SHARE public/PUBLISH,implementation/EXPORT=public
$ link/SHARE plib/LIBRARY/PUBLISH/INCLUDE=public/EXPORT=public
$ LINK/SHAREABLE public/PUBLISH, implementation/EXPORT=public
```

All the global symbols from the modules in PUBLIC.OBJ are exported as a symbol vector option to the file PUBLIC.OPT.

```
$ LINK/SHAREABLE api_table,implementation/PUBLISH/SELECTIVE/EXPORT=public
```

Only the global symbols from the modules in IMPLEMENTATION.OBJ, which are referenced from modules in API_TABLE.OBJ are exported as a symbol vector option to the file PUBLIC.OPT.

```
$ LINK/SHAREABLE api_table,plib/LIBRARY/PUBLISH/EXPORT
```

All the global symbols from the all modules of the library PLIB.OLB, which are included in the shareable image are exported as a symbol vector option to the file PLIB.OPT.

```
$ LINK/SHAREABLE plib/LIBRARY/PUBLISH/INCLUDE=public/EXPORT
```

All the global symbols from the module public of the library PLIB.OLB are exported as a symbol vector option to the file PLIB.OPT.

```
$ LINK/SHAREABLE api_table, plib/LIBRARY/PUBLISH/SELECTIVE/EXPORT=public
```

All the global symbols from all the modules of the library PLIB.OLB, which are referenced from all the modules in api_table, exported as a symbol vector option to the file PUBLIC.OPT. And similar examples with /PUBLISH used in an options file, are in the following:

```
$ link/SHAREABLE TT:/opt/EXPORT  public/PUBLISH, implementation
```

All the global symbols from the modules in PUBLIC.OBJ are exported as a symbol vector option to TT: that is printed to the terminal.

7.1.3.5 New GROUP_SECTIONS and SECTION_DETAILS keywords for the /FULL Qualifier

The OpenVMS I64 Linker takes two keywords to the /FULL qualifier that direct the linker to create a full image map. The format of the /FULL qualifier is:

/FULL [=(keyword [,...])]

The OpenVMS I64 Linker accepts the following keywords to tailor the map (the default is /FULL=SECTION_DETAILS):

Keyword	Meaning
GROUP_SECTIONS	Directs the OpenVMS I64 Linker to list all processed groups (ELF COMDATs).
NOSECTION_DETAILS	Directs the OpenVMS I64 Linker to suppresses zero length contributions in the Program Section Synopsis.
ALL	For the OpenVMS I64 Linker, the ALL keyword is equivalent to specifying both the GROUP_SECTIONS and SECTION_DETAILS keywords.

The first keyword, GROUP_SECTIONS, prints all of the groups that were used in the map. Today the only compiler that takes advantage of groups is C++. Using this keyword with other languages has no effect.

When /FULL=NOSECTION_DETAILS is specified the OpenVMS I64 Linker suppresses zero length contributions in the Program Section Synopsis of the map. When the qualifier /FULL is used, it defaults to /FULL=SECTION_DETAILS, and a full linker map on VAX, Alpha, and I64 systems lists all the module contributions in the Program Section Synopsis.

7.1.4 New Linker Options for OpenVMS I64

Some new linker options have been added to support linking on OpenVMS I64 systems. This section describes these features.

7.1.4.1 New Alignments for the PSECT_ATTRIBUTE Option

The PSECT_ATTRIBUTE option now accepts integers 5, 6, 7, and 8 for the alignment attribute. The integers represent the byte alignment indicated as a power of 2. (For example, 2 ** 6 represents a 64-byte alignment.) The keyword HEXA (for hexadecimal word) was added for 2 ** 5 (that is, a 32-byte or 16-word alignment).

7.1.5 New Ways to Use Existing Linker Qualifiers and Options

The following sections describe new ways to use existing linker qualifiers and options on I64 systems. Topics include:

- Using mixed-case arguments in linker options (Section 7.1.5.1)
- Conventions for specifying image names (Section 7.1.5.2)
- Using the PSECT_ATTRIBUTE option to specify alignment (Section 7.1.5.3)

7.1.5.1 Mixed-Case Arguments in Linker Options on I64 Systems

On OpenVMS I64 systems, names issued by compilers may be mixed-case names. If you need to operate on mixed-case names in the options file (for example you have a library include statement and the module names are mixed-case) the linker already has an option to process the names in mixed-case, rather than using its default behavior (uppercasing all names). That option is the CASE_SENSITIVE option:

```
CASE_SENSITIVE=YES.
```

When the CASE_SENSITIVE option is set to YES, all characters to the right of the left-most equal sign (such as option arguments) have their case preserved. In other words, these characters are taken as-is without modification. This includes file names, module names, symbol names and keywords. To restore the linker's default behavior of upcasing the entire option line, specify the CASE_SENSITIVE option with the NO keyword, as follows:

```
CASE_SENSITIVE=NO
```

Note that the NO keyword must appear in uppercase or it will not be recognized by the linker.

For example, the following module contains mixed-case names that you want to preserve by setting the linker to case-sensitive mode.

```
case=Yes
My_Lib/library/include=(Add_Func, Sub_Func)
symbol_vector=(Add_Func=PROCEDURE,PAGE_COUNT=DATA)
case=NO
```

When processed by the linker, the text appears as follows:

```
CASE=YES
MY_LIB/LIBRARY/INCLUDE=(Add_Func,Sub_Func)
SYMBOL_VECTOR=(Add_Func=PROCEDURE,PAGE_COUNT=DATA)
CASE=NO
```

The case of all names to the right of the first equal sign in each option remains the same.

To maintain VAX and Alpha behavior, HP recommends that you switch to case sensitivity only when needed.

7.1.5.2 Conventions for Specifying Image Names

The following conventions describe the various names that apply to an image:

- Images are given an image file specification (for example, FOO.EXE) that can be changed with the DCL RENAME command.
- The image name is specified with the NAME= option and stored in the image in a note of type NT_VMS_IMGNAM. This name can be different than the image file specification name. However, if you do not use the NAME= option, the name defaults to the image file specification name. The Analyze utility displays this name as the "Image name". You cannot change this name with the DCL RENAME command.
- An additional name for the image is associated with the Global Symbol Table (GST) and stored in the image in a note of type NT_VMS_GSTNAM. The linker sets this name to be the same as the image file specification name. This name is used by the Librarian when you insert an image into an image

library. It is displayed by the Analyze utility as the "Global Symbol Table Name". You cannot change this name with the DCL RENAME command.

On Alpha systems, the image name specified with NAME= is used to identify self-references in the shareable image list. Self-references are calls from within the image to itself, by means of an alias name in the symbol vector.

On I64 systems, there is no entry in the shareable image list for the current image. Self-references are referred to with a special index value into the shareable image list (-1 in the DT_VMS_FIXUP_NEEDED field) that results in a a set of DT_NEEDED entries.

7.1.5.3 Using the PSECT_ATTRIBUTE Option to Specify Alignment

Do not specify a smaller section alignment with the PSECT_ATTRIBUTE than the alignment that the compiler gave to the section.

If you specify a smaller alignment for a program section than any compiler-assigned alignment from all contributions to this section, the linker now issues a warning. For example:

```
$ link hi,sys$input/opt
psect_attr=$literal$,byte
%ILINK-W-CONFALGN, PSECT option alignment (1) less than compiler
assigned (16);
alignment ignored
        section: $LITERAL$
        module: HI
        file: DISK$USER:[JOE]HI.OBJ;3
```

For the VAX and Alpha systems, the linker inappropriately aligns the program section on the boundary that you specified ("byte", in the above code example), and places all the contributions to that program section (from other modules you might have linked with "hi", in the above example) on boundaries that were not specified by the compiler. The linker would not have issued an error message.

The linker always aligns sections on a boundary that is greater than or equal to what was specified by the compiler.

The PSECT_ATTRIBUTE option aligns the section on the specified boundary when it is equal to or greater than that which the compiler specified. It does not align each individual contribution to the section; rather, the total section. The PSECT_ATTRIBUTE option follows the compiler's alignment specification when it aligns each individual contribution.

7.1.5.4 Special Linker Handling of Nonexistent Files

When the RMS_RELATED_CONTEXT linker option is on (the default is RMS_RELATED_CONTEXT=YES) and a nonexistent file is specified in a list of files for the LINK command, the linker's call to LIB$FIND_FILE takes a long time to complete and the linker may appear to hang. Depending on the number of files being linked and the use of logical names in their specification, the linker may take hours to finish because LIB$FIND_FILE locates every combination of the missing file's prefix before displaying a "file not found" message. Note that you cannot terminate the linker process with Ctrl/Y after the linker has called LIB$FIND_FILE.

To quicly find out what the missing file specification is, specify /OPTION in the LINK command. When you press Return, the linker waits for you to enter information into an options file. When you are finished, press Ctrl/Z. To avoid the problems with LIB$FIND_FILE, include the following items in the options file:

- On the first line, specify this option:

  ```
  RMS_RELATED_CONTEXT=NO
  ```

  With the RMS_RELATED_CONTEXT option set to NO, any missing file listed in this options file will generate an immediate "file not found" message.
- On subsequent lines, specify the files to be linked, using full file specifications (in the form *disk*:[*dir*]*filename.ext*) for *every* file. Full file specifications are required because when you specify RMS_RELATED_CONTEXT=NO, file name "stickiness" is disabled.

For example, consider the following LINK command:

```
$ LINK DSK:[TEST]A.OBJ, B.OBJ
```

If you want to specify this command with RMS_RELATED_CONTEXT=NO, you would specify /OPTION and then enter full file specifications for the files to be linked, as follows:

```
$  LINK SYS$INPUT:/OPTION
RMS_RELATED_CONTEXT=NO
DSK:[TEST]A.OBJ, DSK:[TEST]B.OBJ  Ctrl/Z
$
```

Example

The following example shows how the linker appears to hang when file DOES_NOT_EXIST.OBJ is included in the list and the RMS_RELATED_CONTEXT option is not specified (and therefore defaults to YES).

```
$  DEFINE DSKD$ WORK4:[TEST.LINKER.OBJ.]
$  DEFINE RESD$ ROOT$, ROOT2$, ROOT3$, ROOT4$, ROOT5$, DISK_READ$:[SYS.]  ❶
$  DEFINE ROOT$ WORK4:[TEST.PUBLIC.TEST]
$  DEFINE ROOT2$ WORK4:[TEST.LINKER.]
$  DEFINE ROOT3$ WORK4:[TEST.UTIL32.]
$  DEFINE ROOT4$ WORK4:[TEST.PUBLIC.]
$  DEFINE ROOT5$ WORK4:[TEST.PUBLIC.TMP]
$  LINK/MAP/FULL/CROSS/EXE=ALPHA.EXE  RESD$:[TMPOBJ] A.OBJ,-
_$  RESD$:[SRC]B.OBJ,C,DSKD$:[OBJ]D.OBJ,E,RESD$:[TMPSRC]F.OBJ,-
_$  RESD$:[TEST]G.OBJ,RESD$:[SRC.OBJ]H,RESD$:[COM]DOES_NOT_EXIST.OBJ
Ctrl/T NODE6::_FTA183: 15:49:46 LINK CPU=00:02:30.04 PF=5154 IO=254510 MEM=134  ❷
Ctrl/T NODE6::_FTA183: 15:49:46 LINK CPU=00:02:30.05 PF=5154 IO=254513 MEM=134
Ctrl/T NODE6::_FTA183: 15:50:02 LINK CPU=00:02:38.27 PF=5154 IO=268246 MEM=134
Ctrl/T NODE6::_FTA183: 15:50:02 LINK CPU=00:02:38.28 PF=5154 IO=268253 MEM=134
Ctrl/T NODE6::_FTA183: 15:50:14 LINK CPU=00:02:44.70 PF=5154 IO=278883 MEM=134
```

❶ This command defines logical names and equivalents.

❷ Each time you press Ctrl/T, the CPU and IO values increase, but the MEM and PF values do not, indicating that LIB$FIND_FILE is being called.

As shown in the following example, using an options file to set RMS_RELATED_CONTEXT to NO causes the link operation to finish immediately when it encounters the missing file.

