OpenVMS Programming Environment Manual

Order Number: AA–PV66B–TK

March 1994

This manual describes the OpenVMS programming environment and shows how Digital products and tools can be integrated into the software development process.

Revision/Update Information: This manual supersedes the *OpenVMS Programming Environment Manual*, OpenVMS AXP Version 1.5 and OpenVMS VAX Version 6.0.

Software Version: OpenVMS AXP Version 6.1
OpenVMS VAX Version 6.1

Digital Equipment Corporation
Maynard, Massachusetts

March 1994

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

ZK5842

This document is available on CD–ROM.

This document was prepared using VAX DOCUMENT Version 2.1.

Send Us Your Comments

We welcome your comments on this or any other OpenVMS manual. If you have suggestions for improving a particular section or find any errors, please indicate the title, order number, chapter, section, and page number (if available). We also welcome more general comments. Your input is valuable in improving future releases of our documentation.

You can send comments to us in the following ways:

- Internet electronic mail: OPENVMSDOC@ZKO.MTS.DEC.COM
- Fax: 603-881-0120 Attn: OpenVMS Documentation, ZKO3-4/U08
- A completed Reader's Comments form (postage paid, if mailed in the United States), or a letter, via the postal service. Two Reader's Comments forms are located at the back of each printed OpenVMS manual. Please send letters and forms to:

  Digital Equipment Corporation
  Information Design and Consulting
  OpenVMS Documentation
  110 Spit Brook Road, ZKO3-4/U08
  Nashua, NH 03062-2698
  USA

You may also use an online questionnaire to give us feedback. Print or edit the online file SYS$HELP:OPENVMSDOC_SURVEY.TXT. Send the completed online file by electronic mail to our Internet address, or send the completed hardcopy survey by fax or through the postal service.

Thank you.

Contents

4 User Interface Tools for OpenVMS Applications

5 Editors

6 Tools for Managing Program Files

7 Compilers, Interpreters, and Assemblers

8 Linker and Librarian

9 Debugging and Testing Tools

10 Using Callable System Routines

11 Additional Programming Utilities

Index

Figures

Tables

Preface

Intended Audience

This manual is intended for programmers who want to become familiar with the OpenVMS operating system programming environment. The information in this manual applies to OpenVMS operating systems running on both VAX and AXP platforms. Unless otherwise noted, the environments function the same way.

Document Structure

This manual introduces the programming tools supported by the OpenVMS operating system. It does not cover programming concepts, nor is it intended to be a complete description of any one programming language or tool (see the list of related documentation in the Associated Documents section).

This book is organized as follows:

- Chapter 1 provides an overview of the OpenVMS software development tools.
- Chapter 2 discusses OpenVMS support of portable and interoperable applications.
- Chapter 3 describes OpenVMS support of distributed computing, including distributed applications.
- Chapter 4 discusses user interface tools for OpenVMS applications only. Tools for creating portable user interfaces are described in Chapter 3.
- Chapter 5 describes available tools for creating source files.
- Chapter 6 describes available tools for managing source files, objects, and images.
- Chapter 7 describes language compilers, interpreters, and assemblers.
- Chapter 8 covers the linker and librarian utilities.
- Chapter 9 describes features of debugging and testing tools.
- Chapter 10 provides an overview of callable system routines.
- Chapter 11 describes additional programming utilities.

Associated Documents

To find out more about using the programming tools described in this manual, refer to the following documents:

- *OpenVMS Compatibility Between VAX and AXP*
- *OpenVMS Programming Concepts Manual*
- *OpenVMS Programming Interfaces: Calling a System Routine*

- *OpenVMS Calling Standard*
- *OpenVMS Command Definition, Librarian, and Message Utilities Manual*
- *OpenVMS Debugger Manual*
- *OpenVMS Delta/XDelta Debugger Manual*
- *OpenVMS AXP Device Support: Developer's Guide*
- *OpenVMS Linker Utility Manual*
- *OpenVMS National Character Set Utility Manual*
- *OpenVMS VAX Patch Utility Manual*
- *OpenVMS VAX System Dump Analyzer Utility Manual*
- *OpenVMS AXP System Dump Analyzer Utility Manual*
- *OpenVMS SUMSLP Utility Manual*
- *OpenVMS System Services Reference Manual*
- *OpenVMS Utility Routines Manual*
- The documentation set for your programming language or optional products

Conventions

In this manual, every use of OpenVMS AXP means the OpenVMS AXP operating system, every use of OpenVMS VAX means the OpenVMS VAX operating system, and every use of OpenVMS means both the OpenVMS AXP operating system and the OpenVMS VAX operating system.

The following conventions are used to identify information specific to OpenVMS AXP or to OpenVMS VAX:

	The AXP icon denotes the beginning of information specific to OpenVMS AXP.
	The VAX icon denotes the beginning of information specific to OpenVMS VAX.
♦	The diamond symbol denotes the end of a section of information specific to OpenVMS AXP or to OpenVMS VAX.

The following conventions are also 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.

. . .	Horizontal ellipsis points 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.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason (user action that triggers a callback). Boldface text is also used to show user input in Bookreader versions of the manual.
italic text	Italic text emphasizes important information and indicates complete titles of manuals and variables. Variables include information that varies in system messages (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.

1

Introduction to the OpenVMS Programming Environment

The OpenVMS operating system provides a rich and varied environment for developing software application programs. Programming software that is included in the OpenVMS operating system and a wide range of optional tools offer a comprehensive environment for building software applications. By using the right tool for the right job, programmers at all levels can enhance productivity, improve software quality, and manage complex programming tasks.

This book can help you choose the right tool for the programming task at hand. It briefly describes the features of OpenVMS tools and points to other manuals for additional information about using them.

This chapter provides introductions to:

- Built-in and optional OpenVMS programming tools
- Portable and interoperable applications support
- Distributed computing support, including the client/server style
- Object-oriented design support
- Database and transaction processing support
- Specialized development environments
- Migration tools for moving OpenVMS VAX applications to OpenVMS AXP

1.1 Built-in and Optional OpenVMS Programming Tools

OpenVMS programming tools are available for performing the following tasks:

- Creating source files
- Managing software development tasks
- Compiling, linking, and debugging programs
- Accessing libraries of prewritten and debugged routines

Some OpenVMS programming tools are built into the operating system while others are optional tools that are separately orderable. Tools built into the OpenVMS operating system include text processors, assemblers, linkers, debuggers, utilities, run-time libraries, system services, and other callable system routines. Optional tools include compilers, interpreters, and project management tools.

Table 1–1 lists the software development tools supported by the OpenVMS operating system, shows whether they are built-in or optional, and provides pointers to their descriptions in this manual.

Table 1–1 OpenVMS Software Development Tools

Task	Name	Built-in or Optional	Where Described
Creating source files	DEC Text Processing Utility	Built-in	Chapter 5
	EVE editor	Built-in	
	EDT editor	Built-in	
	TECO editor	Built-in	
	vi editor (POSIX)	Optional	
	DEC Language-Sensitive Editor/Source Code Analyzer (LSE/SCA)	Optional	
Managing code and modules	DEC Code Management System (CMS)	Optional	Chapter 6
	DEC Module Management System (MMS)	Optional	
Creating object files	Compilers and interpreters[1]	Optional	Chapter 7
	†VAX MACRO assembler	Built-in	
	‡MACRO-64 assembler	Optional	
Linking	Linker utility	Built-in	Chapter 8
	Librarian utility	Built-in	
Debugging and testing programs	Symbolic debugger	Built-in	Chapter 9
	Delta/XDelta Debugger	Built-in	
	‡OpenVMS AXP System-Code Debugger	Built-in	
	System Dump Analyzer (SDA)	Built-in	
	DEC Performance and Coverage Analyzer (PCA)	Optional	
	DEC Language-Sensitive Editor/Source Code Analyzer (LSE/SCA)	Optional	
	DEC Test Manager	Optional	
Using callable system routines	Run-time libraries	Built-in	Chapter 10
	System services	Built-in	
	Utility routines	Built-in	
	Record Management Services (OpenVMS RMS)	Built-in	

[1] Except the VAX MACRO-32 Compiler for OpenVMS AXP which is a built-in compiler

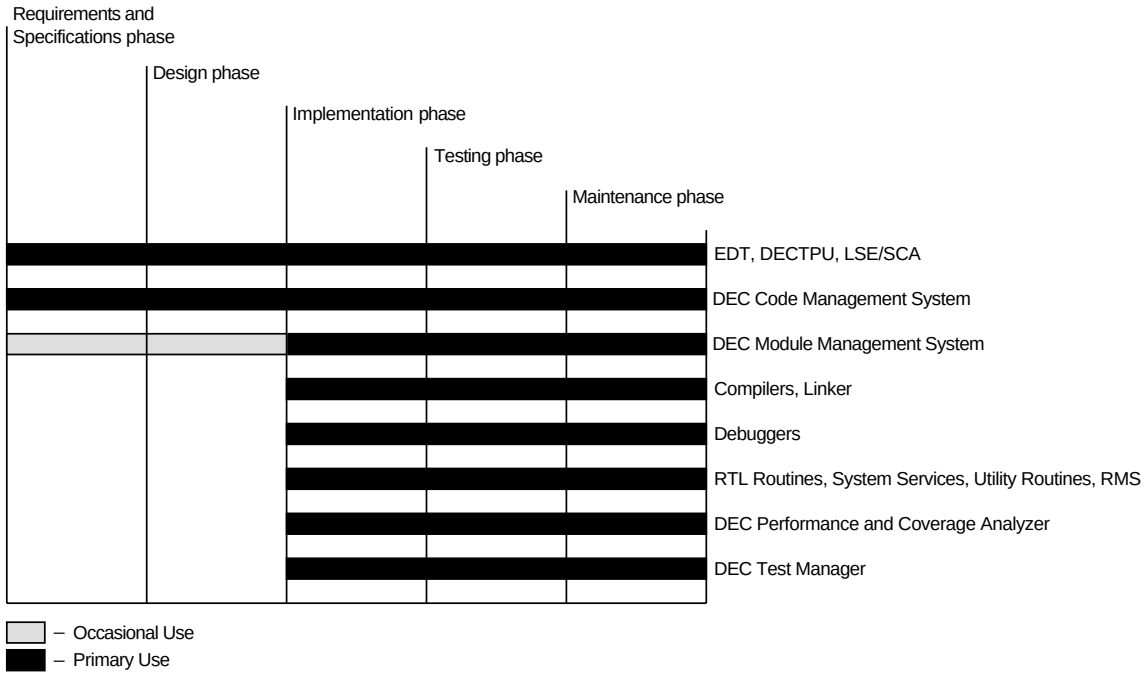
†VAX only

‡AXP only

Many tools can be used in more than one phase of the software development process. For example, you can use text editors throughout the development

process to create and modify source files. Figure 1–1 shows how some development tools work in more than one phase of the software development life cycle.