```
$  DEFINE DSKD$ WORK4:[TEST.LINKER.OBJ.]
$  DEFINE RESD$ ROOT$, ROOT2$, ROOT3$, ROOT4$, ROOT5$, DISK_READ$:[SYS.]
$  DEFINE ROOT$ WORK4:[TEST.PUBLIC.TEST.]
$  DEFINE ROOT2$ WORK4:[TEST.LINKER.]
$  DEFINE ROOT3$ WORK4:[TEST.UTIL32.]
$  DEFINE ROOT4$ WORK4:[TEST.PUBLIC.]
$  DEFINE ROOT5$ WORK4:[TEST.PUBLIC.TMP.]
$  LINK/MAP/FULL/ CROSS /EXE=ALPHA.EXE SYS$INPUT:/OPTION
RMS_RELATED_CONTEXT=NO
RESD$:[TMPOBJ]A.OBJ,RESD$:[SRC]B.OBJ,RESD$:[SRC]C,DSKD$:[OBJ]D.OBJ
```

```
DSKD$:[OBJ]E,RESD$:[TMPSRC]F.OBJ,RESD$:[TEST]G.OBJ
RESD$:[SRC.OBJ]H,RESD$:[COM]DOES_NOT_EXIST.OBJ Ctrl/Z
```

```
%LINK-F-OPENIN, error opening DISK_READ$:[SYS.][COM]DOES_NOT_EXIST.OBJ; as input
-RMS-E-FNF, file not found
$
```

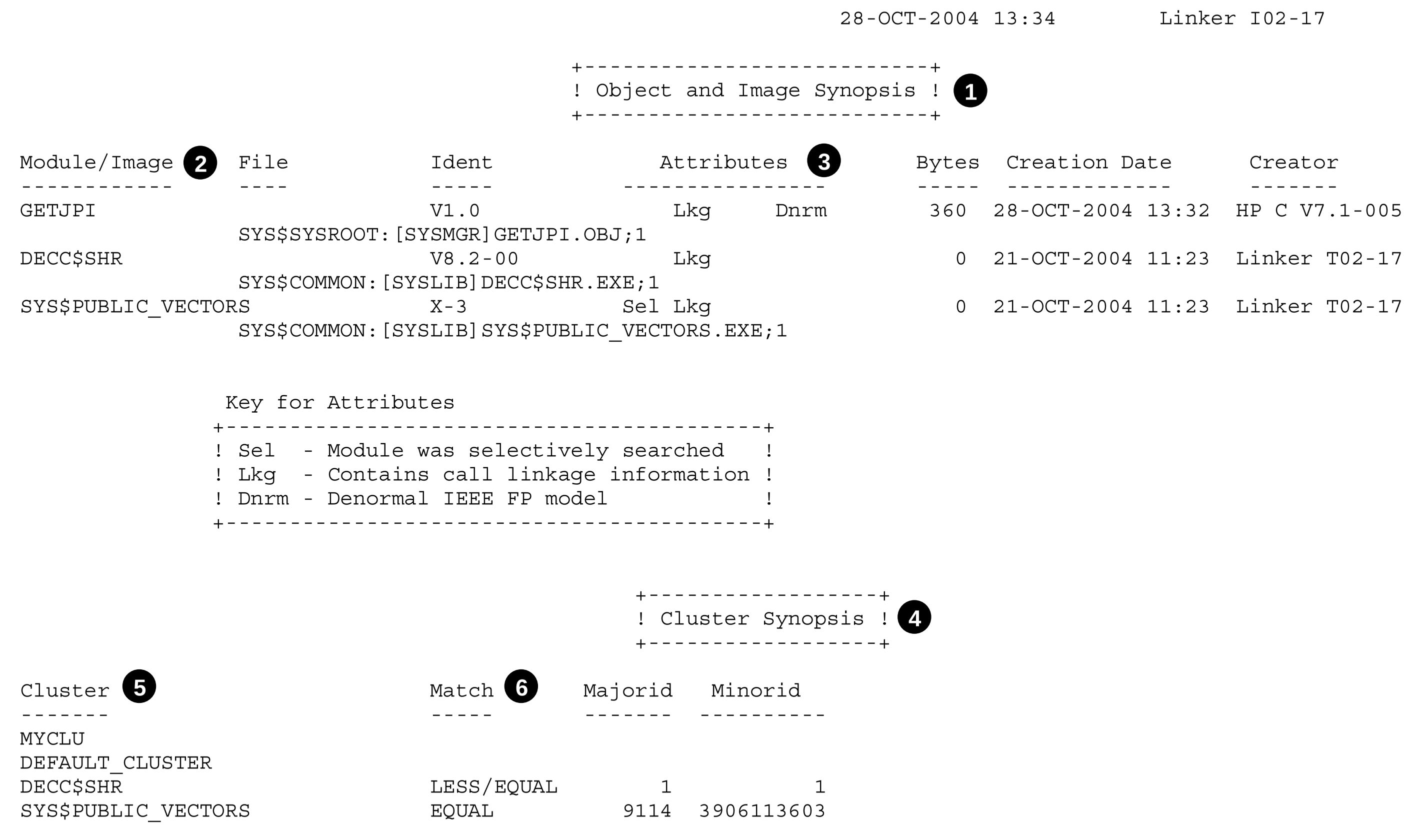
7.1.6 New OpenVMS I64 Linker Map

The Linker map has been enhanced with new information for OpenVMS I64 Linker. The Linker map comprises the following information, shown in Figure 7–1, Figure 7–2, Figure 7–3, Figure 7–4, Figure 7–5, Figure 7–6, Figure 7–7, and Figure 7–8.

- Object and image synopsis
- Cluster synopsis
- Image segment synopsis
- Program section synopsis
- Symbol cross reference
- Symbols by value
- Image synopsis
- Link run statistics

Callout descriptions describing new and changes portions of the Linker map are provided after the example Linker map.

Figure 7–1 Object and Image Synopsis and Cluster Synopsis



```
                                                       28-OCT-2004 13:34        Linker I02-17

                                         +---------------------------+
                                         ! Object and Image Synopsis ! 1
                                         +---------------------------+

Module/Image 2  File            Ident             Attributes 3      Bytes  Creation Date      Creator
------------    ----            -----          ----------------     -----  -------------      -------
GETJPI                          V1.0              Lkg     Dnrm        360  28-OCT-2004 13:32  HP C V7.1-005
                SYS$SYSROOT:[SYSMGR]GETJPI.OBJ;1
DECC$SHR                        V8.2-00           Lkg                   0  21-OCT-2004 11:23  Linker T02-17
                SYS$COMMON:[SYSLIB]DECC$SHR.EXE;1
SYS$PUBLIC_VECTORS              X-3            Sel Lkg                  0  21-OCT-2004 11:23  Linker T02-17
                SYS$COMMON:[SYSLIB]SYS$PUBLIC_VECTORS.EXE;1


                Key for Attributes
               +-------------------------------------------+
               ! Sel  - Module was selectively searched    !
               ! Lkg  - Contains call linkage information  !
               ! Dnrm - Denormal IEEE FP model             !
               +-------------------------------------------+


                                              +------------------+
                                              ! Cluster Synopsis ! 4
                                              +------------------+

Cluster 5                       Match 6     Majorid   Minorid
-------                         -----       -------  ----------
MYCLU
DEFAULT_CLUSTER
DECC$SHR                        LESS/EQUAL        1           1
SYS$PUBLIC_VECTORS              EQUAL          9114  3906113603
```

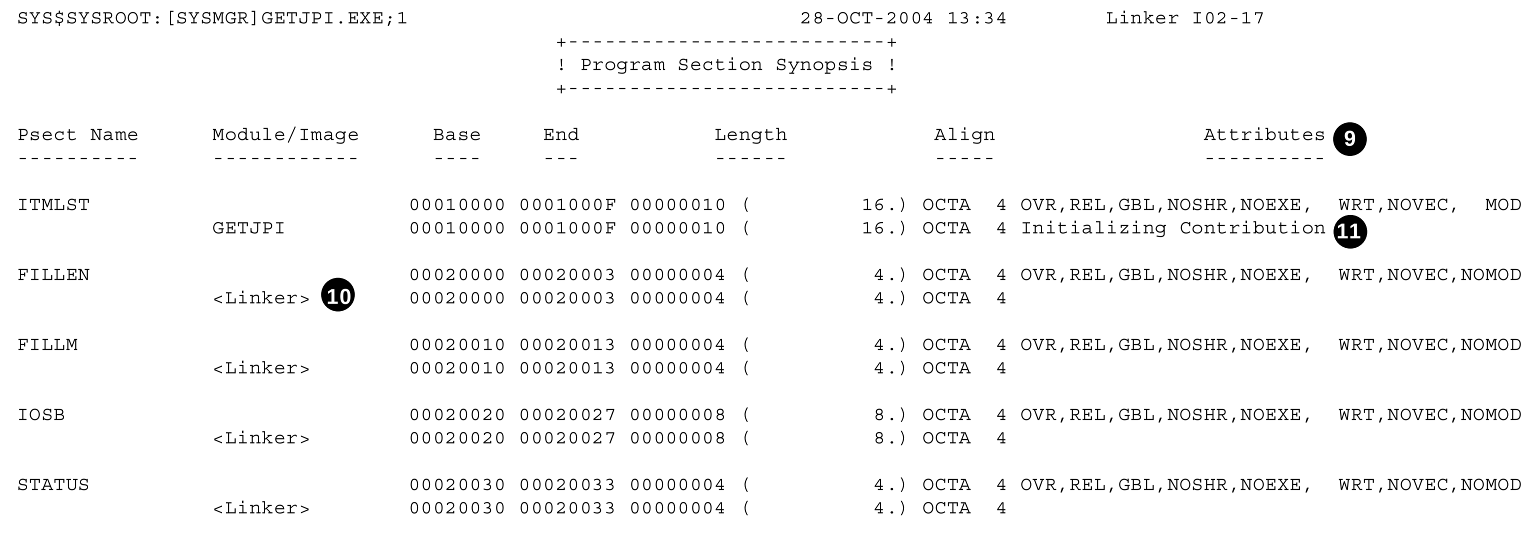
VM-1173A-AI

Figure 7–2 Image Segment Synopsis

```
SYS$SYSROOT:[SYSMGR]GETJPI.EXE;1                              28-OCT-2004 13:34       Linker I02-17

                                             +------------------------+
                                             ! Image Segment Synopsis ! ❼
                                             +------------------------+

Seg#  Cluster               Type      Pglts   Base Addr  Disk VBN  PFC  Protection  Attributes
----  -------               ----      -----   ---------  --------  ---  ----------  ----------
   0  MYCLU                 LOAD          1    00010000         2    0  READ WRITE
   1                        LOAD          1    00020000         0    0  READ WRITE  DEMAND ZERO
   2                        LOAD          1    00030000         3    0  READ ONLY   EXECUTABLE,SHARED
   3                        LOAD          1    00040000         4    0  READ ONLY   SHARED
   4                        LOAD          1    00050000         5    0  READ ONLY   [UNWIND]
❽ 5  DEFAULT_CLUSTER       LOAD          1    00060000         6    0  READ ONLY   SHORT
   6                        DYNAMIC       2  Q-00000000
                                               80000000         7    0  READ ONLY


              Key for special characters above
               +----------------------+
               !   Q  - Quadword      !
               +----------------------+
```

VM-1174A-AI

Figure 7–3 Program Section Synopsis



```
SYS$SYSROOT:[SYSMGR]GETJPI.EXE;1                               28-OCT-2004 13:34       Linker I02-17
                                             +--------------------------+
                                             ! Program Section Synopsis !
                                             +--------------------------+

Psect Name      Module/Image      Base     End           Length           Align                 Attributes 9
----------      ------------      ----     ---           ------           -----                 ----------

ITMLST                          00010000 0001000F 00000010 (         16.) OCTA  4 OVR,REL,GBL,NOSHR,NOEXE,  WRT,NOVEC,  MOD
                GETJPI          00010000 0001000F 00000010 (         16.) OCTA  4 Initializing Contribution 11

FILLEN                          00020000 00020003 00000004 (          4.) OCTA  4 OVR,REL,GBL,NOSHR,NOEXE,  WRT,NOVEC,NOMOD
                <Linker> 10     00020000 00020003 00000004 (          4.) OCTA  4

FILLM                           00020010 00020013 00000004 (          4.) OCTA  4 OVR,REL,GBL,NOSHR,NOEXE,  WRT,NOVEC,NOMOD
                <Linker>        00020010 00020013 00000004 (          4.) OCTA  4

IOSB                            00020020 00020027 00000008 (          8.) OCTA  4 OVR,REL,GBL,NOSHR,NOEXE,  WRT,NOVEC,NOMOD
                <Linker>        00020020 00020027 00000008 (          8.) OCTA  4

STATUS                          00020030 00020033 00000004 (          4.) OCTA  4 OVR,REL,GBL,NOSHR,NOEXE,  WRT,NOVEC,NOMOD
                <Linker>        00020030 00020033 00000004 (          4.) OCTA  4
```

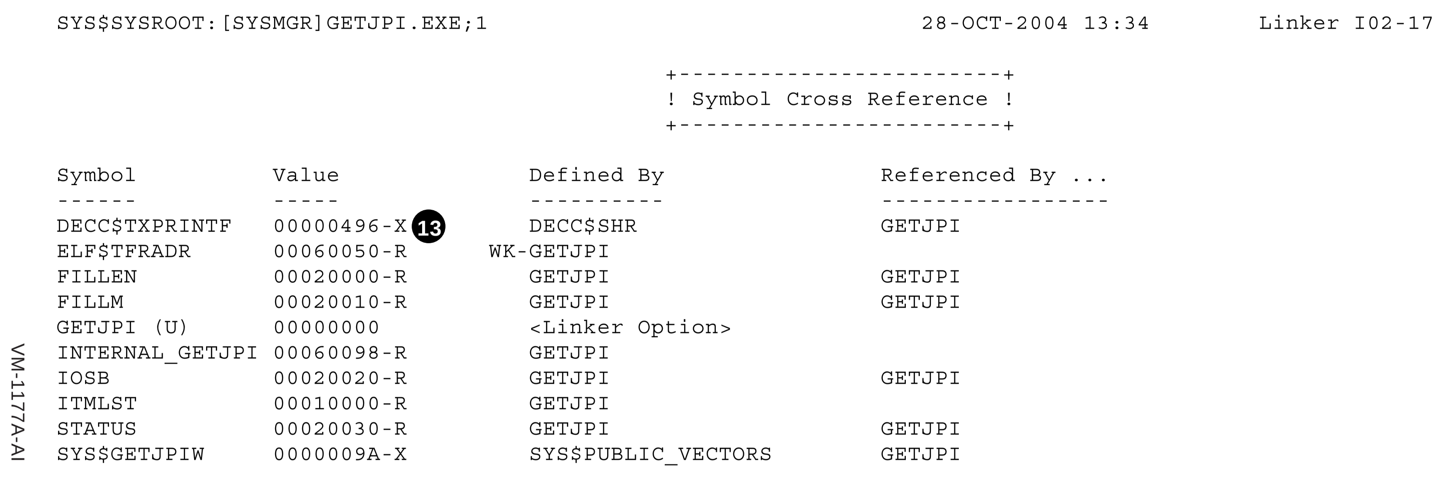
VM-1175A-AI

Figure 7–4 Program Section Synopsis (Continued)

```
$CODE$                                00030000 0003015F 00000160 (    352.) OCTA  4 CON,REL,LCL,  SHR,  EXE,NOWRT,NOVEC,  MOD
                GETJPI                00030000 000300FF 00000100 (    256.) OCTA  4
                <Linker> 12           00030100 0003015F 00000060 (    96.) OCTA  4

$LINK$                                00040000 00040000 00000000 (      0.) OCTA  4 CON,REL,LCL,NOSHR,NOEXE,NOWRT,NOVEC,NOMOD
                GETJPI                00040000 00040000 00000000 (      0.) OCTA  4

$LITERAL$                             00040000 00040017 00000018 (     24.) OCTA  4 CON,REL,LCL,  SHR,NOEXE,NOWRT,NOVEC,  MOD
                GETJPI                00040000 00040017 00000018 (     24.) OCTA  4

$READONLY$                            00040020 0004002F 00000010 (     16.) OCTA  4 CON,REL,LCL,  SHR,NOEXE,NOWRT,NOVEC,  MOD
                GETJPI                00040020 0004002F 00000010 (     16.) OCTA  4

$LINKER UNWIND$                       00050000 00050017 00000018 (     24.) QUAD  3 CON,REL,LCL,NOSHR,NOEXE,NOWRT,NOVEC,  MOD
                GETJPI                00050000 00050017 00000018 (     24.) QUAD  3

$LINKER UNWINFO$                      00050018 0005002F 00000018 (     24.) QUAD  3 CON,REL,LCL,NOSHR,NOEXE,NOWRT,NOVEC,  MOD
                GETJPI                00050018 0005002F 00000018 (     24.) QUAD  3

$LINKER SYMBOL_VECTOR$                00060000 00060007 00000008 (      8.) OCTA  4 CON,REL,GBL,NOSHR,NOEXE,NOWRT,NOVEC,  MOD,SHORT
                <Linker Option>       00060000 00060007 00000008 (      8.) OCTA  4

$LINKER SDATA$                        00060008 000600AF 000000A8 (     168.) OCTA  4 CON,REL,GBL,NOSHR,NOEXE,NOWRT,NOVEC,  MOD,SHORT
                <Linker>              00060008 000600AF 000000A8 (     168.) OCTA  4
```

VM-1176A-AI

Figure 7–5 Symbol Cross Reference



```
SYS$SYSROOT:[SYSMGR]GETJPI.EXE;1                              28-OCT-2004 13:34       Linker I02-17

                                                         +------------------------+
                                                         ! Symbol Cross Reference !
                                                         +------------------------+

Symbol          Value              Defined By                Referenced By ...
------          -----              ----------                -----------------
DECC$TXPRINTF   00000496-X 13      DECC$SHR                  GETJPI
ELF$TFRADR      00060050-R      WK-GETJPI
FILLEN          00020000-R         GETJPI                    GETJPI
FILLM           00020010-R         GETJPI                    GETJPI
GETJPI (U)      00000000           <Linker Option>
INTERNAL_GETJPI 00060098-R         GETJPI
IOSB            00020020-R         GETJPI                    GETJPI
ITMLST          00010000-R         GETJPI
STATUS          00020030-R         GETJPI                    GETJPI
SYS$GETJPIW     0000009A-X         SYS$PUBLIC_VECTORS        GETJPI
```

VM-1177A-AI

Figure 7–6 Symbols by Value

```
SYS$SYSROOT:[SYSMGR]GETJPI.EXE;1                    28-OCT-2004 13:34      Linker I02-17

                                         +-------------------+
                                         ! Symbols By Value !
                                         +-------------------+

Value          Symbols...
-----          ----------
00000000         GETJPI (U)
0000009A       X-SYS$GETJPIW
00000496       X-DECC$TXPRINTF
00010000       R-ITMLST
00020000       R-FILLEN
00020010       R-FILLM
00020020       R-IOSB
00020030       R-STATUS
00060050       R-ELF$TFRADR
00060098       R-INTERNAL_GETJPI

               Key for special characters above (14)
               +-----------------------+
               !   *  - Undefined      !
               !  (U) - Universal      !
               !   R  - Relocatable    !
               !   X  - External       !
               !   C  - Code Address   !
               !  WK  - Weak           !
               ! UxWk - Unix-Weak      !
               +-----------------------+
```

VM-1178A-AI

Figure 7–7 Image Synopsis

```
SYS$SYSROOT:[SYSMGR]GETJPI.EXE;1                                28-OCT-2004 13:34        Linker I02-17

                                                 +----------------+
                                                 ! Image Synopsis !
                                                 +----------------+

Virtual memory allocated:                         00010000 0006FFFF 00060000 (393216. bytes, 768. pages)
64-Bit Virtual memory allocated:                  00000000 00000000 00000000
                                                  80000000 80010000 00010000 (65536. bytes, 128. pages)
Stack size:                                              0. pages
Image header virtual block limits:                       1.        1. (    1. block)
Image binary virtual block limits:                       2.        8. (    7. blocks)
Image name and identification:                    GETJPI V1.0
Number of files:                                         5.
Number of modules:                                       3.
Number of program sections:                              9.
Number of global symbols:                             3338.
Number of cross references:                             17.
Number of image segments:                                7.
Transfer address from module:                       GETJPI
User transfer FD address:                1000000000 00060050
User transfer code address:              00000000 00030000
Initial FP mode:                         00000000 09800000 (IEEE DENORM_RESULTS)
Number of code references to shareable images:        3332.
Image type:                                       SHAREABLE. Global Section Match=EQUAL, Ident, Major=9120, Minor=2068313277
Reduced Floating Point model (RFP):               Image does not use RFP model
Map format:                                       FULL WITH CROSS REFERENCE in file SYS$SYSROOT:[SYSMGR]GETJPI.MAP;1
Estimated map length:                             441. blocks
```

VM-1179A-AI

Figure 7–8 Link Run Statistics

```
                                                  +---------------------+
                                                  ! Link Run Statistics !
                                                  +---------------------+

Performance Indicators                                Page Faults      CPU Time     Elapsed Time
----------------------                                -----------      --------     ------------
    Command processing:                                        55   00:00:00.00    00:00:00.00
    Pass 1:                                                   173   00:00:00.06    00:00:00.05
    Allocation/Relocation:                                      5   00:00:00.02    00:00:00.02
    Pass 2:                                                    32   00:00:00.01    00:00:00.00
    Symbol table output:                                        4   00:00:00.00    00:00:00.00
    Map data after object module synopsis:                      5   00:00:00.00    00:00:00.07
Total run values:                                             274   00:00:00.09    00:00:00.17

Quota usage 15                                         ByteCount  FileCount  PgFlCount
------------                                           ---------  ---------  ---------
    Available:                                            127808        100     512000
    Command processing:                                      384          2       7888
    Pass 1:                                                  384          2      10240
    Allocation/Relocation:                                   576          3      10240
    Pass 2:                                                  576          3      18624
    Symbol table output:                                     384          2      18624
    Map data after object module synopsis:                   384          2      18624

Using a working set limited to 16384 pages and 11029 pages of data storage (excluding image)

Number of modules extracted explicitly             = 0
    with 0 extracted to resolve undefined symbols



1 library searches were for symbols not in the library searched

A total of 8 global symbol table entries was written

LINK/DEB/MAP/FULL/CROSS/SHARE GETJPI.OPT/OPT
<SYS$SYSROOT:[SYSMGR]GETJPI.OPT;2>
cluster=myclu,,,getjpi.obj
symbol_vector=(getjpi/internal_getjpi=procedure
```

VM-1180A-AI

The following list correspond to the numbered items in the preceding figures:

❶ Object and Image Synopsis. The Alpha section titled Object Module Synopsis has been renamed on I64 to Object and Image Synopsis.

❷ Module/Image. The Alpha column Module Name has been renamed on I64 to Module/Image.

❸ Attributes. New information has been added in the four columns titled Attributes. The first of the four columns indicates whether the search of the module is selective. If selective, "Sel" appears. If it is not selective, this column is blank.

The second column indicates whether the module has call linkage information. If the module has linkage, "Lkg" appears. If the module does not have linkage, this column is blank.

The third column indicates whether the module is compiled with the Reduced Floating Point model. If it is, "RFP" appears. If the module was not compiled with the Reduced Floating Point model, this column is blank. This designation is suppressed for shareable images.

The fourth column indicates the whole program mode. Several abbreviations can appear in this column that are listed in the Key for Attributes section. The following example lists all of the possible abbreviations in the Keys for Attributes section. The Creation Date and Creator columns are truncated from this example; see the following map example for the the entire Object and Image Synopsis.