Figure 1–1 Software Development Phases and Tools



1.2 OpenVMS Support for Portable and Interoperable Applications

The OpenVMS operating system and related optional products support software that conforms to international standards for an open environment. These industry-accepted open standards specify interfaces and services that enable the creation of portable and interoperable applications. For more information on developing portable and interoperable applications, see Chapter 2.

1.3 OpenVMS Support for Distributed Applications

A distributed application must be able to coordinate its activities over a dispersed operating environment, two or more systems or processors, each with its own autonomous operating environment. The OpenVMS operating system supports the development of distributed applications through its support of the Open Systems Foundation Distributed Computing Environment (OSF DCE) and various networking products, such as DECnet for OpenVMS, DECnet/OSI, PATHWORKS, and DEC TCP/IP Services for OpenVMS. Chapter 3 provides an overview of the Digital products available for distributed programming.

1.4 OpenVMS Support for Object-Oriented Design

The OpenVMS operating system supports the object-oriented design of applications through its support of the C++ programming language and such Digital application-development products as ObjectBroker (formerly named DEC ACA Services) and DEC Forté. These products support object-oriented design but are not limited to that style. A major feature of ObjectBroker is that it facilitates the development of distributed applications. For more information about these products, see the references listed in Table 1–2.

Table 1–2 More Information on Products That Support Object-Oriented Design

For more information on...	Refer to...
ObjectBroker	Section 3.4.2
C++	Section 7.8
DEC Forté	Section 1.6

1.5 Database and Transaction Processing Support

OpenVMS systems support many Digital database and transaction processing (TP) products and similar products from other vendors.

1.5.1 Database Support

The Record Management Services (RMS), a subsystem of OpenVMS, is a collection of routines that give programmers a device-independent method for storing, retrieving, and modifying data. RMS routines are described in Section 10.5.

In addition to supporting many third-party database management systems, OpenVMS supports two Digital database management systems: DEC Rdb and DEC DBMS. DEC Rdb is a relational database management system and can be used for distributed database applications. DEC Rdb includes the Standard Query Language (SQL) and SQL Services.

Examples of the DEC Rdb product family and related products are shown in Table 1–3.

Table 1–3 Examples of DEC Rdb Products and Related Products

Product	Description
DEC RdbAccess for RMS	Enables transparent read and write access to OpenVMS RMS files
DEC RdbAccess for ORACLE on OpenVMS	Enables read-only access to ORACLE databases on local or remote OpenVMS systems
DECquery for MS–DOS	Enables PC users, who do not know SQL, to query a database
DEC RALLY	Provides a fourth-generation language environment for generating Rdb applications

DEC DBMS is a general-purpose, high-performance database management system. It is designed to handle high transaction volumes and is CODASYL-compliant.

Besides DEC Rdb and DEC DBMS, Digital also offers DEC ACCESSWORKS. DEC ACCESSWORKS enables application programs, running on various desktop computers, to access information in many database systems over PATHWORKS network connections. The database systems from which DEC ACCESSWORKS can retrieve information include the following:

- DEC Rdb
- RMS
- DEC DBMS
- ORACLE
- IBM DB2
- VSAM
- IMS

For more information about these products, see the *OpenVMS Software Overview*.

1.5.2 Transaction Processing Support

OpenVMS supports several Digital transaction processing products. DECtp is a transaction processing system that provides control and management of TP applications. DEC RTS (Reliable Transaction Monitor) is a distributed software message routing system that supports TP applications. ACMS is an application control and management system for developing, controlling, and maintaining transaction processing applications.

For more information about these products, see the *OpenVMS Software Overview*.

1.6 Specialized Development Environments

Digital offers several products that provide specialized development environments, such as DECADMIRE and DEC Forté. DECADMIRE enables developers to generate ACMS, DECforms, or DEC Rdb applications. DEC Forté is designed to address and manage the complexities of client/server computing. It provides tools for screen design, an object fourth-generation language, an object repository, an interpretive mode for testing, and graphical debugging.

For more information about these products, see the *OpenVMS Software Overview*.

1.7 Migration Tools and Documentation

Programs that run on OpenVMS VAX can be converted to run on OpenVMS AXP systems by recompiling and relinking them, by translating them, or by a combination of the two methods. A VAX MACRO compiler is provided with the OpenVMS AXP operating system for compiling VAX MACRO programs into AXP executable programs. The DEC compilers, such as DEC C and DEC Fortran, provide special options for recompiling VAX C and VAX FORTRAN applications to run on OpenVMS AXP.

DECmigrate for OpenVMS AXP, an optional product, is primarily used to translate OpenVMS VAX images to run on OpenVMS AXP. The major component of this product, the translator, is named the VAX Environment Software Translator (VEST). DECmigrate for OpenVMS AXP can also be used to analyze code to determine how easy or difficult it would be to migrate it.

In addition to these software products, documentation, training, and migration services are available. For more information, see the *OpenVMS Compatibility Between VAX and AXP* manual.

2
Portable and Interoperable Application Support

You can use OpenVMS programming tools to design portable applications, that is, applications that can be easily moved from one computer system to another. An example of a portable application is one that runs on an OpenVMS AXP system with POSIX for OpenVMS AXP installed and also runs on a Sun SPARCstation. (POSIX is the acronym for Portable Operating System Interface for UNIX.)

You can also use OpenVMS programming tools to design interoperable applications, that is, applications that can work with applications from other vendors, sharing data and other resources.

This chapter presents an introduction to the following topics:

- Application portability and interoperability
- OpenVMS support of standards
- DECwindows Motif programming support
- POSIX programming support
- Database interface with Structured Query Language (SQL)
- Industry standard 2D and 3D graphics support

For more information about these topics, see the *OpenVMS Software Overview*.

2.1 Application Portability and Interoperability

To achieve portability and interoperability, applications must be developed using programming interfaces, programming languages, routines, and tools that are supported by formal standards. Modular programming techniques can contribute to portability and interoperability. Platform-specific features such as run-time services, file formats, and uncommon language extensions must be avoided.

Furthermore, for applications to be portable, the target platforms must support the same standards. In order for data to be portable and interoperable, the target applications must support the same standards.

Portable applications written strictly to a suite of open specifications provide the following benefits:

- Applications can be written once and run on other open platforms that support the standards used in the applications.
- Applications are vendor independent.
- Application maintenance is less costly.

The following software specifications, supported by OpenVMS, contribute significantly to the creation of portable applications:

- Languages that conform to the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) standards
- OSF/Motif graphical user interface
- Structured Query Language (SQL)

2.2 OpenVMS Support of Standards

OpenVMS supports a broad spectrum of national and international standards, draft standards, and specifications. Some of the most significant standards that OpenVMS VAX and OpenVMS AXP support are shown in Table 2–1. For a comprehensive list of such standards, see the *OpenVMS Software Overview*. For a comprehensive list of the standards that each Digital product supports, see its software product description (SPD).

Table 2–1 Selected Standards Supported by OpenVMS VAX and OpenVMS AXP

Technical Area	Standards Body /Originator	Standard/Specification
Languages	ANSI, MIL-STD., ISO	Ada
	ANSI, ISO	BASIC
	ANSI, ISO	C
	ANSI	C++
	ANSI, ISO	COBOL
	ANSI, ISO	FORTRAN
	ANSI, ISO	Pascal
User Interfaces	MIT X Consortium[1]	X Window System
	OSF[2]	OSF/Motif
	IEEE[3]	POSIX P1003.2
	X/Open Company Ltd.	XPG3[4] BASE
Operating System Interfaces	IEEE, NIST[5]	POSIX 1003.1, 1003.1b, FIPS 151-1
	X/Open Company Ltd.	XPG3 BASE
Database	ANSI, ISO	SQL, CODASYL
Graphics Interfaces	ISO	ISO GKS
		ISO GKS 3–D
		ISO PHIGS
		PEXlib
Distributed Applications	OSF	Distributed Computing Environment (DCE)

[1]Massachusetts Institute of Technology X Consortium

[2]Open Software Foundation

[3]Institute of Electrical and Electronics Engineers, Inc.

[4]X/Open Portability Guide Issue 3

[5]National Institute of Standards and Technology

(continued on next page)

Table 2–1 (Cont.) Selected Standards Supported by OpenVMS VAX and OpenVMS AXP

Technical Area	Standards Body /Originator	Standard/Specification
Networking and Communication	CCITT[6], ISO	X.400
	ANSI	SCSI
	IEEE	802
	ISO	8802 (CSMA/CD)
	CCITT	X.25
	ISO	FTAM
	ARPANET[7]	TCP/IP
	Sun Microsystems' ONC[8]	NFS

[6]International Telegraph and Telephone Consultative Committee

[7]ARPANET Networking Group

[8]Open Network Computing Architecture (NFS published as RFC 1094)

2.3 DECwindows Motif Programming Support

The DECwindows Motif for OpenVMS environment provides a consistent user interface for developing software applications and includes an extensive set of programming libraries and tools. You can use the following DECwindows Motif software to build a graphical user interface:

- A user interface toolkit composed on graphical user interface objects (widgets and gadgets); widgets provide advanced programming capabilities that permit users to create graphic applications easily; gadgets, similar to widgets, require less memory to create labels, buttons, and separators.
- A user interface language to describe visual aspects of objects (menus, labels, and forms) and to specify changes resulting from user interaction.
- The OSF/Motif Window Manager, which allows users to customize the interface.

The DECwindows Motif programming libraries provided include:

- Standard X Window System libraries such as Xlib and the Intrinsics
- Libraries needed to support the earlier base of XUI applications
- OSF/Motif toolkit support for developing applications using the Motif user interface style
- Digital added-value libraries that give users capabilities beyond the standards

2.3.1 Linking and Navigation Capabilities with DEClinks

DEClinks services are included in the DECwindows Motif environment. They are used for creating, managing, and traversing informational links between different application-specific data. (DEClinks was formerly named LinkWorks.)

An application is said to be a **hyperapplication** if it participates in a DECwindows DEClinks environment. Hyperapplications provide linking and navigation capabilities to application end users through a new Link menu.

You can design new applications with DEClinks support or add DEClinks support to existing applications. DEClinks services, with the DEClinks Manager application, help organize information into a hyperinformation environment.

2.4 POSIX Programming Support

POSIX for OpenVMS offers VAX and AXP users the capability to develop and run open, portable applications on the OpenVMS operating system. Applications written to POSIX standards are portable across a wide range of UNIX and other operating systems that support those same standards. Application developers can develop and deploy their applications on any POSIX conformant system, including VAX and AXP systems.

Most applications that strictly conform to the POSIX and X/Open standards and draft standards can be developed on an OpenVMS VAX system with POSIX for OpenVMS VAX or an OpenVMS AXP system with POSIX for OpenVMS AXP, and then ported without modification to any other platform that supports the same POSIX and X/Open standards and draft standards.

The converse is also true, that is, applications developed on platforms other than OpenVMS systems that strictly conform to the POSIX and X/Open standards and draft standards can be ported and run on an OpenVMS VAX or an OpenVMS AXP system on which OpenVMS POSIX is installed. OpenVMS POSIX conforms to IEEE Standard 1003.1-1990,XPG-3, and FIPS 151-1. It is also compliant with drafts of 1003.2 (POSIX shell interface) and 1003.4 (real-time programming interface).

OpenVMS POSIX application programs are written using the C language and functions defined by the POSIX and X/Open standards and draft standards.

OpenVMS POSIX supports the POSIX 1003.1 standard for C bindings, which incorporates ANSI C and includes a series of system services. POSIX system services supported by OpenVMS POSIX include:

- Process creation, execution, and termination functions
- Process environment functions
- A series of POSIX functions that provide for file and directory operations
- I/O functions that include file I/O and creation of a pipe that serves as an interprocess channel
- Terminal interface functions involving mapping of a set of control character functions to sets of keys that are UNIX style or OpenVMS style
- Header files used for POSIX applications

The OpenVMS POSIX shell complies with a draft of POSIX 1003.2. The shell is based on the Korn shell, and it includes common UNIX commands and utilities, including *make* and *c89* (the POSIX interface to the compiler and linker, which is analogous to the *cc* command in UNIX).

OpenVMS POSIX implements the POSIX 1003.4 draft standard, which defines a set of real-time functions. For applications that have real-time computing requirements, these extensions provide support for such functions as enhanced interprocess communication, scheduling and memory management control, and asynchronous I/O operations.

For more information about OpenVMS POSIX programming interfaces, see the *Guide to Programming with VMS POSIX*, the *Guide to Programming with POSIX for OpenVMS AXP*, and other documentation in the OpenVMS POSIX documentation set.

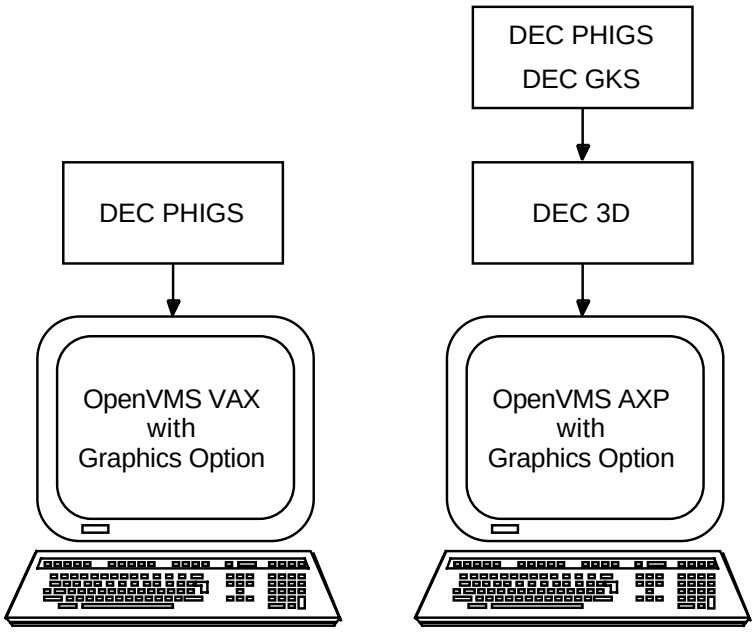
2.5 Database Interfaces with SQL

Applications need a standard way to interface with the variety of databases available in open environments. The data language of choice for applications is SQL, the ISO and ANSI language standard that OpenVMS supports through the DEC Rdb database product. Applications based on relational database management systems can use SQL as the language for defining, updating, and querying in the database.

2.6 Industry Standard 2D and 3D Graphics Support

Digital offers several optional products for developing two dimensional (2D) and three dimensional (3D) graphics applications: DEC Open3D, DEC PHIGS (Programmers Hierarchical Interactive Graphics System), and DEC GKS (Graphic Kernel System), as shown in Figure 2–1.

Figure 2–1 Graphics Applications Development Support



ZK–6832A–GE

Each product is a subroutine library, packaged as a set of shareable images against which an application program is linked. The shareable images are activated at run-time as needed. DEC Open3D also provides drivers for the

graphics options on OpenVMS AXP systems and is required for performing any 3D graphics operations on such systems.

2.6.1 DEC Open3D (AXP Only)

AXP

DEC Open3D for OpenVMS AXP provides support for the PXG family of graphic accelerators on Digital's Alpha AXP workstations. It also provides an extensive set of programming libraries for use by developers of new applications.

DEC Open3D is required to support any of the graphics options that are available for OpenVMS AXP, as shown in the following list:

- PXG
- ZLX-M1
- ZLX-M2 (also known as PixelVision)
- ZLX-E1 (also known as HX^{+})

  The ZLX-E1 is a 2D graphics option. Beginning with OpenVMS AXP Version 6.1, support for this graphics option will be included with the operating system, and DEC Open3D will no longer be required for 2D operations.

In conjunction with DECwindows Motif for OpenVMS AXP, DEC Open3D for OpenVMS AXP provides a complete run-time environment for 2D and 3D applications. DEC Open3D supports the MIT X Window System client/server model for network transparent graphics and windowing. The X11 DECwindows server provided with DEC Open3D will display output from client 2D applications supporting the MIT X Window System Version 11 Release 5 (X11R5) and client 3D applications supporting the PEX 3D extension to the X Window System.

In addition to providing an X and PEX compliant server for PXG series graphics accelerators, DEC Open3D includes Digital's implementation of PEXlib, a low-level programming interface to the PEX protocol. PEXlib enables programmers to develop and run applications on any platform supported by DEC Open3D and display the results on graphics devices which support the PEX Version 5.1 protocol.

Together, DECwindows Motif and DEC Open3D provide a robust environment for developers creating interactive applications that require both 2D and 3D graphics.♦

2.6.2 DEC PHIGS

DEC PHIGS for OpenVMS VAX and DEC PHIGS for OpenVMS AXP are 3D graphics support systems that control the definition, modification, and display of hierarchical graphics data. They manage the organization and display of graphical data stored in a conceptually centralized database.

DEC PHIGS for OpenVMS VAX and DEC PHIGS for OpenVMS AXP are device independent. That is, a program developed with one of these products can generate graphical output on different devices without modification to the source code. These products are Digital's implementations of the 1988 ANSI/ISO PHIGS standard for 3D device-independent graphics.

AXP

On OpenVMS AXP systems, DEC Open3D is required for using DEC PHIGS.♦

2.6.3 DEC GKS (AXP Only)

AXP

DEC GKS for OpenVMS AXP is a 3D graphics support system that provides application programmers with a set of functions for interactive and noninteractive computer graphics applications. You use the functions to define and display computer-generated 3D graphics using a variety of computer graphics equipment. As a development tool, it provides a base for portable, device-independent, 2D and 3D graphics application development.

DEC GKS for OpenVMS AXP implements the International Standard ISO 8805, the Graphical Kernel System for Three Dimension (GKS–3D). It conforms to level 2c of ISO 8805.

DEC Open3D is required for using DEC GKS for OpenVMS AXP.♦

Note

Although DEC GKS was supported on OpenVMS VAX Version 6.0 and earlier versions, it is not supported on OpenVMS VAX Version 6.1.

3
Distributed Computing Support

The OpenVMS operating system supports distributed computing, including the development and execution of distributed applications, with many built-in services and many optional Digital products.

This chapter presents an introduction to the following topics:

- Distributed computing
- OpenVMS networking support for distributed computing
- OpenVMS client/server support, including that provided by PATHWORKS and DECwindows Motif
- Distributed application support, including Digital DCE and ObjectBroker

For more information about these topics, see the *OpenVMS Software Overview*.

3.1 Distributed Computing

Distributed computing refers to the sharing of resources between two or more processors. The processors can be located in one computer, or, as more often the case, in separate computers. The resources to be shared include peripherals such as disks and printers, data, applications, and software to manage remote computers.

A distributed computing system that connects all parts of an enterprise can function as though it were a single system. A user can have transparent access to the integrated resources of the enterprise. A predominant design for distributed computing is client/server computing, and a predominant form of client/server computing is the use of larger computers as servers to personal computers (PCs).

3.2 OpenVMS Networking Support for Distributed Computing

OpenVMS networking support for distributed processing is provided by DECnet software and hardware and by the DEC TCP/IP Services for OpenVMS product.

The DECnet family of communication products (software and hardware) allows the OpenVMS operating system to participate in the DECnet network. DECnet/OSI for OpenVMS VAX and DECnet/OSI for OpenVMS AXP are based on the Open Systems Interconnection (OSI) model.

Users of DECnet/OSI for OpenVMS can choose between OSI networking protocols and the Digital Network Architecture (DNA) networking protocols, which can run simultaneously. OSI protocols permit communication with other vendors' systems that support OSI. The DNA protocols are the traditional networking protocols that permit communication with other systems supporting compatible versions of the DNA protocols.