```
Module/Image    File           Ident           Attributes       Bytes
------------    ----           -----        ----------------    -----
NONE                            V1.0             Lkg               568
                DISK1:[JOE]NONE.OBJ;1
DNORM_CASE                                       Lkg RFP Dnrm      504
                DISK1:[JOE]DENORM_W.OBJ;1
FAST _CASE                                       Lkg RFP Fast      504
                DISK1:[JOE]FAST_W.OBJ;1
NEPCT_CASE                                       Lkg RFP Inex      504
                DISK1:[JOE]INEXACT_W.OBJ;1
SPCL _CASE                                       Lkg RFP Spcl      504
                DISK1:[JOE]SPECIAL_W.OBJ;1
UNDER_CASE                                       Lkg RFP Undr      504
                DISK1:[JOE]UNDERFLOW_W.OBJ;1
DG_FL_CASE                                       Lkg RFP VXfl      504
                DISK1:[JOE]VAXFLOAT_W.OBJ;1
DECC$SHR                        V8.2-00          Lkg                 0
                RESD$:[SYSLIB]DECC$SHR.EXE;1
SYS$PUBLIC_VECTORS              X-2          Sel Lkg                 0
                RESD$:[SYSLIB]SYS$PUBLIC_VECTORS.EXE;1

                 Key for Attributes
                +-------------------------------------------+
                ! Sel  - Module was selectively searched    !
                ! Lkg  - Contains call linkage information  !
                ! RFP  - Conforms to the reduced FP model   !
                ! VXfl - VAX Float FP model                 !
                ! Dnrm - Denormal IEEE FP model             !
                ! Fast - Fast IEEE FP model                 !
                ! Inex - Inexact IEEE FP model              !
                ! Undr - Underflow-to-zero IEEE FP model    !
                ! Spcl - Special FP model                   !
                +-------------------------------------------+
```

❹ Cluster Synopsis. The Alpha section titled Image Section Synopsis has been divided into two sections for I64: Cluster Synopsis and Image Segment Synopsis. The Cluster Synopsis section no longer contains image sections, which on I64 are referred to as *segments*. Further, image sections are no longer printed for sharable images, which was formerly done on VAX map files whenever there was a possibility of having based shareable images. (Based shareable images are not allowed on Alpha or I64 systems.)

❺ Cluster. The Cluster column shows clusters that were created for and used by the Linker, and the order in which they were processed.

❻ Match, Majorid, and Minorid. The Match, Majorid, and Minorid columns show version criteria, if any.

❼ Image Segment Synopsis. The Image Segment Synopsis section shows each image segment as it was created. It contains the remaining columns of the Image Section Synopsis on OpenVMS Alpha. The first column, Seg #, contains the image segment's number, which is used in the relocations that are applied to it. (See an analysis of an image for a display of the segment number in relocations.) The Alpha section Protection and Paging has been divided into two columns Protection and Attributes on I64. The column Global Section Name was eliminated.

❽ If the module was compiled /TIE and the image is linked /NONATIVE only and if the image contains non-standard signatures, a separate segment will appear immediately after the short data segment (indicated by SHORT) that contains them.

❾ The section attributes PIC, NOPIC are not valid attributes on I64 and were removed.

❿ The Linker contributes storage for common or relaxed refdef symbols. It is marked with <Linker> under the Module/Image header. The section name is always named after the symbol. (This module was compiled with the default switch /EXTERN=RELAXED, and the variables ITMLST, FILLEN, FILLIM and IOSB are relaxed refdef symbols).

⓫ The Linker indicates which module made the initialization (if there was one) to sections which have the attributes OVR, REL and GBL with the designation "Initializing Contribution". If you get a multiple initializations error, the Linker will have two or more sections marked with the designation "Initializing Contribution", in order to help you debug an instance that has many contributors.

⓬ The Linker makes a contribution to the code segment containing trampolines (instructions with larger branches) or code to branch to another segment (either inside or outside the image). It is marked with <Linker> under the Module/Image header.

⓭ The designation of an external symbol has changed from Alpha maps. The prefix or suffix used on Alpha was RX, meaning relocatable and external. However, the Linker does not know whether an external symbol is relocatable or not. As a result, on I64 systems the prefix or suffix has been changed to X (external).

⓮ Keys for Special Characters. The keys for special characters have changed as follows:

- On I64, the special character C appears for code address. When a function does not have a function descriptor assigned by the Linker, its value is its code address.
- On OpenVMS Alpha, a universal symbol appeared once with its internal value. The map never displayed an external, universal value (its index into the symbol vector on I64). Universal symbols on I64 now appear once with a suffix of (U) defined by <Linker Option> to indicate the external value, and again, possibly with the prefix or suffix R, to indicate their internal value. If you had a symbol vector with an alias name the alias name appears with the universal value, and the internal name appears with the internal value. The prefixes and suffixes A and I (for Alias and Internal) on OpenVMS Alpha have been removed.

  For example, symbol_vector=(getjpi/internal_getjpi=procedure) yields:

  ```
  00000000        GETJPI (U)
  00050098      R-INTERNAL_GETJPI
  ```
- UNIX weak symbols, designated with UxWk, are a new addition to OpenVMS. They are similar to OpenVMS weak symbols; however, more than one symbol with a UNIX weak definition can be processed when linking multiple modules without producing a multiple definitions error. UNIX weak symbols are currently produced by the C++ compiler.

⓯ Quota Usage. A new section titled Quota Usage was added to the Link Run Statistics section to keep track of the quotas that are being used by the I64 Linker. If quota issues occur, the Linker is able to work around them. But the Linker outputs a special message to the Quota Usage section indicating, as best it can, what quota should be increased to improve performance. For example:

```
Performance of this link operation could be improved by increasing quotas
   Quota related to status return:  %SYSTEM-SECTBLFUL, process or global
   section table is full
2688 extra file I/O operations performed due to current process quota(s)
36 performed on object files; 2652 performed on library files
```

Part II

OpenVMS Documentation

8 OpenVMS Documentation Overview

Table 8–1 describes any reorganization of the OpenVMS documentation set for OpenVMS Version 8.2. For this release, two new manuals are added to the OpenVMS Documentation Set, and three manuals have been archived. Also, the following manuals are again in hardcopy format:

- *HP OpenVMS Management Station Overview and Release Notes*
- *COM, Registry, and Events for HP OpenVMS Developer's Guide*

Table 8–1 Documentation Set Changes for OpenVMS Version 8.2

New Manuals	
Porting Applications from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers	This new manual for application programmers who are planning to migrate OpenVMS Alpha applications to OpenVMS I64 applications.
HP MACRO Compiler Porting and User's Guide	This is a revised manual making its first appearance in the OpenVMS Documentation Set.
Archived Manuals	
VAX MACRO and Instruction Set Reference Manual	This manual describes the features of the VAX MACRO instruction set and assembler. Beginning Version 8.2, this manual is archived. Look for the link to archived manuals on the OpenVMS documentation Web site: `http://www.hp.com/go/openvms/doc/`
OpenVMS VAX RTL Mathematics (MTH$) Manual	This manual documents the mathematics routine contained in the MTH$ facility of the OpenVMS VAX Run-Time Library. Beginning Version 8.2, this manual is archived. Look for the link to archived manuals on the OpenVMS documentation Web site: `http://www.hp.com/go/openvms/doc/`
OpenVMS VAX System Dump Analyzer Utility Manual	This manual explains how to use the System Dump Analyzer (SDA) to investigate system failures and examine a running system. Beginning Version 8.2, this manual is archived. Look for the link to archived manuals on the OpenVMS documentation Web site: `http://www.hp.com/go/openvms/doc/`

9
OpenVMS Printed and Online Documentation

OpenVMS documentation is provided, in the following ways:

- Printed documentation

  If you need paper documents, you can purchase most OpenVMS manuals in the form of printed documentation sets. Individual OpenVMS hardcopy documents cannot be purchased separately but are available in kits. One exception is the *Porting Applications from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers*, which you can order in hardcopy.

- Online documentation on CD

  All OpenVMS manuals are available in online formats on CDs that also include the documentation for many associated products. You automatically receive the documentation CDs in your OpenVMS media kit.

- Online documentation on the OpenVMS documentation Web site

  You can preview or read any OpenVMS document, including archived manuals, on the OpenVMS Web site.

- Online help

  You can quickly display online help for OpenVMS commands, utilities, and system routines when you need task-related information.

The following sections describe each format in which OpenVMS documentation is provided and specifies the titles that are available in that format.

9.1 Printed Documentation

Some printed documentation comes with your OpenVMS Media Kit. All other printed manuals are orderable in kits. This section describes the OpenVMS printed documentation offerings, which are categorized as follows:

- Media kit
- Documentation sets:
  - Base
  - Full
  - Operating Environment Extensions
- System-integrated products
- Archived manuals

9.1.1 OpenVMS Media Kit Documentation

The OpenVMS Media Kit, for both OpenVMS Alpha and OpenVMS I64 systems, contains the documents you need to get started with the latest version of the OpenVMS operating system. Table 9–1 lists the books included in the OpenVMS media kit. The books you receive are determined by whether you are a new or a service customer. New customers receive all books; service customers receive only new books and books that have been updated since the last release.

Note

The *HP OpenVMS License Management Utility Manual*, *Guide to HP OpenVMS Version 8.2 Media*, and *HP OpenVMS Version 8.2 Upgrade and Installation Manual* are provided only in the OpenVMS Media kit and, therefore, are not part of the OpenVMS Full Documentation set (described in Section 9.1.2).

Table 9–1 OpenVMS Media Kit Manuals

Manual	Order Number
HP OpenVMS License Management Utility Manual	AA-PVXUG-TK
Guide to HP OpenVMS Version 8.2 Media	BA322-90001
HP OpenVMS Version 8.2 New Features and Documentation Overview	BA322-90003
HP OpenVMS Version 8.2 Upgrade and Installation Manual	BA322-90002
HP OpenVMS Version 8.2 Release Notes	BA322-90004

9.1.2 OpenVMS Documentation Sets

OpenVMS documentation is available in the following documentation sets:

Documentation Set	Description	Alpha Order Number	I64 Order Number
Full Set	Intended for users who need extensive explanatory information for all major OpenVMS resources. Contains all the OpenVMS documentation in one offering. Includes the Base Documentation set.	QA-001AA-GZ.8.2	BA554MN
Base Set	Subset of the Full Documentation set. Intended for general users and system managers of small standalone systems. Includes the most commonly used OpenVMS manuals.	QA-09SAA-GZ.8.2	BA555MN

There is one common documentation set for both OpenVMS Alpha and OpenVMS I64 systems. OpenVMS Alpha documentation set and the OpenVMS I64 documentation set contain the identical books with one exception. The OpenVMS Alpha documentation set contains the *COM, Registry, and Events for HP OpenVMS Developer's Guide*, which is an Alpha-only document. Table 9–2 lists the manuals in the OpenVMS Base and Full Documentation sets. For a description of each manual, see Section 10.2.

Table 9–2 OpenVMS Full Documentation Set (QA-001AA-GZ.8.2/BA554MN)

Manual	Order Number
OpenVMS Base Documentation Set	**QA-09SAA-GZ.8.2/BA555MN**
HP OpenVMS DCL Dictionary: A–M[1]	AA-PV5KK-TK
HP OpenVMS DCL Dictionary: N–Z[1]	AA-PV5LK-TK
HP OpenVMS Guide to System Security	AA-Q2HLG-TE
HP OpenVMS System Management Utilities Reference Manual: A–L[1]	AA-PV5PJ-TK
HP OpenVMS System Management Utilities Reference Manual: M–Z[1]	AA-PV5QJ-TK
HP OpenVMS System Manager's Manual, Volume 1: Essentials[1]	AA-PV5MJ-TK
HP OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems[1]	AA-PV5NJ-TK
OpenVMS User's Manual	AA-PV5JG-TK
HP OpenVMS Version 8.2 New Features and Documentation Overview[2]	BA322-90003
HP OpenVMS Version 8.2 Release Notes[2]	BA322-90004
Additional Books in the Full Documentation Set	**QA-001AA-GZ.8.2**
HP OpenVMS Availability Manager User's Guide[1]	AA-RNSJD-TE
COM, Registry, and Events for HP OpenVMS Developer's Guide[1],[3]	AA-RSCWC-TE
HP C Run-Time Library Reference Manual for OpenVMS Systems[1]	AA-RSMUC-TE
Compaq C Run-Time Library Utilities Reference Manual	AA-R238C-TE
Compaq Portable Mathematics Library	AA-PV6VE-TE
DECamds User's Guide	AA-Q3JSE-TE
DEC Text Processing Utility Reference Manual	AA-PWCCD-TE
Extensible Versatile Editor Reference Manual	AA-PWCDD-TE
Guidelines for OpenVMS Cluster Configurations[1]	AA-Q28LH-TK
Guide to Creating OpenVMS Modular Procedures	AA-PV6AD-TK
Guide to OpenVMS File Applications	AA-PV6PE-TK
Guide to the POSIX Threads Library	AA-QSBPD-TE
Guide to the DEC Text Processing Utility	AA-PWCBD-TE
HP Open Source Security for OpenVMS, Volume 1: Common Data Security Architecture	AA-RSCUB-TE
HP Open Source Security for OpenVMS, Volume 2: HP SSL for OpenVMS[1]	AA-RSCVC-TE
HP Open Source Security for OpenVMS, Volume 3: Kerberos[1]	AA-RUEBB-TE
HP OpenVMS Alpha Partitioning and Galaxy Guide	AA-REZQD-TE
HP OpenVMS Guide to Upgrading Privileged-Code Applications[1]	AA-QSBGE-TE

[1]Revised for Version 8.2.
[2]New for Version 8.2.
[3]Alpha only - Provided only in QA-001AA-GZ.8.2

(continued on next page)

Table 9–2 (Cont.) OpenVMS Full Documentation Set (QA-001AA-GZ.8.2/BA554MN)

Manual	Order Number
Additional Books in the Full Documentation Set	**QA-001AA-GZ.8.2**
HP OpenVMS System Analysis Tools Manual[1]	AA-REZTE-TE
HP OpenVMS Calling Standard[1]	AA-QSBBE-TE
HP OpenVMS Cluster Systems	AA-PV5WF-TK
HP OpenVMS Command Definition, Librarian, and Message Utilities Manual[1]	AA-QSBDE-TE
HP OpenVMS Debugger Manual[1]	AA-QSBJE-TE
HP OpenVMS Delta/XDelta Debugger Manual[1]	AA-PWCADE-TE
HP OpenVMS I/O User's Reference Manual[1]	AA-PV6SG-TK
HP OpenVMS Linker Utility Manual	AA-PV6CDE-TK
HP OpenVMS MACRO Compiler Porting and User's Guide[1]	AA-PV64DE-TE
HP OpenVMS Management Station Overview and Release Notes[1]	AA-QJGCG-TE
OpenVMS Performance Management	AA-R237C-TE
Porting Applications from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers[2]	BA442-90001
HP OpenVMS Programming Concepts Manual, Volume I[1]	AA-RNSHD-TK
HP OpenVMS Programming Concepts Manual, Volume II[1]	AA-PV67H-TK
OpenVMS Record Management Services Reference Manual	AA-PV6RE-TK
OpenVMS Record Management Utilities Reference Manual	AA-PV6QD-TK
HP OpenVMS RTL General Purpose (OTS$) Manual[1]	AA-PV6HE-TK
HP OpenVMS RTL Library (LIB$) Manual[1]	AA-QSBHE-TE
OpenVMS RTL Screen Management (SMG$) Manual	AA-PV6LD-TK
OpenVMS RTL String Manipulation (STR$) Manual	AA-PV6MD-TK
OpenVMS System Messages: Companion Guide for Help Message Users	AA-PV5TD-TK
HP OpenVMS System Services Reference Manual: A–GETUAI[1]	AA-QSBMG-TE
HP OpenVMS System Services Reference Manual: GETUTC–Z[1]	AA-QSBNG-TE
OpenVMS Utility Routines Manual	AA-PV6EF-TK
OpenVMS VAX RTL Mathematics (MTH$) Manual	AA-PVXJD-TE
OpenVMS VAX System Dump Analyzer Utility Manual	AA-PV6TD-TE
POLYCENTER Software Installation Utility Developer's Guide[1]	AA-Q28MF-TK
VAX MACRO and Instruction Set Reference Manual	AA-PS6GD-TE
HP Volume Shadowing for OpenVMS	AA-PVXMK-TE

[1]Revised for Version 8.2.
[2]New for Version 8.2.

9.1.3 Operating Environments Extensions Documentation Set (I64 Only)

The Operating Environments Extensions Documentation Set includes manuals that support the products that are included in the OEs. See Section 10.5 for a list of these documents.

9.1.4 Documentation for System Integrated Products

System Integrated Products (SIPs) are included in the OpenVMS software, but you must purchase separate licenses to enable them. Table 9–3 shows the documentation associated with System Integrated Products.

Table 9–3 System Integrated Products Documentation

System Integrated Product	Related Documentation
HP Galaxy Software Architecture on OpenVMS Alpha	The documentation is included in the OpenVMS Full Documentation Set.
OpenVMS Clusters	The OpenVMS Cluster documentation is included in the OpenVMS Full Documentation Set.
RMS Journaling for OpenVMS	RMS Journaling for OpenVMS manual is provided in HTML format on the OpenVMS Documentation Web site: `http://www.hp.com/go/openvms/doc`
Volume Shadowing for OpenVMS	The documentation is included in the OpenVMS Full Documentation Set.

9.1.5 Archived OpenVMS Documentation

OpenVMS continuously updates, revises, and enhances the OpenVMS operating system documentation. From time to time, manuals are archived. You can access the archived manuals online from the following Web site:

`http://www.hp.com/go/openvms/doc`

For a list of the archived OpenVMS manuals, see Section 10.6.

9.2 Authoring Tool for OpenVMS Documentation

OpenVMS Documentation team is continuing to introduce books that have been authored and published using a tool based on the Standard Generalized Markup Language (SGML). SGML is an industry standard and will provide many benefits to both the customer and OpenVMS documentation.

Readers will notice a difference in appearance between books produced from SGML and others in the documentation set. This is true for HTML, PDF, and printed formats and is a natural result of the new authoring environment.

The following Version 8.2 books have been produced with this new tool:

- *HP Open Source Security for OpenVMS, Volume 2: HP SSL for OpenVMS*
- *HP Open Source Security for OpenVMS, Volume 3: Kerberos*
- *HP OpenVMS Version 8.2 Upgrade and Installation Manual*
- *Guide to HP OpenVMS Version 8.2 Media*
- *HP Open Source Security for OpenVMS, Volume 3: Kerberos*
- *HP OpenVMS I/O User's Reference Manual*
- *HP OpenVMS System Manager's Manual, Volume 1: Essentials*
- *HP OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems*

9.3 Online Documentation on CD

Online documentation for the OpenVMS operating system and many associated products is provided on two CDs. One is an ISO 9660 Level 2 CD that is readable on Windows® and Macintosh® systems. The other is a Files-11 CD that is readable on Windows and OpenVMS systems. The contents of the CDs are the same except for the following:

- The ISO 9660 Level 2 CD contains Adobe® Acrobat® Reader Version 5.0.5.
- The Files-11 CD contains the HP Secure Web Browser for OpenVMS Alpha (based on Mozilla) and a command procedure to launch the browser.

9.3.1 Online Formats

The documentation CDs contain documentation in the following formats:

Documentation	Available Formats
Current OpenVMS manuals	HTML, PDF
HP OpenVMS Version 8.2 Upgrade and Installation Manual	HTML, PDF
HP OpenVMS Version 8.2 Release Notes	HTML, PDF
HP OpenVMS Version 8.2 New Features and Documentation Overview	HTML, PDF
Layered product documents	HTML, PDF

Bookreader files are no longer available on the documentation CDs.

9.3.2 PDF Reader

The Adobe Acrobat Reader is provided for viewing PDF files. This self-extracting file can be installed on a computer running Windows. It is located on the ISO 9660 Level 2 CD.

For information about how to access documents on the documentation CDs and about the PDF reader, refer to the *HP OpenVMS Version 8.2 Upgrade and Installation Manual*.

9.4 Online Documentation on the OpenVMS Web Site

You can access OpenVMS manuals in various online formats from the following OpenVMS Web site:

`http://www.hp.com/go/openvms/doc`

This site contains links to current versions of manuals in the OpenVMS Full Documentation Set as well as to manuals for selected layered products.

9.5 Online Help

The OpenVMS operating system provides online help for the commands, utilities, and system routines documented in the Full Documentation set.

You can use the Help Message facility to quickly access online descriptions of system messages. In addition, you can add your own source files, such as messages documentation that you have written to the Help Message database.

The *OpenVMS System Messages: Companion Guide for Help Message Users* manual explains how to use the Help Message facility. You can also access DCL Help for Help Message by entering:

```
$ HELP HELP/MESSAGE
```

Reference information for OpenVMS utility routines is also included in online help.

10 Descriptions of OpenVMS Manuals

This chapter provides summary descriptions for the following OpenVMS documentation:

- Manuals in the OpenVMS Media Kit (Section 10.1)
- Manuals in the OpenVMS Base and Full Documentation sets (Section 10.2 and Section 10.3)
- RMS Journaling manual (Section 10.4)
- Manuals in the OpenVMS I64 OE Extensions Kit
- Archived manuals (Section 10.6)

10.1 Manuals in the OpenVMS Media Kit

Guide to HP OpenVMS Version 8.2 Media

Provides information about the OpenVMS Alpha operating system and documentation CDs. Lists the contents of the OpenVMS Alpha and the OpenVMS I64 Version 8.2 media kits, includes pointers to installation information, and gives instructions about how to access manuals on the documentation CD.

HP OpenVMS License Management Utility Manual

Describes the License Management Facility (LMF), the OpenVMS license management tool. LMF includes the License Management Utility (LICENSE) and VMSLICENSE.COM, the command procedure you use to register, manage, and track software licenses.

HP OpenVMS Version 8.2 Upgrade and Installation Manual

Provides step-by-step instructions for installing the OpenVMS Alpha and OpenVMS I64 operating systems on their respective platforms. Includes information about booting, shutdown, backup, and licensing procedures.

HP OpenVMS Version 8.2 New Features and Documentation Overview

Describes new and improved components for the I64 and Alpha operating systems for the Version 8.2 release. Includes information about OpenVMS documentation changes for Version 8.2 as well as the printed and online OpenVMS documentation offerings.

HP OpenVMS Version 8.2 Release Notes

Describes changes to the software; installation, upgrade, and compatibility information; new and existing software problems and restrictions; and software and documentation corrections.

10.2 Manuals in the OpenVMS Base Documentation Set

HP OpenVMS DCL Dictionary

Describes the DIGITAL Command Language (DCL) and provides an alphabetical listing of detailed reference information and examples for all DCL commands and lexical functions. This manual is in two volumes.

HP OpenVMS Guide to System Security

Describes the security features available in the OpenVMS Alpha and VAX operating systems. Explains the purpose and proper application of each feature in the context of specific security needs.

HP OpenVMS System Management Utilities Reference Manual

Presents reference information about the utilities you can use to perform system management tasks on your system as well as the tools to control and monitor system access and resources. Includes a description of the AUTOGEN command procedure. This manual is in two volumes.

HP OpenVMS System Manager's Manual, Volume 1: Essentials

Provides instructions for setting up and maintaining routine operations such as starting up the system, installing software, and setting up print and batch queues. Also explains routine disk and magnetic tape operations.

HP OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems

Describes how to configure and control the network, how to monitor the system, and how to manage system parameters. Also includes information about OpenVMS Cluster systems, network environments, and DECdtm functionality.

OpenVMS User's Manual

Provides an overview of the operating system and presents basic concepts, task information, and reference information that allow you to perform daily computing tasks. Describes how to work with files and directories. Also includes these additional topics:

- Sending messages with the Mail utility and the Phone utility
- Using the Sort/Merge utility
- Using logical names and symbols
- Writing command procedures
- Editing files with the EVE and EDT text editors

HP OpenVMS Version 8.2 New Features and Documentation Overview

Describes new and improved components for the Alpha and VAX operating systems for the Version 8.2 release. Includes information about OpenVMS documentation changes for Version 8.2 as well as the printed and online OpenVMS documentation offerings.

HP OpenVMS Version 8.2 Release Notes

Describes changes to the software; installation, upgrade, and compatibility information; new and existing software problems and restrictions; and software and documentation corrections.

10.3 Additional Manuals in the OpenVMS Full Documentation Set

HP OpenVMS Availability Manager User's Guide

Describes how to use the HO Availability Manager system management tool, from either an OpenVMS Alpha or a Windows node, to monitor one or more OpenVMS nodes on an extended local area network (LAN) or to target a specific node or process for detailed analysis.

COM, Registry, and Events for HP OpenVMS Developer's Guide

For programmers developing applications that move easily between the OpenVMS and Windows NT environments. Read this manual if you are encapsulating existing OpenVMS applications or data, or creating new COM appplications for OpenVMS systems. It also provides information for those who want to use the OpenVMS Registry to store information about their OpenVMS systems alone, or who want to use the OpenVMS Registry as a shared repository for both OpenVMS and Windows NT registry information. This manual was formerly available online as the *OpenVMS Connectivity Developer Guide*.

HP C Run-Time Library Reference Manual for OpenVMS Systems

Provides reference information on the functions and macros found in the HP C RTL that perform I/O operations, character and string manipulation, mathematical operations, error detection, subprocess creation, system access, and screen management. Includes portability concerns between operating systems, and describes the HP C for OpenVMS socket routines used for writing Internet application programs for the TCP/IP protocol.

Compaq C Run-Time Library Utilities Reference Manual

Provides detailed usage and reference information about the Run-Time Library utilities for managing localization and time zone data in international software applications.

Compaq Portable Mathematics Library

Documents the mathematics routines in the Compaq Portable Mathematics Library (DPML), supplied only with OpenVMS Alpha systems. VAX programmers should refer to the *OpenVMS VAX RTL Mathematics (MTH$) Manual*.

DECamds User's Guide

Provides information for installing and using the DECamds software. DECamds is a system management tool that lets you monitor, diagnose, and track events in OpenVMS system and OpenVMS Cluster environments.

DEC Text Processing Utility Reference Manual

Describes the DEC Text Processing Utility (DECTPU) and provides reference information about the EDT Keypad Emulator interfaces to DECTPU.

Extensible Versatile Editor Reference Manual

Contains command reference information about the EVE text editor. Also provides a cross-reference between EDT and EVE commands.

Guidelines for OpenVMS Cluster Configurations

This manual provides information to help you choose systems, interconnects, storage devices, and software. It can help you configure these components to achieve high availability, scalability, performance, and ease of system management. Detailed directions using SCSI and Fibre Channel in an OpenVMS Cluster system are also included in this manual.

Guide to Creating OpenVMS Modular Procedures

Describes how to perform a complex programming task by dividing it into modules and coding each module as a separate procedure.

Guide to OpenVMS File Applications

Contains guidelines for designing, creating, and maintaining efficient data files by using Record Management Services (RMS). This manual is intended for application programmers and designers responsible for programs that use RMS files, especially if performance is an important consideration.

Guide to the POSIX Threads Library

Describes the POSIX Threads Library (formerly named DECthreads) package, HP's multithreading run-time libraries. Use the routines in this package to create and control multiple threads of execution within the address space provided by a single process. Offering both usage tips and reference synopses, this document describes three interfaces: routines that conform to the IEEE POSIX 1003.1c standard (called *pthread*), routines that provide thread-related services in nonthreaded applications (called thread-independent services or *tis*), and a set of HP proprietary routines (called *cma*) that provide a stable, upwardly compatible interface.

Guide to the DEC Text Processing Utility

Provides an introduction to developing DECTPU programs.

HP Open Source Security for OpenVMS, Volume 1: Common Data Security Architecture

For application developers who want to use the Common Data Security Architecture (CDSA) to add security to their programs. Describes CDSA, gives information about installation and initialization, and provides example programs. Contains the CDSA application programming interface modules.

HP Open Source Security for OpenVMS, Volume 2: HP SSL for OpenVMS

For application developers who want to protect communication links to OpenVMS applications with HP Secure Sockets Layer (HP SSL) for OpenVMS. Contains installation instructions, release notes, and provides example programs. Includes programming information and a reference section for the OpenSSL application programming interface modules.

HP OpenVMS Alpha Partitioning and Galaxy Guide

Provides complete details about how to use all of the OpenVMS Galaxy features and capabilities available in OpenVMS Alpha Version 7.3–2. Includes procedures for creating, managing, and using OpenVMS Galaxy computing environments on AlphaServer 8400, 8200, and 4100 systems.

HP OpenVMS Guide to Upgrading Privileged-Code Applications

Explains the OpenVMS Alpha Version 7.0 changes that might impact Alpha privileged-code applications and device drivers as a result of the OpenVMS Alpha 64-bit virtual addressing and kernel threads support provided in OpenVMS Alpha Version 7.0.

Privileged-code applications from versions prior to OpenVMS Alpha Version 7.0 might require the source-code changes described in this guide.

HP OpenVMS System Analysis Tools Manual

Describes the following system analysis tools in detail, while also providing a summary of the dump off system disk (DOSD) capability and the DELTA/XDELTA debugger:

- System Dump Analyzer (SDA)
- System Code Debugger (SCD)
- System Dump Debugger (SDD)
- Watchpoint utility

Intended primarily for the system programmer who must investigate the causes of system failures and debug kernel mode code, such as a device driver.

HP OpenVMS Calling Standard

Documents the calling standard for the OpenVMS I64, Alpha, and VAX operating systems.

HP OpenVMS Cluster Systems

Describes procedures and guidelines for configuring and managing OpenVMS Cluster systems. Also describes how to provide high availability, building-block growth, and unified system management across clustered systems.

HP OpenVMS Command Definition, Librarian, and Message Utilities Manual

Contains descriptive and reference information about the following utilities:

- Command Definition utility
- Librarian utility
- Message utility

HP OpenVMS Debugger Manual

Explains the features of the OpenVMS Debugger for programmers.

HP OpenVMS Delta/XDelta Debugger Manual

Describes the Delta/XDelta utility used to debug programs that run in privileged processor mode or at an elevated interrupt priority level.

HP OpenVMS I/O User's Reference Manual

Contains the information that system programmers need to program I/O operations using the device drivers that are supplied with the operating system.

HP OpenVMS Linker Utility Manual

Describes how to use the Linker utility to create images that run on OpenVMS systems. Also explains how to control a link operation with link qualifiers and link options.

HP OpenVMS MACRO Compiler Porting and User's Guide

Describes how to port existing VAX MACRO assembly language code to an OpenVMS Alpha system by using the features of the MACRO-32 compiler. It also describes how to port existing OpenVMS Alpha code to OpenVMS I64 systems. Also documents how to use the compiler's 64-bit addressing support.

HP OpenVMS Management Station Overview and Release Notes

Provides an overview and release notes for OpenVMS Management Station and describes how to get started using the software. OpenVMS Management Station is a powerful, Microsoft Windows based management tool for system managers and others who perform user account and printer management tasks on OpenVMS systems.

OpenVMS Performance Management

Introduces and explains the techniques used to optimize performance on an OpenVMS system.

Porting Applications from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers

Provides a framework for application developers who are migrating from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers.

HP OpenVMS Programming Concepts Manual

Describes concepts such as process creation, kernel threads and the kernel threads process structure, interprocess communication, process control, data sharing, condition handling, and ASTs. This two-volume manual uses system services, utility routines, and run-time library (RTL) routines to illustrate mechanisms for utilizing OpenVMS features.