DECnet/OSI for OpenVMS is compatible with DECnet for OpenVMS, a Phase IV product. Nodes running DECnet/OSI for OpenVMS and nodes running DECnet for OpenVMS can communicate on the same network.

Other DECnet products provide services for distributed processing over the network. These products include DECdts, a distributed time service; DECdns, a distributed name service; and DECdfs, a distributed file service.

DEC TCP/IP Services for OpenVMS enable TCP/IP connections and supply Network File System (NFS) server capabilities. TCP/IP is the protocol used by UNIX systems on the Internet. NFS permits UNIX clients to access OpenVMS files and UNIX files stored on the OpenVMS system.

DEC TCP/IP Services for OpenVMS permit an OpenVMS system to become a full participant in TCP/IP networks. The product includes the following:

- Set of industry-standard communication protocols (TCP, IP, FTP, Telnet, and other protocols)
- Digital Remote Procedure Call (DECrpc) for OpenVMS
- NFS server software

3.3 OpenVMS Client/Server Capabilities

One style of distributed computing that permits resource sharing between different systems is client/server computing. In the client/server environment, portions of an application are distributed across the network between servers and clients. The **server** is any system that provides a service or resource to other systems. The **client** is a system requesting a service. This style of computing allows each portion of a distributed application to run in its own optimal environment.

VMScluster software and Network Application Software (NAS) support client/server computing. VMScluster software enables client/server computing among OpenVMS VAX and OpenVMS AXP systems. NAS software supports client/server computing among OpenVMS VAX, OpenVMS AXP, and DEC OSF/1 AXP systems, and among Digital systems and other vendors' systems including UNIX, MS–DOS, Microsoft Windows, and Windows NT systems. PATHWORKS software, which is part of NAS, enables complex information-sharing environments involving PC clients and operating system servers. For more information about PATHWORKS products, see Section 3.3.4.

3.3.1 OpenVMS Client/Server Configurations

OpenVMS systems support a wide variety of client/server configurations.

A single OpenVMS system or a VMScluster can function as a server. OpenVMS servers can provide file access, printing, application services, communication services, and computing power as application engines to clients on desktop devices or in laboratories or factories.

Clients requiring resources can be any of the following:

- Personal computers
- Workstations
- Point-of-sale devices

- OpenVMS systems
- Other vendor systems that are running the client software

User interfaces on client systems can be character-cell terminals or windowing desktops.

Client/server configurations permit the enterprise-wide capabilities of OpenVMS host systems to be integrated with the personal-computing capabilities of desktop systems.

3.3.2 VMScluster Servers and OpenVMS Clients

In any VMScluster system, users can share computing, disk and tape storage, and batch and print processing resources. Any node in the cluster can use clusterwide batch and print queues. VMScluster technology also allows cooperating OpenVMS systems to share file and print resources over a LAN. This capability provides a mechanism for offering print and computing services to the network.

In a VMScluster configuration, the mass storage control protocol server and the tape mass storage control protocol server make locally connected disks and tapes available across the cluster.

A VMScluster system can serve files and databases to the LAN for use by cluster members. In addition, databases can be offered over a VMScluster using Rdb/VMS with Structured Query Language (SQL) Services. These services enable a client OpenVMS system to issue SQL requests to a VMScluster system that serves its Rdb/VMS databases to the LAN. The Rdb/VMS request is processed on the database server and the resulting information is sent to the SQL Services client.

3.3.3 Client/Server Features of DECwindows Motif

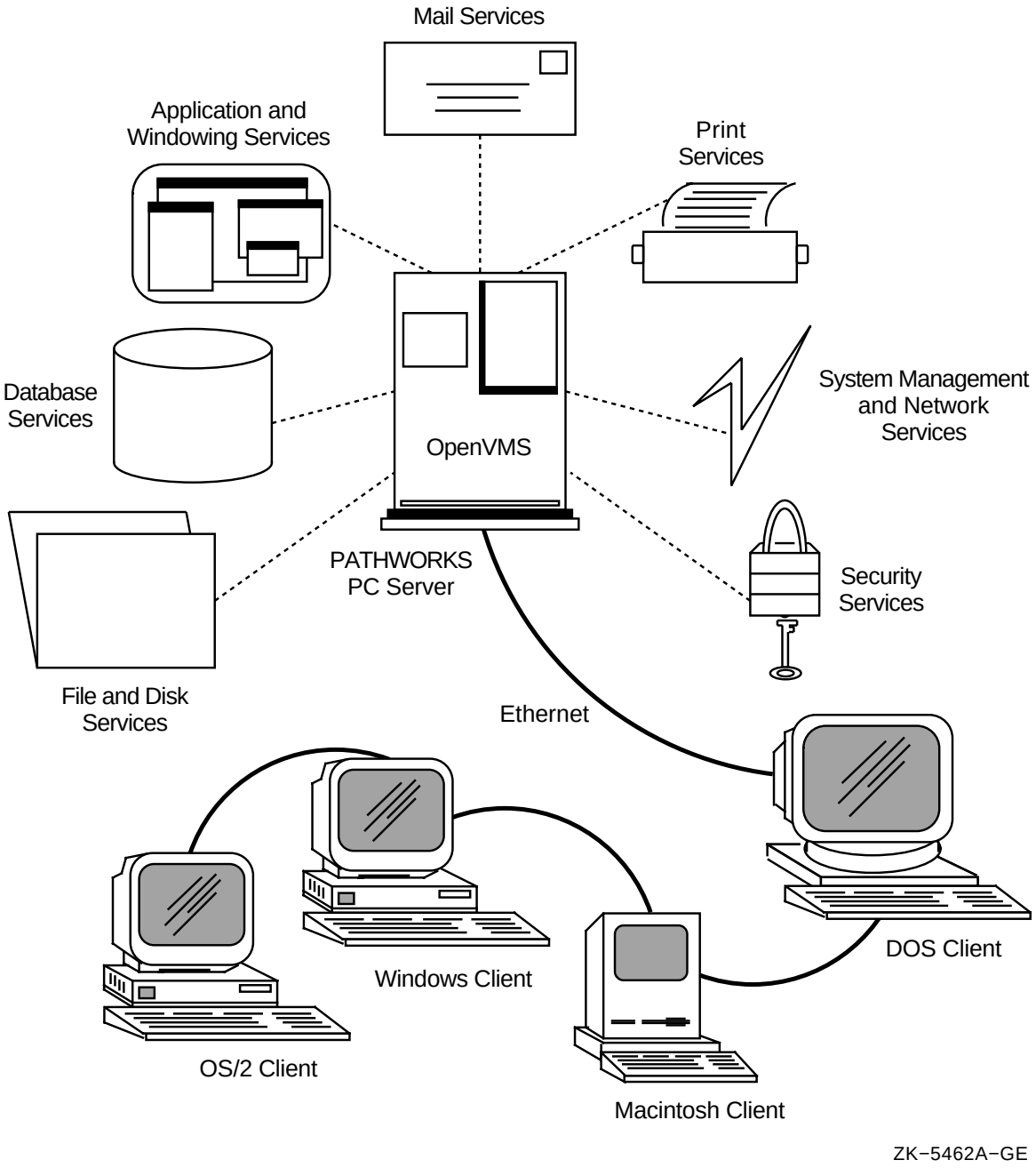
DECwindows Motif, based on industry-standard OSF/MOTIF, lets users access application programs running on other machines in the network as if the applications were running locally. With the DECwindows software, multiple device-independent applications can run simultaneously in various separate workstation windows. Applications function as clients and the DECwindows program that responds to the applications is the DECwindows server. (The DECwindows Motif user interface is described in Section 2.3.)

3.3.4 OpenVMS Servers with PC Clients

The OpenVMS operating system running on a VAX or AXP computer can act as an application, file, disk, and print server for large groups of personal computer clients through DECnet and TCP/IP networking connections.

Servers and clients can also be connected using PATHWORKS products. You use PATHWORKS server software on an OpenVMS system and PATHWORKS client software on personal computers. PATHWORKS supports a variety of personal computers including MS–DOS, Windows, OS/2, and the Macintosh operating system, as illustrated in Figure 3–1. PATHWORKS server software is also available for ULTRIX, SCO UNIX, and OS/2 operating systems.

Figure 3–1 OpenVMS Services to Personal Computer Clients in a PATHWORKS Configuration



PATHWORKS enables PC users to share applications, data, and system resources such as printers, disks, CD–ROM readers, and network gateways, without losing the benefits of industry-standard personal computing.

PATHWORKS for OpenVMS enables the OpenVMS server to provide the following kinds of services (shown in Figure 3–1) to PC clients:

- File, disk, and print services
- Database services
- Electronic mail services
- Application and windowing services
- System management and network services
- Security services

3.3.5 PATHWORKS Configurations

PATHWORKS configurations are flexible and can be changed easily. In most cases, PATHWORKS client software is originally stored on the server and downline loaded over the network to the PC when the PC is booted. The PC user can connect to a particular PATHWORKS server. PC clients can be connected to multiple servers at the same time.

Configuration changes, such as the addition of new PATHWORKS clients and servers, do not cause disruption to the PC user environment. PC users can continue to use the applications they prefer.

3.4 Distributed Application Support

A distributed application consists of separate modules, running on different processors, that communicate with each other by passing data between modules or by sharing access to files or databases. The processors in the distributed configuration can be uniprocessor, multiprocessor, or VMScluster systems; systems from different vendors can be included in the same configuration.

3.4.1 Support for the OSF Distributed Computing Environment

The OSF Distributed Computing Environment (DCE) is a standard set of software services and interfaces that support the creation, use, and maintenance of distributed client/server applications. Digital has implemented a family of DCE products that include a certified set of DCE functions along with software for developing distributed applications.

Digital's implementation of OSF DCE standards,[1] is adapted and enhanced for OpenVMS as follows:

- Simplified installation and configuration
- Interface Definition Language (IDL) support for both C and Fortran languages
- IDL development templates
- A conversion utility for DCE RPC programs
- The PC NSI Proxy Agent, which enables interoperability with Microsoft's PC RPC.

[1] OSF DCE Version 1.0.2 (without security replication)

The DCE for OpenVMS product family supports the following networking transports: TCP/IP, User Datagram Protocol (UDP), and DECnet Phase IV.