OpenVMS Record Management Services Reference Manual

Provides reference and usage information for all programmers who use RMS data files.

OpenVMS Record Management Utilities Reference Manual

Contains descriptive and reference information about the following RMS utilities:

- Analyze/RMS_File utility
- Convert and Convert/Reclaim utilities
- File Definition Language facility

HP OpenVMS RTL General Purpose (OTS$) Manual

Documents the general-purpose routines contained in the OTS$ facility of the OpenVMS Run-Time Library. Indicates which routines are specific to I64, Alpha or VAX, as well as how routines function differently on each system.

HP OpenVMS RTL Library (LIB$) Manual

Documents the general-purpose routines contained in the LIB$ facility of the OpenVMS Run-Time Library. Indicates which routines are specific to I64, Alpha or VAX, as well as how routines function differently on each system.

OpenVMS RTL Screen Management (SMG$) Manual

Documents the screen management routines contained in the SMG$ facility of the OpenVMS Run-Time Library. Indicates which routines are specific to Alpha or VAX, as well as how routines function differently on each system.

OpenVMS RTL String Manipulation (STR$) Manual

Documents the string manipulation routines contained in the STR$ facility of the OpenVMS Run-Time Library. Indicates which routines are specific to Alpha or VAX, as well as how routines function differently on each system.

OpenVMS System Messages: Companion Guide for Help Message Users

Describes features of the Help Message facility, a tool that you can use to display message descriptions. Describes the HELP/MESSAGE command and qualifiers and also includes detailed information about customizing the Help Message database. Also provides descriptions of messages that can occur when the system and Help Message are not fully operable.

HP OpenVMS System Services Reference Manual

Presents the set of routines that the operating system uses to control resources, allow process communication, control I/O, and perform other such operating system functions. This manual is in two volumes.

OpenVMS Utility Routines Manual

Describes the routines that allow a program to use the callable interface of selected OpenVMS utilities.

OpenVMS VAX RTL Mathematics (MTH$) Manual

Documents the mathematics routines contained in the MTH$ facility of the OpenVMS Run-Time Library, which is relevant only to programmers using OpenVMS VAX. (Alpha programmers should refer to *Compaq Portable Mathematics Library*.)

OpenVMS VAX System Dump Analyzer Utility Manual

Explains how to use the System Dump Analyzer utility to investigate system failures and examine a running OpenVMS VAX system. VAX programmers should refer to this manual; Alpha and I64 programmers should refer to the *OpenVMS Alpha System Dump Analyzer Utility Manual*.

POLYCENTER Software Installation Utility Developer's Guide

Describes the procedure and provides guidelines for developing software products that will be installed using the POLYCENTER Software Installation utility. Intended for developers who are designing installation procedures for software products layered on the OpenVMS operating system.

VAX MACRO and Instruction Set Reference Manual

Documents both the assembler directives of VAX MACRO and the VAX instruction set.

HP Volume Shadowing for OpenVMS

Describes how to provide high data availability with phase II volume shadowing.

10.4 RMS Journaling Manual

RMS Journaling for OpenVMS Manual

Describes the three types of RMS Journaling as well as other OpenVMS components that support RMS Journaling. This manual also describes the RMS Recovery utility (which is used to recover data saved using journaling), the transaction processing system services, and system management tasks required when using RMS Journaling.

10.5 Manuals in the OpenVMS I64 OE Extensions Kit

The following list contains manuals relevant to the OpenVMS I64 Operating Environments.

- *HP DECwindows Motif for OpenVMS Installation Guide*
- *HP DECwindows Motif for OpenVMS New Features*
- *HP DECwindows Motif for OpenVMS Documentation Overview*
- *HP DECwindows Motif for OpenVMS Management Guide*
- *HP DECnet-Plus for OpenVMS Installation and Configuration*
- *HP DECnet-Plus for OpenVMS Introduction and User's Guide*
- *HP DECnet-Plus Network Management*
- *HP DECnet-Plus for OpenVMS DECdts Programming Reference*
- *HP DECnet-Plus for OpenVMS DECdts Management*
- *HP DECnet-Plus for OpenVMS DECdns Management*
- *HP DECnet-Plus for OpenVMS Network Management Quick Reference Guide*
- *HP DECnet-Plus for OpenVMS OSAK Programming*
- *HP DECnet-Plus for OpenVMS OSAK Programming Reference*
- *HP DECnet-Plus for OpenVMS OSAK SPI Programming Reference*
- *HP DECnet-Plus for OpenVMS Problem Solving Manual*
- *HP DECnet-Plus for OpenVMS Programming Manual*
- *HP DECnet-Plus for OpenVMS FTAM and Virtual Terminal User and Management*
- *HP DECnet-Plus for OpenVMS Problem Solving*
- *HP DECnet-Plus for OpenVMS Network Control Language Reference*
- *HP DECnet-Plus for OpenVMS Planning Guide*
- *HP TCP/IP Services for OpenVMS Installation and Configuration*
- *HP TCP/IP Services for OpenVMS Sockets API and System Services Programming*
- *HP TCP/IP Services for OpenVMS Concepts and Planning*
- *HP TCP/IP Services for OpenVMS SNMP Programming Reference*
- *HP TCP/IP Services for OpenVMS ONC RPC Programming*
- *HP TCP/IP Services for OpenVMS Tuning and Troubleshooting*
- *HP TCP/IP Services for OpenVMS Guide to SSH for OpenVMS*
- *HP TCP/IP Services for OpenVMS Management*
- *HP TCP/IP Services for OpenVMS Management Command Reference*
- *HP TCP/IP Services for OpenVMS Management Command Quick Reference Card*
- *HP TCP/IP Services for OpenVMS User's Guide*

- *HP TCP/IP Services for OpenVMS UNIX Command Equivalents Reference Card*
- *HP TCP/IP Services for OpenVMS Guide to IPv6*
- *HP DECprint Supervisor (DCPS) for OpenVMS User's Guide*
- *HP DECprint Supervisor (DCPS) for OpenVMS Software Installation*
- *HP DECprint Supervisor (DCPS) for OpenVMS Manager's Guide*
- *HP DCE for OpenVMS Product Guide*
- *HP DCE for OpenVMS Reference Guide*
- *HP DCE for OpenVMS Installation and Configuration Guide*

10.6 Archived Manuals

Table 10–1 lists the OpenVMS manuals that have been archived. Note that most information from the archived manuals has been incorporated in other documents or online help.

Table 10–1 Archived OpenVMS Manuals

Manual	Order Number
A Comparison of System Management on OpenVMS AXP and OpenVMS VAX	AA-PV71B-TE
Building Dependable Systems: The OpenVMS Approach	AA-PV5YB-TE
Creating an OpenVMS Alpha Device Driver from an OpenVMS VAX Device Driver	AA-R0Y8A-TE
Creating an OpenVMS AXP Step 2 Device Driver from a Step 1 Device Driver	AA-Q28TA-TE
Creating an OpenVMS AXP Step 2 Device Driver from an OpenVMS VAX Device Driver	AA-Q28UA-TE
Guide to OpenVMS AXP Performance Management	AA-Q28WA-TE
Guide to OpenVMS Performance Management	AA-PV5XA-TE
Migrating an Application from OpenVMS VAX to OpenVMS Alpha	AA-KSBKB-TE
Migrating an Environment from OpenVMS VAX to OpenVMS Alpha	AA-QSBLA-TE
Migrating to an OpenVMS AXP System: Planning for Migration	AA-PV62A-TE
Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications	AA-PV63A-TE
OpenVMS Alpha Guide to 64-Bit Addressing and VLM Features	AA-QSBCC-TE
OpenVMS Alpha System Dump Analyzer Utility Manual	AA-PV6UC-TE
OpenVMS AXP Device Support: Developer's Guide	AA-Q28SA-TE
OpenVMS AXP Device Support: Reference	AA-Q28PA-TE
OpenVMS Bad Block Locator Utility Manual	AA-PS69A-TE
OpenVMS Compatibility Between VAX and Alpha	AA-PYQ4C-TE
OpenVMS Developer's Guide to VMSINSTAL	AA-PWBXA-TE
OpenVMS DIGITAL Standard Runoff Reference Manual	AA-PS6HA-TE

(continued on next page)

Table 10–1 (Cont.) Archived OpenVMS Manuals

Manual	Order Number
OpenVMS EDT Reference Manual	AA-PS6KA-TE
OpenVMS Exchange Utility Manual	AA-PS6AA-TE
OpenVMS Glossary	AA-PV5UA-TK
OpenVMS Guide to Extended File Specifications	AA-REZRB-TE
OpenVMS Master Index	AA-QSBSD-TE
OpenVMS National Character Set Utility Manual	AA-PS6FA-TE
OpenVMS Obsolete Features Manual	AA-PS6JA-TE
OpenVMS Programming Environment Manual	AA-PV66B-TK
OpenVMS Programming Interfaces: Calling a System Routine	AA-PV68B-TK
OpenVMS RTL DECtalk (DTK$) Manual	AA-PS6CA-TE
OpenVMS RTL Parallel Processing (PPL$) Manual	AA-PV6JA-TK
OpenVMS Software Overview	AA-PVXHB-TE
OpenVMS SUMSLP Utility Manual	AA-PS6EA-TE
OpenVMS System Messages and Recovery Procedures Reference Manual: A–L	AA-PVXKA-TE
OpenVMS System Messages and Recovery Procedures Reference Manual: M–Z	AA-PVXLA-TE
OpenVMS Terminal Fallback Utility Manual	AA-PS6BA-TE
OpenVMS VAX Card Reader, Line Printer, and LPA11–K I/O User's Reference Manual	AA-PVXGA-TE
OpenVMS VAX Device Support Manual	AA-PWC8A-TE
OpenVMS VAX Device Support Reference Manual	AA-PWC9A-TE
OpenVMS VAX Patch Utility Manual	AA-PS6DA-TE
OpenVMS Wide Area Network I/O User's Reference Manual	AA-PWC7A-TE
PDP-11 TECO User's Guide	AA-K420B-TC
POLYCENTER Software Installation Utility User's Guide	AA-Q28NA-TK
TCP/IP Networking on OpenVMS Systems	AA-QJGDB-TE
Standard TECO Text Editor and Corrector for the VAX, PDP-11, PDP-10, and PDP-8	Available only on CD

Table 10–2 lists the networking manuals and installation supplements that have been archived.

Table 10–2 Archived Networking Manuals and Installation Supplements

Manual	Order Number
DECnet for OpenVMS Guide to Networking	AA-PV5ZA-TK
DECnet for OpenVMS Network Management Utilities	AA-PV61A-TK
DECnet for OpenVMS Networking Manual	AA-PV60A-TK

(continued on next page)

Table 10–2 (Cont.) Archived Networking Manuals and Installation Supplements

Manual	Order Number
OpenVMS VAX Upgrade and Installation Supplement: VAX 8820, 8830, 8840	AA-PS6MA-TE
OpenVMS VAX Upgrade and Installation Supplement: VAX 8200, 8250, 8300, 8350	AA-PS6PA-TE
OpenVMS VAX Upgrade and Installation Supplement: VAX 8530, 8550, 8810 (8700), and 8820–N (8800)	AA-PS6QA-TE
OpenVMS VAX Upgrade and Installation Supplement: VAX 8600, 8650	AA-PS6UA-TE
VMS Upgrade and Installation Supplement: VAX-11/780, 785	AA-LB29B-TE
VMS Upgrade and Installation Supplement: VAX-11/750	AA-LB30B-TE

Descriptions of the archived OpenVMS manuals are as follows:

A Comparison of System Management on OpenVMS AXP and OpenVMS VAX

Discusses system management tools, the impact of Alpha page sizes on system management operations, the system directory structure, interoperability issues, and performance information. Designed for system managers who need to learn quickly how to manage an OpenVMS Alpha system.

Building Dependable Systems: The OpenVMS Approach

Offers practical information about analyzing the dependability requirements of your business applications and deciding how to use your computing systems to support your dependability goals. This information is complemented by technical summaries of the dependability features of OpenVMS and related hardware and layered software products.

Creating an OpenVMS Alpha Device Driver from an OpenVMS VAX Device Driver

Describes the procedures for converting a device driver used on OpenVMS VAX to a device driver that runs on OpenVMS Alpha. This book also contains data structures, routines, and macros for maintaining an Alpha driver written in Macro-32.

Creating an OpenVMS AXP Step 2 Device Driver from a Step 1 Device Driver

Provides information for upgrading a Step 1 device driver (used in earlier versions of OpenVMS AXP) to a Step 2 device driver. A Step 2 device driver is required for OpenVMS AXP Version 6.1.

Creating an OpenVMS AXP Step 2 Device Driver from an OpenVMS VAX Device Driver

Provides information for migrating a device driver used on OpenVMS VAX to a Step 2 device driver used on OpenVMS AXP Version 6.1.

Guide to OpenVMS AXP Performance Management

Introduces and explains the techniques used to optimize performance on an OpenVMS Alpha system.

Guide to OpenVMS Performance Management

Introduces and explains the techniques used to optimize performance on an OpenVMS VAX system.

Migrating an Application from OpenVMS VAX to OpenVMS Alpha

Describes how to create an OpenVMS Alpha version of an OpenVMS VAX application. Provides an overview of the VAX to Alpha migration process and information to help you plan a migration. It discusses the decisions you must make in planning a migration and the ways to get the information you need to make those decisions. In addition, this manual describes the migration methods available so that you can estimate the amount of work required for each method and select the method best suited to a given application.

Migrating an Environment from OpenVMS VAX to OpenVMS Alpha

Describes how to migrate a computing environment from an OpenVMS VAX system to an OpenVMS Alpha system or a mixed-architecture cluster. Provides an overview of the VAX to Alpha migration process and describes the differences in system and network management on VAX and Alpha computers.

Migrating to an OpenVMS AXP System: Planning for Migration