The DCE for OpenVMS products include:

- DCE Runtime Services for OpenVMS, required for all systems participating in the DCE cells. A DCE cell is the term used to describe the group of systems that participate in the distributed computing environment. The runtime services kit includes DCE client functions and DCE administration tools.
- DCE Application Development Kit for OpenVMS, required for those developing distributed applications but optional for other users. The kit provides users with an Interface Definition Language (IDL) for writing remote procedure calls.

Digital is committed to support the following DCE products on OpenVMS:

- DCE Cell Directory Server (CDS), a central repository containing information about the location of resources in the DCE cell. The CDS server allows access to resources by a single name, regardless of physical location. One CDS server is required for each DCE cell.
- DCE Security Server, which protects resources from illegal access by providing authentication and authorization services and provides for secure communications within and between DCE cells. One Security server is required for each DCE cell.

3.4.2 ObjectBroker

ObjectBroker, formerly named DEC ACA Services, enables developers to use an object-oriented design to integrate independently-developed applications across heterogeneous environments. ObjectBroker facilitates the transition to client/server computing and reduces the cost of developing new client/server applications. ObjectBroker is available for OpenVMS VAX and OpenVMS AXP systems.

ObjectBroker complies with the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) and provides both dynamic and static interfaces for greater user flexibility.

ObjectBroker extends the Microsoft Dynamic Data Exchange (DDE) communication protocol to allow Microsoft Windows applications on a networked PC to interact, using DDE, with applications running on ULTRIX, OpenVMS, and SunOS. ULTRIX, OpenVMS, and SunOS applications can function as DDE clients and servers. DDE support in ObjectBroker allows two Microsoft applications to communicate on different PCs.

4

User Interface Tools for OpenVMS Applications

This chapter describes the Digital tools available for creating user interfaces for applications that run only on OpenVMS systems. Digital tools for creating user interfaces for portable applications, including graphical user interface tools, are described in Chapter 2.

The following tools are described in this chapter:

- DIGITAL Command Language (DCL)
- Command Definition utility
- Message utility
- DECforms

4.1 DIGITAL Command Language

DIGITAL Command Language (DCL) commands can be used to invoke program development software (compilers, editors, and linkers) and to run and control the execution of programs. DCL command procedures can be used to perform repetitive operations in software development. For more information, see the documentation listed in Table 4–1.

Table 4–1 More Information on DCL Commands and Their Use

For more information on...	Refer to...
DCL commands, qualifiers, and options	*OpenVMS DCL Dictionary*
How to use DCL Commands	*OpenVMS User's Manual*

4.2 Command Definition Utility

The Command Definition utility (CDU) enables application developers to create commands with a syntax similar to OpenVMS DCL commands that are executed at the DCL level. Using CDU, the developer can create applications with user interfaces similar to those of operating system applications.

The Command Definition utility (CDU) creates, deletes, or changes command definitions in a command table. Command tables are data structures created by the CDU and used by the Command Language Interpreter (CLI) to parse and evaluate DCL commands.

There are two types of command tables: system command tables used to parse system commands and process command tables used to parse process-specific commands. The CDU creates command tables from command definition files, from existing command tables, or from a combination of these sources. The new tables can be either executable code or object modules.

You can modify your process command table, the system command table in SYS$LIBRARY, or create a new command table to be used with user-written applications.

For more information about creating your own commands with CDU, refer to the *OpenVMS Command Definition, Librarian, and Message Utilities Manual*.

4.3 Message Utility

The Message utility (MESSAGE) allows you to supplement OpenVMS system messages with your own messages to signal any condition—error or success. Use an editor to create a message source file, which consists of message definition statements and directives that define the message text, the message code values, and the message symbol. With these directives, you can assign severity levels, specify message text, and define the facility to which the message relates.

After compiling your message source file using the Message utility, you link the message object module with the program object module. By using message pointers, you can use different text for the same message. This option is particularly useful for multilingual applications. To use pointers, you create a nonexecutable message file that contains the message text and a pointer file that contains the symbols and pointer to the nonexecutable file.

For complete information about creating your own messages, refer to the *OpenVMS Command Definition, Librarian, and Message Utilities Manual*.

You can also make descriptions of your messages available from the DCL prompt ($) by including message descriptions in the Help Message database. For information about linking your messages into the system and adding them to the Help Message database, see *OpenVMS System Messages: Companion Guide for Help Message Users*. This manual provides complete information about using the Help Message utility to create and access online message descriptions; it also includes basic information about message formats, severity levels, and recovery procedures.

4.4 DECforms

DECforms is a set of development tools and services for developing user interfaces. DECforms lets programmers create forms and menus quickly and efficiently. With DECforms, the form remains independent of both the application program and the display device. This permits either to be changed without affecting the other.

DECforms also permits a single application to support multiple devices as well as multiple users who speak different languages. At run-time, DECforms manages communication among the form, the display device, and the application program. It selects the appropriate form and language independent of the application.

DECforms is the industry's first implementation of the ANSI/ISO standard for a Forms Interface Management System (FIMS). When other vendors implement the FIMS standard, DECforms will be considered a tool for creating a portable interface.

5
Editors

This chapter describes features of the following Digital editors:

- DEC Text Processing Utility (DECTPU) and its Extensible Versatile Editor (EVE)
- EDT editor
- DEC Language-Sensitive Editor/Source Code Analyzer (LSE/SCA)
- SUMSLP utility

In addition to the editors described in this chapter, another editor, the vi editor, is supplied with the POSIX interface. For information about the vi editor on OpenVMS AXP, see the *Guide to Programming with POSIX for OpenVMS AXP* and the *POSIX for OpenVMS AXP Reference Manual: Shell and Utilities*. For information about the vi editor on OpenVMS VAX, see the *Guide to Programming with VMS POSIX* and the *VMS POSIX Reference Manual: Shell and Utilities*.

This broad selection of editors enables you to choose an editor that matches your preferences and the type of work you want to do.

5.1 DEC Text Processing Utility

DECTPU is a high-performance, text processor that you can use to create text-editing interfaces such as EVE. DECTPU has the following features:

- A high-level procedure language with several data types, relational operators, error interception, looping, case language statements, and built-in procedures
- A compiler for the DECTPU procedure language
- An interpreter for the DECTPU procedure language
- The EVE editing interface, which, in addition to the EVE keypad, provides EDT, VT100, WPS, and numeric keypad emulation

With these tools, you can further customize the EVE editing interface or create your own editing interface designed for your programming needs.

Special features offered with DECTPU include the following:

- Multiple buffers
- Multiple windows
- Multiple subprocesses
- Text processing in batch mode
- Insert or overstrike text entry
- Free or bound cursor motion
- Learn sequences

- Pattern matching
- Key definition

For more information about using DECTPU, refer to the *Guide to the DEC Text Processing Utility* and the *DEC Text Processing Utility Reference Manual*.

5.1.1 EVE

For most uses, the EVE editing interface is preferable to EDT because of its features and customizable interface. The EVE editing interface is installed with DECTPU.

EVE is easy to learn and easy to use. You can access most common editing functions by pressing a single key on the EVE keypad. You invoke EVE commands and special DECTPU features and advance functions by entering commands on the EVE command line. With EVE, you can customize your editing interface by using initialization files, command files, learn sequences, key definitions, and DECTPU built-in procedures.

If you are an experienced EDT user, you can use the EVE command SET KEYPAD EDT to redefine the default EVE keypad bindings to emulate an EDT keypad. EDT keypad emulation in EVE provides most of the functions of the EDT keypad and binds these functions to the same keys that EDT uses.

5.2 EDT Editor

EDT is an interactive text editor that has the following capabilities:

- Three types of editing modes: keypad mode for screen-oriented editing, line mode for line-number editing, and nokeypad mode for defining your own key sequences. You can use any mode you prefer and you can switch back and forth during a single editing session.
- Journaling to protect your editing session in the event of a system interruption.
- Multiple buffers.
- Access to as many files as you need.
- Startup command files to initialize the EDT editing environment to your own needs.
- EDT macros to automate repetitive editing procedures.

5.3 DEC Language-Sensitive Editor/Source Code Analyzer

The DEC Language-Sensitive Editor/Source Code Analyzer (LSE/SCA) is a multilanguage, multiwindow, screen-oriented editor and a source code analyzer, designed for program development and maintenance. LSE/SCA is a component of DECset, an optional Digital product. Each DECset tool provides a DECwindows Motif user interface and a consistent look and feel across platforms. LSE/SCA works in concert with supported OpenVMS languages and the OpenVMS Debugger to provide a highly interactive, online environment for editing, compiling, debugging, and analyzing a program.

LSE provides the following features:

- Contains source code templates for the language constructs it supports
- Tailors the editing sessions for supported languages and products

- Uses source code templates
- Allows coding, compiling, reviewing, and correcting of compile-time errors without leaving the editing session
- Provides interactive editing capabilities during a debugging session
- Allows programmers to tailor the defined language environments or to define their own environment
- Provides integrated access to the cross-referencing features of the DEC Source Code Analyzer (SCA)

SCA allows interactive inquiries about program structure, including cross-reference information, calling structure, and where and how often different program elements are used. It also performs static error checks (for example, checking the number and type of arguments passed).

For more information about DEC LSE/SCA, see the *Guide to DEC Language-Sensitive Editor and DEC Source Code Analyzer*.

5.4 SUMSLP Utility

The SUMSLP utility (SUMSLP) is a batch-oriented editor that is useful when you need to make several updates to a single file. To use it, you create a series of editing commands to add, delete, or update lines in the file. The editing commands are specific to SUMSLP and can be used only by SUMSLP. It can be useful if you are maintaining several copies of a single file, because it allows you to update the file by creating one update program and applying the update program to each copy of the file.

SUMSLP requires at least the following input files:

- The source file to be updated. Because you use line-oriented editing commands, you should generate a sequence-oriented listing.
- The update file. This file contains SUMSLP command lines and the updated lines used to alter the input file.

SUMSLP applies the edits specified in the SUMSLP update file to the input source file. The SUMSLP output file generated is the updated source file.

The *OpenVMS SUMSLP Utility Manual* describes each of the SUMSLP commands and how SUMSLP processes files.