Describes the general characteristics of RISC architectures, compares the Alpha architecture to the VAX architecture, and presents an overview of the migration process and a summary of migration tools provided by HP. The information in this manual is intended to help you define the optimal migration strategy for your application.

Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications

Provides detailed technical information for programmers who must migrate high-level language applications to OpenVMS Alpha. Describes how to set up a development environment to facilitate the migration of applications, helps programmers identify application dependencies on elements of the VAX architecture, and introduces compiler features that help resolve these dependencies. Individual sections of this manual discuss specific application dependencies on VAX architectural features, data porting issues (such as alignment concerns), and the process of migrating VAX shareable images.

OpenVMS Alpha Guide to 64-Bit Addressing and VLM Features

Introduces and describes OpenVMS Alpha operating system support for 64-bit virtual addressing and Very Large Memory (VLM). Intended for system and application programmers, this guide highlights the features and benefits of OpenVMS Alpha 64-bit and VLM capabilities. It also describes how to use these features to enhance application programs to support 64-bit addresses and to efficiently harness very large physical memory.

OpenVMS Alpha System Dump Analyzer Utility Manual

Explains how to use the System Dump Analyzer utility to investigate system failures and examine a running OpenVMS Alpha system. Alpha programmers should refer to this manual; VAX programmers should refer to the *OpenVMS VAX System Dump Analyzer Utility Manual*.

OpenVMS AXP Device Support: Developer's Guide

Describes how to write a driver for OpenVMS Alpha for a device not supplied by Compaq.

OpenVMS AXP Device Support: Reference

Provides the reference material for the *Writing OpenVMS Alpha Device Drivers in C* by describing the data structures, macros, and routines used in device-driver programming.

OpenVMS Bad Block Locator Utility Manual

Describes how to use the Bad Block Locator utility to locate bad blocks on older types of media.

OpenVMS Compatibility Between VAX and Alpha

Compares and contrasts OpenVMS on VAX and Alpha computers, focusing on the features provided to end users, system managers, and programmers.

OpenVMS Developer's Guide to VMSINSTAL

Describes the VMSINSTAL command procedure and provides guidelines for designing installation procedures that conform to standards recommended by Compaq. Intended for developers who are designing installation procedures for software products layered on the OpenVMS operating system.

OpenVMS DIGITAL Standard Runoff Reference Manual

Describes the DSR text-formatting utility.

OpenVMS EDT Reference Manual

Contains complete reference information for the EDT editor.

OpenVMS Exchange Utility Manual

Describes how to use the Exchange utility to transfer files between some foreign format volumes and OpenVMS native volumes.

OpenVMS Glossary

Defines terms specific to OpenVMS that are used throughout the documentation.

OpenVMS Guide to Extended File Specifications

Provides an overview of Extended File Specifications and describes the overall differences and impact Extended File Specifications introduce to the OpenVMS environment.

OpenVMS Master Index

Offers an edited compilation of indexes from the manuals in the OpenVMS Full Documentation set.

OpenVMS National Character Set Utility Manual

Describes how to use the National character set utility to build NCS definition files.

OpenVMS Obsolete Features Manual

Presents the DCL commands, system services, RTL routines, and utilities made obsolete by VMS Version 4.0 through Version 5.0. Includes an appendix of DCL commands, RTL routines, and utilities eliminated from VMS Version 4.0.

OpenVMS Programming Environment Manual

Provides a general description of Compaq products and tools that define the programming environment. Introduces facilities and tools such as the compilers, the linker, the debugger, the System Dump Analyzer, system services, and routine libraries.

OpenVMS Programming Interfaces: Calling a System Routine

Describes the OpenVMS programming interface and defines the standard conventions to call an OpenVMS system routine from a user procedure. The Alpha and VAX data type implementations for various high-level languages are also presented in this manual.

OpenVMS RTL DECtalk (DTK$) Manual

Documents the DECtalk support routines contained in the DTK$ facility of the OpenVMS Run-Time Library.

OpenVMS RTL Parallel Processing (PPL$) Manual

Documents the parallel-processing routines contained in the PPL$ facility of the OpenVMS Run-Time Library. Indicates which routines are specific to Alpha or VAX, as well as how routines function differently on each system.

OpenVMS Software Overview

Provides an overview of the OpenVMS operating system and some of its available products.

OpenVMS SUMSLP Utility Manual

Describes how to use the SUMSLP batch-oriented editor to update source files.

OpenVMS System Messages and Recovery Procedures Reference Manual

Contains an alphabetical listing of the errors, warnings, and informational messages issued by the operating system. Also provides the meaning of each message and a statement of the action to be taken in response to each message. This manual is in two volumes.

OpenVMS Terminal Fallback Utility Manual

Describes how to use the Terminal Fallback utility to manage the libraries, character conversion tables, and terminal parameters that are available within this utility.

OpenVMS VAX Card Reader, Line Printer, and LPA11–K I/O User's Reference Manual

Describes the card reader, laboratory peripheral accelerator, and line printer drivers on OpenVMS VAX.

OpenVMS VAX Device Support Manual

Describes how to write an OpenVMS VAX driver for a device not supplied by Compaq.

OpenVMS VAX Device Support Reference Manual

Provides the reference material for the *OpenVMS VAX Device Support Manual* by describing the data structures, macros, and routines used in device-driver programming.

OpenVMS VAX Patch Utility Manual

Describes how to use the Patch utility to examine and modify executable and shareable OpenVMS VAX images.

OpenVMS Wide Area Network I/O User's Reference Manual

Describes the DMC11/DMR11, DMP11 and DMF32, DR11-W and DRV11-WA, DR32, and asynchronous DDCMP interface drivers on OpenVMS VAX.

PDP–11 TECO User's Guide

Describes the operating procedures for the PDP-11 TECO (Text Editor and Corrector) program.

POLYCENTER Software Installation Utility User's Guide

Provides information on the POLYCENTER Software Installation utility, a new component that lets you install and manage software products that are compatible with the utility.

TCP/IP Networking on OpenVMS Systems

Provides an introductory overview of TCP/IP networking and describes OpenVMS DCL support for TCP/IP capabilities.

Index

A

AES, 5–3
Analyze utility, 4–1
ATI RADEON graphics, 5–1

B

Based clusters, 7–4
Binary translator, 4–35

C

CDSA, 5–1
Checksum utility, 4–1
  enhanced for I64 objects, 4–2
Clusters, 3–1
Commands
  SDA, 3–8
  SET, 4–36
  SHOW, 4–36
C Run-Time Library (RTL) enhancements, 4–2

D

DCE RPC, 4–4
DCL command enhancements, 2–1
Debugger, 4–4
  Intel Itanium architecture support, 4–4
  supported Intel Itanium hardware registers, 4–4
  supported languages, 4–5
Device drivers
  debugging, 10–5
  supplied with OpenVMS, 10–5
  writing, 10–4

E

ELF type
  STB_VMS_WEAK, 4–34
  STB_WEAK, 4–34
Elliptic curve cryptography, 5–4
Encryption type, 5–1
Enterprise Operating Environment, 2–3
ERLBUFFERPAG_S2 parameter, 3–10
ERRORLOGBUFF_S2 parameter, 3–10
Executable and Linkable Format (ELF), 4–34
Extended Lock Value Block, 4–5

F

Fibre Channel HBA support, 3–7
Foundation Operating Environement, 2–3

H

HBMM
  See Volume Shadowing for OpenVMS
HELP/MESSAGE facility, 9–7
HELP command, 9–7
Host-Based Minimerge (HBMM), 3–12
HP Secure Web Browser for OpenVMS Alpha, 9–6

I

Initialized overlaid program sections, 7–4

K

Kerberos, 5–2
ktutil, 5–2

L

LBR$DELETE_DATA library routine, 4–16
LBR$DELETE_KEY library routine, 4–18
LBR$GET_INDEX library routine, 4–20
LBR$INSERT_KEY library routine, 4–23
LBR$LOOKUP_KEY library routine, 4–25
LBR$LOOKUP_TYPE library routine, 4–10
LBR$MAP_MODULE library routine, 4–12
LBR$PUT_MODULE library routine, 4–13
LBR$PUT_RECORD library routine, 4–27
LBR$REPLACE_KEY library routine, 4–29
LBR$SEARCH library routine, 4–31
LBR$UNMAP_MODULE library routine, 4–14
LBR library services, 4–13
Lexical function enhancements, 2–1
Librarian utility, 4–5
  changes to (I64 only), 4–6
  defaults, 4–6
  enhanced /REMOVE, 4–7

Librarian utility (cont'd)
  format changes, 4–34
  new group section symbols, 4–35
  new UNIX-style weak symbols, 4–34
  precedence ordering rules, 4–35
  usage summary, 4–6
  use of /ALPHA and /VAX qualifiers, 4–7
  V6.0 library index format, 4–34
Library index format
  V6.0 libraries, 4–34
Library routines
  accessing ELF object libraries, 4–8
  changes to, 4–7
  extended, 4–15
  LBR$DELETE_DATA, 4–16
  LBR$DELETE_KEY, 4–18
  LBR$GET_INDEX, 4–20
  LBR$INSERT_KEY, 4–23
  LBR$LOOKUP_KEY, 4–25
  LBR$LOOKUP_TYPE, 4–10
  LBR$MAP_MODULE, 4–12
  LBR$PUT_MODULE, 4–13
  LBR$PUT_RECORD, 4–27
  LBR$REPLACE_KEY, 4–29
  LBR$SEARCH, 4–31
  LBR$UNMAP_MODULE, 4–14
  new, 4–9
  new library types, 4–7
License Management Facility, 2–2
Linker
  conventions for image names, 7–18
  for OpenVMS Alpha
    hangs when processing many files, 7–19
    RMS_RELATED_CONTEXT option, 7–19
  for OpenVMS I64
    alignments for PSECT_ATTRIBUTE option, 7–19
    flags set for /TRACEBACK, /DEBUG, /DSF, 7–8
    handling initialized overlaid psects, 7–4
    linking ELF common symbols selectively against sharable images, 7–7
  images compiled with reduced floating point, 7–12
  linker map, 7–21
  linking with ELF group symbols, 7–12
  linking with UNIX-style weak symbols, 7–12
  on I64 systems, 7–9
  options, 7–17
  program section synopsis, 7–5
  restrictions for /SEGMENT_ATTRIBUTE, 7–14
  understanding messages, 7–10
  utility, 7–1
Linking on OpenVMS I64 systems, 7–2
LMF, 2–2
Lock management
  Extended Lock Value Block, 4–5
Logging
  SSL, 3–8
  system service, 3–8
Lowercase symbol names, 4–36

M

Merge and copy operations
  prioritizing, 3–12
Migration software, 4–35
Mission Critical Operating Environment, 2–3
Monitor utility, 2–3
Mozilla, 9–6

N

Networking products, 5–5
New Features
  Debugger, 4–4
  XDELTA, 4–46
New Linker qualifiers, 7–2

O

OCSP, 5–3
OpenSSL, 5–3
OpenVMS I64 Boot Manager, 3–1
Operating environments, 2–3

P

Patch utility, 4–41
POSIX Threads, 4–36
PSECT_ATTRIBUTE options, 7–17

R

RESET_THRESHOLD policy keyword, 3–12
RTL LIB routines, 4–39
  LIB$CVTS_FROM_INTERNAL_TIME, 4–39
  LIB$EMODS, 4–39
  LIB$EMODT, 4–39
  LIB$GET_UIB_INFO, 4–39
  LIB$I64_CREATE_INVO_CONTEXT, 4–39
  LIB$I64_FREE_INVO_CONTEXT, 4–39
  LIB$I64_GET_CURR_INVO_CONTEXT, 4–39
  LIB$I64_GET_CURR_INVO_HANDLE, 4–39
  LIB$I64_GET_FR, 4–39
  LIB$I64_GET_GR, 4–39
  LIB$I64_GET_INVO_CONTEXT, 4–39
  LIB$I64_GET_PREV_INVO_CONTEXT, 4–39
  LIB$I64_GET_PREV_INVO_HANDLE, 4–39
  LIB$I64_GET_UNWIND_HANDLER_FV, 4–39
  LIB$I64_GET_UNWIND_LSDA, 4–39
  LIB$I64_GET_UNWIND_OSSD, 4–39
  LIB$I64_INIT_INVO_CONTEXT, 4–39
  LIB$I64_INVO_HANDLE, 4–39
  LIB$I64_IS_AST_DISPATCH_FRAME, 4–39

RTL LIB routines (cont'd)
  LIB$I64_IS_EXC_DISPATCH_FRAME, 4–39
  LIB$I64_PREV_INVO_END, 4–40
  LIB$I64_PUT_INVO_REGISTERS, 4–40
  LIB$I64_SET_FR, 4–40
  LIB$I64_SET_GR, 4–40
  LIB$I64_SET_PC, 4–40
  LIB$LOCK_IMAGE, 4–40
  LIB$MULTS_DELTA_TIME, 4–40
  LIB$POLYS, 4–40
  LIB$POLYT, 4–40
  LIB$UNLOCK_IMAGE, 4–40
RTL OTS routines, 4–40
  OTS$CNVOUT_S, 4–40
  OTS$CNVOUT_T, 4–40
  OTS$CVT_T_S, 4–40
  OTS$CVT_T_T, 4–40
  OTS$DIVCS_R3, 4–40
  OTS$DIVCT_R3, 4–40
  OTS$MULCT_R3, 4–40
  OTS$POWCSCS_R3, 4–41
  OTS$POWCSCT_R3, 4–41
  OTS$POWCSJ, 4–41
  OTS$POWCTJ, 4–41
  OTS$POWSJ, 4–41
  OTS$POWSLU, 4–41
  OTS$POWSS, 4–41
  OTS$POWTJ, 4–41
  OTS$POWTLU, 4–41
  OTS$POWTT, 4–41

S

SCSI_ERROR_POLL parameter, 3–10
SDA commands, 3–8
SET command, 4–36
SET SHADOW command
  /EVALUATE=RESOURCES, 3–12
  /PRIORITY, 3–12
SHADOW_ENABLE parameter, 3–10
SHADOW_HBMM_RTC parameter, 3–10, 3–12
SHADOW_MAX_COPY parameter, 3–12
SHADOW_PSM_DLY parameter, 3–10
SHADOW_REC_DLY parameter, 3–11, 3–13
SHADOW_SITE_ID parameter, 3–11
SHOW command, 4–36
SSL, 5–3
  System Service Logging, 3–8
Symbol definiton
  OpenVMS-style weak, 4–34
  Unix-style weak, 4–34
SYSSER_LOGGING parameter, 3–11
System parameters
  ERLBUFFERPAG_S2, 3–10
  ERRORLOGBUFF_S2, 3–10
  new in Version 8.2, 3–9
  SCSI_ERROR_POLL, 3–10
  SHADOW_ENABLE, 3–10

System parameters (cont'd)
  SHADOW_HBMM_RTC, 3–10, 3–12
  SHADOW_MAX_COPY, 3–12
  SHADOW_PSM_DLY, 3–10
  SHADOW_REC_DLY, 3–11, 3–13
  SHADOW_SITE_ID, 3–11
  SYSSER_LOGGING, 3–11
  TTY_DEFCHAR3, 3–11
  VHPT_SIZE, 3–11
System Service Logging, 3–8
System services, 4–41

T

TCP/IP Services for OpenVMS, 5–5
Time zones, 3–12, 4–44
Traceback, 4–44
TTY_DEFCHAR3 parameter, 3–11

U

Ultra-SCSI HBA adapter support, 3–8
UNIX-style weak symbols, 4–34

V

VHPT_SIZE parameter, 3–11
Volume Shadowing for OpenVMS
  HBMM, 3–12
  prioritizing merge and copy operations, 3–12

W

Web browser, 9–6

X

XDELTA
  new features, 4–46