6

Tools for Managing Program Files

This chapter describes the following tools for managing program files:

- DEC Code Management System (CMS)
- DEC Module Management System (MMS)

Both of these tools are components of DECset. DECset is a multipurpose, multiplatform toolset with a DECwindows Motif user interface. This interface provides similar functions and a consistent look and feel across platforms. DECset supports software coding, testing, debugging, and maintenance activities for multiple languages.

6.1 DEC Code Management System

Code management is especially important on large projects with long life spans and several versions of the software. The DEC Code Management System (CMS) provides an efficient method for storing project files and tracking all changes to those files.

CMS stores any kind of file, including files created by an editor, compiler, or a linker. You can use CMS to store documents (for example, plans, specifications, and status reports), object files, executable files, sixel files, or other records. (CMS cannot store directory files.)

You can use CMS to do the following:

- Keep track of files at every stage of development by showing who made changes, when, and why.
- Allow programming development team members to work concurrently on the same file without losing the changes made by any team member.
- Conserve disk space (CMS stores consecutive versions of files in a space-efficient manner.)
- Maintain a history of library activity.
- Store files created by other software development tools, such as DEC Test Manager.
- Mark an element generation for review to indicate that its contents should be reviewed by other users.
- Automatically copy the latest generation of a CMS library into a reference copy area.

CMS keeps your files in project libraries, which are OpenVMS directories. These directories store your project's files, or elements, as well as history information. A CMS library provides a record of the following:

- Transactions that created specific element generations.
- Transactions related to the evolution of a specific element.
- The entire transaction history of the library; that is, all actions that create, delete, or modify the library or its contents.

6.2 DEC Module Management System

DEC Module Management System (MMS) automates and simplifies the building of software applications by providing a consistent way to build modular software applications. MMS software builds a system faster because it builds only the parts that require building. MMS consistently reproduces the same system each time it is built, thereby increasing the accuracy of the build. No time is wasted recompiling and linking modules that have not changed since the previous system build. Once set up, MMS can build small and large systems with one command.

7

Compilers, Interpreters, and Assemblers

The OpenVMS operating system supports a variety of language compilers, interpreters, and two assemblers, one for VAX computers and one for AXP computers. The compilers whose names begin with VAX are available for developing applications on OpenVMS VAX systems. Most of the compilers whose names begin with DEC are available for developing applications on OpenVMS VAX systems as well as OpenVMS AXP systems.

Most OpenVMS programming languages use all of the resources of the OpenVMS operating system, and all of them can access any of the callable routines (system services, utility routines, run-time library routines, and record management services). Most OpenVMS languages are fully supported by the OpenVMS Debugger. VAX APL and VAX DIBOL have their own debugger utility. Note that most OpenVMS languages are optional software products.

This chapter describes:

- Common language environment
- Compilers, interpreters, and assemblers available on OpenVMS VAX and OpenVMS AXP systems

7.1 Common Language Environment

The OpenVMS operating system supports a common language environment that lets you develop mixed-language application programs and portable programs, including the use of distributed functions for client/server environments. For example, a program written in any of the programming languages supported by OpenVMS can contain calls to procedures written in other supported languages.

The common language environment applies to both VAX and AXP computers. For example, native AXP programs call native AXP programs written in other languages, and native VAX programs call native VAX programs written in other languages. The OpenVMS calling standard simplifies migrating mixed-language applications between OpenVMS VAX and OpenVMS AXP systems.

All languages supported by OpenVMS adhere to the OpenVMS calling standard, which describes the techniques used by all OpenVMS languages for invoking routines and passing data between them. The standard also defines the mechanisms that ensure consistency in error and exception handling routines, regardless of the mix of programming languages in use. For more information about OpenVMS data types and calling routines, see *OpenVMS Programming Interfaces: Calling a System Routine*. For complete information about the calling standard, see the *OpenVMS Calling Standard*.

7.2 Summary of Language Features

Table 7–1 lists the languages available for OpenVMS VAX and OpenVMS AXP and their main features. Compilers whose names begin with VAX are supported only on OpenVMS VAX. Compilers whose names begin with DEC are supported on OpenVMS VAX and OpenVMS AXP unless noted otherwise. The sections that follow provide more detail about each one.

Table 7–1 Compilers, Interpreters, and Assemblers

Language	Features
DEC Ada	Complete production-quality implementation of Ada language; fully conforms to ANSI and MIL-STD standards; has Ada validation
VAX APL	Interpreter with built-in editor, debugger, file system, communications facility
VAX BASIC	Supports a robust implementation of the BASIC language, containing most constructs found in traditional programming languages; can be used as either an interpreter or a compiler; fully supported by OpenVMS Debugger; fully reentrant code
DEC BASIC for OpenVMS AXP	Supports a robust implementation of the BASIC language, containing most constructs found in traditional programming languages; is an optimizing compiler which is highly compatible with VAX BASIC; no environment/interpreter support
VAX BLISS-32	Provides advanced set of language features supporting development of modular software according to structured programming concepts; provides access to most VAX hardware features
VAX C	Full implementation of C programming language with added features for performance enhancement in the OpenVMS environment
DEC C for OpenVMS AXP	Compliant with ANSI-standard Systems-Programming Language C (document number: X3.159-1989)
DEC C++	Includes class libraries, a new C run-time library, and support for the debugger and LSE/SCA
DEC COBOL for OpenVMS AXP	Based upon the 1985 ANSI COBOL Standard X3.23-1985 and is closely compatible with VAX COBOL
VAX COBOL	Compatible with ANSI-standard COBOL; supports an embedded data manipulation language interface to Digital's CODASYL-compliant Database Management System (DBMS)
VAX DIBOL	Designed for interactive data processing; includes a compiler, debugger, and utility programs for data handling, data storing, and interprogram communication
DEC Fortran for OpenVMS VAX	Supports ANSI-standard FORTRAN–77 with many industry-leading extensions; conforms to FIPS standards; has a high optimization compiler and takes full advantage of features of the OpenVMS environment

(continued on next page)

Table 7–1 (Cont.) Compilers, Interpreters, and Assemblers

Language	Features
DEC Fortran for OpenVMS AXP	Supports ANSI-standard FORTRAN–77, nearly all DEC Fortran for OpenVMS VAX extensions, and other language features including recursion
VAX MACRO	Assembly language for programming the VAX computer under the OpenVMS operating system; uses all OpenVMS resources; supports large instruction set enabling complex programming statements
VAX MACRO-32 Compiler for OpenVMS AXP	Available for porting existing VAX MACRO code to an AXP system
MACRO-64 for OpenVMS AXP	The AXP assembly language that provides precise control over instructions and data
DEC OPS5	A development environment (compiler, run-time library, and DECwindows Motif-based programming /debugging environment) for constructing high performance, forward chaining, rule-based applications
DEC Pascal	Supports standard ANSI Pascal features and added features using character instruction sets and OpenVMS virtual memory
DEC PL/I for OpenVMS AXP	Includes a compile-time preprocessor that allows language extension and conditional compilation
VAX PL/I	Includes a compile-time preprocessor that allows language extension and conditional compilation

If you are planning to move applications between OpenVMS VAX and OpenVMS AXP, see the documents *OpenVMS Compatibility Between VAX and AXP* and *Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications*. If you are planning move applications written in VAX MACRO, see *Migrating to an OpenVMS AXP System: Porting VAX MACRO Code*.

7.3 Ada

DEC Ada for the OpenVMS operating system is a complete implementation of the Ada programming language, a language which facilitates a portable, modular design. It conforms fully to the ANSI/MIL standard and is validated by NIST and the Ada Validation Facility. DEC Ada features include the following:

- The DEC Ada library manager allowing shared use of a compilation library, use of individual libraries as sublibraries of team libraries, and automatic recompilation of obsolete units.
- Individual units (subprograms, tasks, packages, generic units) that can be compiled separately.
- Strong typing to ensure the integrity of data types. Type checking is done at compile time.
- Data abstraction to free your programmer from needing to know specifically how DEC Ada implements data types, executable statements, and so forth.
- Ability to define system features (for example, memory size) that can limit program scope for each application.
- Use of tasks within the language to support parallel processing.

- Ada defined exception handling to recover from error conditions. User-defined exception handling is also available.

VAX

The DEC Ada Professional Development Option is available on OpenVMS VAX systems and is licensed separately. This option provides smart recompilation and program library file-block caching to increase productivity.♦

7.4 APL

The VAX APL interpreter provides a built-in editor, debugger, system communications facility, and file system. It automatically reserves space for variables, formats input and output statements, and manipulates rows and columns of data without loops. It can call another VAX APL program and have data returned as a result.

7.5 BASIC

Digital offers two highly compatible versions of BASIC, VAX BASIC for developing BASIC programs on OpenVMS VAX and DEC BASIC for developing BASIC programs on OpenVMS AXP. Both products support a robust implementation of the BASIC language, containing most constructs found in other traditional programming languages and access to common record definitions stored in the CDD/Repository (formerly named the Common Data Dictionary). Both products are fully supported by the OpenVMS operating system environment, including access to all utilities, the ability to invoke the callable system routines, and the ability to use object modules from other programming languages.

VAX BASIC and DEC BASIC support the following extensions to the traditional BASIC language:

- Multiple integer and floating-point data types, the packed decimal data type, and user defined records
- Structured programming statements, such as SELECT/CASE, IF/THEN /ELSE, FOR/WHILE/UNTIL loops, local (DEF) and external functions, and alphanumeric labels
- Sequential, relative, and indexed I/O through RMS
- Structured error handling
- Support for the OpenVMS Debugger
- Lexical compiler directives

VAX

VAX BASIC also provides an environment/intrepreter as well as a package of language statements for performing graphics operations.♦

7.6 BLISS-32

VAX BLISS-32 supports development of modular software according to structured programming concepts by providing an advanced set of language features. It provides access to most of the hardware features of a VAX system.

VAX BLISS-32 programs include the following features that allow programs to be transported to other Digital computer systems:

- High-level language constructs can be transferred from one machine to another with little or no alteration.

- Machine-specific functions can be separated from the common, mainline code via modularization, macros, and special Library and Require files (separate files that can be invoked from a BLISS program).
- Machine-specific characters can be passed to BLISS data structures with the use of parameters.

7.7 C

Digital offers two versions of C, VAX C and DEC C. VAX C is available on OpenVMS VAX systems only. DEC C is available on OpenVMS AXP systems and will soon be supported on OpenVMS VAX systems as well.

Both products are full implementations of the C language and both are fully supported by the OpenVMS operating system. This support includes access to all utilities, the ability to invoke the callable system routines, and the ability to use object modules from other programming languages.

DEC C is compliant with the ANSI-standard Systems-Programming Language C (document number: X3.159-1989). By using command-line qualifiers, DEC C is compatible with older dialects of C, including common usage C and VAX C.

VAX C and DEC C provide the following features within the OpenVMS operating system environment:

- Set of structured control flow operators
- Set of mathematical and logical operators
- Data typing and conversions
- Consistent data declarations and data references
- Compiler optimized code, along with listing and cross-referenced storage map
- Common set of run-time support routines for accomplishing common tasks such as I/O or math routines (many UNIX specific routines have been emulated)
- New keywords for sharing data among program modules to allow for easy reference to VAX MACRO programs and OpenVMS callable system routines
- Set of predefined macros to assist in transporting code and performing simple tasks
- Set of built-in functions to efficiently access processor instructions

7.8 C++

DEC C++, a language that supports an object-oriented program design, is supported on both OpenVMS VAX and OpenVMS AXP. DEC C++ supports the full language definition as specified in *The Annotated C++ Reference Manual* by Margaret Ellis and Bjarne Stroustrup (Addison-Wesley, 1990) excluding exception handling. Exception handling is an experimental language feature that is not currently implemented in DEC C++ but is considered for a future version.

DEC C++ supports LSE/SCA templates and LSE/SCA diagnostics and includes enhancements to the debugger support of the C++ language features, such as the OpenVMS Debugger command and DECwindows interfaces.

7.9 COBOL

Digital offers two versions of COBOL, VAX COBOL for developing COBOL programs on OpenVMS VAX and DEC COBOL for developing COBOL programs on OpenVMS AXP. Both are compatible with the ANSI-standard COBOL and both allow access to common record definitions stored in the Common Data Dictionary. They are fully supported by the OpenVMS operating system environment, including access to all utilities and the ability to invoke the callable system routines and to use object modules from other language programs.

VAX COBOL and DEC COBOL support the following features:

- Full report-writing capabilities
- Form and report creation on terminals, with screen handling
- Complete sequential, relative, and indexed I/O
- All data types for ANSI COBOL, plus packed decimal, floating point, double floating point, and address data types
- Structured programming statements such as EVALUATE for CASE-like statements, scope-delimited statements to reduce use of GOTO, and inline PERFORM statements

VAX

VAX COBOL also supports an embedded data manipulation language (DML) interface to Digital's CODASYL-compliant Database Management System. DEC COBOL does not.♦

7.10 DIBOL

VAX DIBOL is designed specifically for interactive data processing. It includes a compiler, a debugger, and a set of utility programs that facilitate data handling, data storing, and interprogram communication. VAX DIBOL can invoke OpenVMS Record Management Services (RMS), system services, utility routines, and run-time library routines. It can use object modules produced from any other VAX language program.

The VAX DIBOL compiler produces a source file listing, symbol table, label table, error report, error listing, and cross-reference listing. The VAX DIBOL Debugger Tool (DDT) allows you to examine or change program data at run time, to set breakpoints, and to examine the flow of execution. The other utilities include a VAX DIBOL Message Manager that stores and retrieves messages for VAX DIBOL programs and the VAX DIBOL Message Status that allows you to examine and delete any messages currently being held by the Message Manager.

7.11 Fortran

Digital offers DEC Fortran on both OpenVMS VAX and OpenVMS AXP systems. Both are fully supported by the OpenVMS operating system programming environment. Both provide a highly efficient, optimizing compiler that offer the following features:

- Full language support of ANSI-standard FORTRAN 77 as well as numerous FORTRAN 77 extensions. These extensions include record structures, recursion, extended-precision REAL*16 data, and other language features associated with VAX FORTRAN and DEC Fortran on other Digital platforms. Both also conform to FIPS-69-1, ISO 1539-1980(E) and MIL-STD 1753.

- Numerous OpenVMS features and layered products, including:
  - All OpenVMS (RMS) file formats
  - Access to object files created by other Digital languages and support of the OpenVMS calling standard
  - Creation of shareable images usable by any program written in a native Digital language
  - Invoking all callable system routines, including the use of the FORSYSDEF library definitions for calling system services
  - Conversion of unformatted nonnative floating-point data to the selected memory format, including several big endian data formats
  - Selection of the floating-point data type used in memory
  - Support for the CDD/Repository
  - Use of all OpenVMS programming utilities

VAX

DEC Fortran for OpenVMS VAX Systems (previously called VAX FORTRAN) also provides parallel processing support (both automatic and directed).♦

AXP

DEC Fortran for OpenVMS AXP Systems also provides data alignment control and the use of IEEE and most VAX floating-point data types in memory.♦

7.12 MACRO

The OpenVMS operating system supports the following assembly language products:

- VAX MACRO
- MACRO-32 Compiler for OpenVMS AXP
- MACRO-64 Assembler for OpenVMS AXP

7.12.1 VAX MACRO

VAX MACRO is an assembly language for programming a VAX computer under the OpenVMS operating system. The instruction set includes approximately 130 instructions and 70 directives, which enable complex programming statements. It can use all OpenVMS resources. For example, it can invoke any callable system routine, use the OpenVMS Debugger and other utilities, and call any object module written in another VAX language.

General assembler directives can perform the following operations:

- Store data or reserve memory for data storage
- Control the alignment of parts of the program in memory
- Specify the methods of accessing the sections of memory in which the program will be stored
- Specify the entry point of the program or a part of the program
- Specify the way in which symbols are referenced
- Specify that a part of the program is to be assembled only under certain conditions
- Control the format and content of the listing file

- Display informational messages
- Control the assembler options that are used to interpret the source program
- Define new opcodes

VAX MACRO directives define macros and repeat blocks. With these directives, you can repeat identical or similar sequences of source statements and use string operators to manipulate and test the contents of source statements.

7.12.2 VAX MACRO-32 Compiler for OpenVMS AXP

The VAX MACRO-32 Compiler for OpenVMS AXP is available for porting VAX MACRO code to an OpenVMS AXP system.

Digital recommends the use of mid- and high-level languages for developing new applications. The VAX MACRO-32 Compiler is provided only for porting existing VAX MACRO code to OpenVMS AXP systems.

For information about using this compiler, see *Migrating to an OpenVMS AXP System: Porting VAX MACRO Code.*

7.12.3 MACRO-64 Assembler for OpenVMS AXP

MACRO-64 Assembler for OpenVMS AXP (MACRO-64) is the assembly language for AXP systems that provides precise control over instructions and data. MACRO-64 supports a rich macro processing language that includes the following features:

- Macro definition and expansion
- Symbolic labels, assembly-time variables, relocatable expression processing, and 64-bit absolute expression processing
- Lexical string symbols and lexical operators
- Conditional assembly
- Implicit base register support
- Optional automatic data alignment
- Optional code optimizations

Included with MACRO-64 are a number of library macros that enable you to write MACRO-64 programs that conform with the OpenVMS calling standard.

7.13 OPS5

DEC OPS5 is a preferred tool for developing high-performance, commercial quality rule-based systems. Such systems are well suited to solve problems in the following areas:

- Configuration
- Selection
- Diagnosis
- Process monitoring and control
- Scheduling

- Planning
- Decision support
- Rapid prototyping

DEC OPS5 for OpenVMS provides an upward compatible migration path for users of VAX OPS5 Version 3.0 or earlier versions. Applications written in DEC OPS5 can call routines written in other languages, and those routines can, in turn, call the DEC OPS5 run-time system. A main program written in another language can also call a DEC OPS5 application.

The DEC OPS5 language is also available in source-compatible form on the RISC /ULTRIX and DEC OSF/1 platforms with some minor restrictions. For details, refer to the DEC OPS5 for RISC Software Product Description (SPD 39.31.xx).

7.14 Pascal

DEC Pascal can use all OpenVMS operating system features, including the following:

- Support for the OpenVMS Debugger
- Compilation of separate modules
- Access to other object modules written in other languages
- Access to all callable system routines
- Access to CDD/Repository data declarations

Along with the standard ANSI Pascal features, DEC Pascal incorporates the following features:

- Exponentiation and concatenation operator
- Hexadecimal, octal, and DOUBLE constants
- Uppercase and lowercase letters treated identically, except in character and string constants
- Dollar sign ($) and underscore (_) characters in identifiers
- DOUBLE, SINGLE, QUADRUPLE, VARYING character strings and UNSIGNED data types
- I/O, arithmetic, ordinal, boolean, transfer, dynamic allocation, character string manipulation, unsigned, and allocation size defined routines
- READ (or READLN) of user-defined ordinal type and string
- WRITE (or WRITELN) of user-defined scalar type or any data using binary, hexadecimal, or octal format
- Conformant array parameters for processing arrays with potentially different bounds
- Optional attribute specification on types, variables, routines, and compilation units to change many of the properties of a program

7.15 PL/I

Digital offers two versions of PL/I: VAX PL/I for developing programs on OpenVMS VAX and DEC PL/I for developing programs on OpenVMS AXP. They both incorporate the following features:

- Compile-time preprocessor that allows language extension and conditional compilation
- Several program control constructs (DO, IF-THEN-ELSE, BEGIN-END, LEVEL, SELECT-WHEN-OTHERWISE, and CALL)
- AUTOMATIC initializations, AREA (user allocation), OFFSET, scalar assignment to arrays, the REFER structure, the ENTRY statement, and the TYPE and LIKE attributes
- Optional access to the CDD/Repository and to the Language-Sensitive Editor component of LSE/SCA
- OpenVMS Debugger support
- Access to callable system routines

VAX

VAX PL/I also offers optional access to the Source Code Analyzer component of LSE/SCA and the Program Design Facility.♦

8

Linker and Librarian

After a program is compiled or assembled, it must be linked to produce an application that can run on an OpenVMS VAX or OpenVMS AXP system. The OpenVMS Linker Utility (linker) performs the following major steps:

- Resolves references to global symbols among the input modules
- Allocates virtual memory for the image
- Initializes the image

You can supply linker options to the linker in an options file. The use of an options file eliminates the need to retype a long linker command every time you relink your module or application.

You can use output from the linker to debug programs. Use the image map to locate an instruction that caused a run-time error, translate a number displayed by the debugger to its related symbol or address, and locate definitions of symbols.

This chapter briefly describes the features of the linker and the Librarian utility. For complete information about using the linker, refer to the *OpenVMS Linker Utility Manual*.

8.1 Linker Input and Output

Depending on the needs of your program, the linker can accept input from the following sources:

- Object file—Any object module created after compiling or assembling a source program.
- Shareable image file—A separate image that was already linked but which cannot be run as a separate file.
- Symbol table file—A separate symbol table produced by a previous link operation. The symbol table contains global symbols and values of an image.
- Library file—The linker accepts object module libraries and shareable image libraries.
- Options file—Input file specifications and link options that cannot be defined at the DCL command level can be specified in this file.

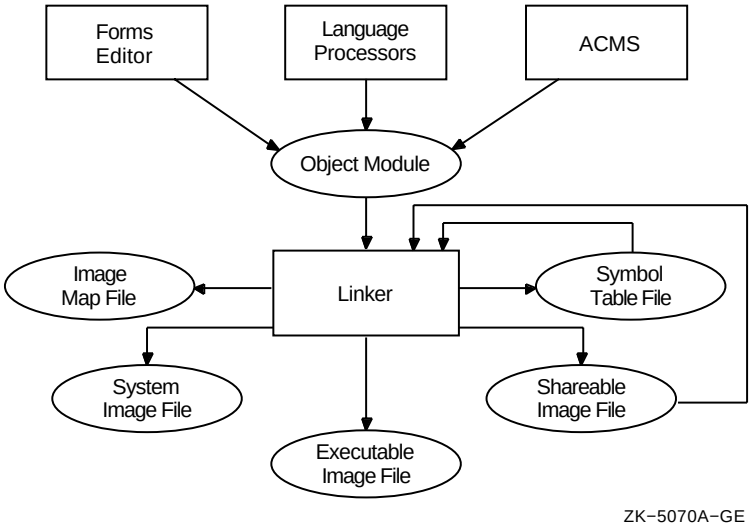
Primarily, the linker produces an executable image of the program. In addition, the linker has the capability to produce the following:

- A shareable image—An image that can be used by other programs but cannot be executed independently.
- A system image—An image that does not execute under the control of the operating system; rather it operates standalone on VAX or AXP hardware.

- An image map—A file containing additional program information including object module synopsis, module relocatable reference synopsis, image section synopsis, program section synopsis, symbols by name and value, image synopsis, and link run statistics.
- A symbol table file—A file containing symbols and their values to be used by other programs being linked.

Figure 8–1 illustrates the relationship of the linker to the language processor in the program development process.

Figure 8–1 Position of the Linker in Program Development



8.2 Linker Command Summary

You can use linker qualifiers to control linker operations in the following ways:

- To produce an abbreviated image map
- To generate a debug symbol table to give the debugger control when the image is run
- To place the entire executable image in P0 address space
- To produce and protect shareable images
- To create a system image
- To include traceback information in the image
- To search system default, shareable image default, and user-default libraries to resolve references

For more information about using linker qualifiers, see the *OpenVMS Linker Utility Manual*.

8.3 Using the LIBRARIAN with the Linker

You can use the Librarian utility (LIBRARIAN) to collect input—object modules and shareable images—for the linker. You can assign system-defined logical names to the libraries. Then, the linker automatically searches these libraries to resolve references. The library logical names are LNK$LIBRARY and LNK$LIBRARY_1 through LNK$LIBRARY_999. When you associate libraries with these logical names, do not skip any logical names in the sequence.

If you choose a name for your library other than the system-defined logical library names, you can link it to your program by specifying the qualifier LIBRARY in the LINK command. For example, you could store object modules in the library INCOME.OLB. To link it with the program INCOME.OBJ, enter the following command:

```
$ LINK INCOME,INCOME.OLB/LIBRARY
```

8.4 Additional Linker Features

The linker also incorporates the following features:

- Options file
- Image map
- Object language

Use an options file to specify linker options and input file specifications. Linker options allow you to control aspects of a link operation, such as the starting virtual address of an image. Shareable images used as input files must be specified in an options file. In addition, options files can be useful to specify a set of frequently used input files or to specify input files in a LINK command that exceeds the maximum size of a DCL command.

The image map contains information on the contents of the image and on the link process. You can use the map to locate link-time errors, view the image layout in virtual memory, keep track of global symbols, and so forth.

8.5 Librarian Utility

Libraries are files you create to store frequently used modules of code or text. With the LIBRARIAN, you can create a library, maintain the modules in a library, or display information about a library and its modules. You use LIBRARIAN commands to manage modules within a library. You can use DCL commands to manage the entire library as one unit. For example, if you want to rename the library, use the DCL command RENAME.

8.5.1 Library Types

Table 8–1 lists the types of available libraries.

Table 8–1 Types of Libraries

	File Type		
Library	**Library**	**Module**	**Contents**
Help	HLB	HLP	Help text modules that provide users with information about a program
Macro	MLB	MAR	VAX MACRO definitions used as input to the assembler
Object	OLB	OBJ	Object modules of frequently called routines
Shareable image	OLB	EXE	Symbol tables of shareable images used as input to the linker
Text	TLB	TXT	Sequential record files used as input data to a program

8.5.2 Using the LIBRARY Command

The DCL command LIBRARY invokes the LIBRARIAN and accepts a number of command qualifiers. The LIBRARIAN manages library modules in the following ways:

- Creates a new library and specifies the type
- Adds, deletes, or replaces a module within the library
- Copies a module from the library
- Lists the modules in the library, with a history, with global symbols, or before or after a specified time
- Enables a log of each library action

You can create command procedures that manipulate libraries using the DCL command LIBRARY.

8.5.3 Sharing Code Using Text Libraries

You can share code easily by creating text or macro libraries that all users can access. You can also share data by creating text libraries of data files that all users can access.

For complete information on creating, managing, and using libraries, refer to the *OpenVMS Command Definition, Librarian, and Message Utilities Manual*.

9
Debugging and Testing Tools

This chapter describes the debuggers and testing tools supplied with the OpenVMS operating system and some optional testing tools, as shown in Table 9–1.

Table 9–1 Debugging and Testing Tools

Name	Built-in or Optional	Where Described
OpenVMS Debugger	Built-in	Section 9.1
OpenVMS Delta/XDelta Debugger	Built-in	Section 9.2
‡OpenVMS AXP System-Code Debugger	Built-in	Section 9.3
System Dump Analyzer	Built-in	Section 9.4
Crash Log Utility Extractor	Built-in	Section 9.5
DEC Performance and Coverage Analyzer	Optional	Section 9.6
DEC Test Manager	Optional	Section 9.7

‡OpenVMS AXP

DEC Performance Coverage Analyzer (PCA) and DEC Test Manager are components of DECset, an integrated programming tool set that supports software developers' coding, debugging, testing, and maintenance activities.

The DEC Language-Sensitive Editor/Source Code Analyzer (LSE/SCA), another optional testing tool and also a component of DECset, is described in Section 5.3.

9.1 OpenVMS Debugger

The OpenVMS Debugger (debugger) is a symbolic debugger and is the preferred debugger for debugging user-mode code. It enables you to reference program locations using the symbols you defined in the program. You do not need to keep track of program addresses. To enter commands with the debugger, you can use the keypad, the command line, or an input file (to enter a lengthy series of commands). The debugger has a screen mode that allows you to view several lines of source code at one time, the commands you enter, and the output from the commands you enter. It also provides a robust error message facility.

The debugger helps you locate run-time programming or logic errors, also known as bugs. You use the debugger with a program that has been compiled and linked successfully but does not run correctly. For example, the program might give incorrect output, go into an infinite loop, or terminate prematurely.

You can locate errors with the debugger by observing and manipulating your program interactively as it executes. By entering debugger commands at the terminal, you can perform the following basic debugging techniques:

- Display your program's source code, identifying where execution is currently paused
- Browse through the source code to locate points of interest where you might test for certain conditions
- Set breakpoints to suspend program execution at such points
- Execute your program, including stepping one source line at a time and restarting from the beginning of the program
- Trace the execution path of the program
- Display the current value of a program variable
- Monitor changes in variables and other program entities during program execution
- Change the value of a variable and, in some cases, test the modification without editing the source code, recompiling, and relinking
- Monitor exception conditions and language-specific events, and force events to occur

These are the basic debugging techniques. After you are satisfied that you have found the error in the program, you can edit the source code and compile, link, and execute the corrected version.

9.1.1 Programming Language Support

Most of the languages supported by the debugger are available on both OpenVMS VAX and OpenVMS AXP, as shown in Table 9–2.

Table 9–2 Language Support on OpenVMS VAX and OpenVMS AXP

On VAX and AXP	On VAX Only	On AXP Only
Ada	Bliss	MACRO-64
BASIC	DIBOL	
C		
C++		
COBOL		
Fortran		
Pascal		
PL/I		
‡VAX MACRO		

‡On OpenVMS AXP, VAX MACRO is supported by the MACRO-32 compiler, which converts VAX MACRO code into AXP machine language code.

The debugger recognizes the syntax, data types, operators, expressions, scoping rules, and other constructs of a given language. You can change the debugging context from one language to another during a debugging session with the SET LANGUAGE command.

9.1.2 User-Interface Options

The debugger has the following user-interface options to accommodate different needs and debugging styles:

- DECwindows interface for workstations

  When using this interface, you interact with the debugger by manipulating a mouse and pointer to choose items from menus, click on buttons, select names in windows, and so on. This interface is the default interface. It provides the basic debugging and convenience features that you will probably need most of the time.

- Command interface for terminals and workstations

  When using this interface, you interact with the debugger by entering commands at a prompt. In addition to general-purpose debugging features, the command interface provides special features not available through the default DECwindows interface (for example, changing the radix for the display of integer data).

  The DECwindows interface also has a command-entry prompt. You can use it as an alternative to the DECwindows interface for certain operations, including debugging tasks not available through the DECwindows interface.

9.1.3 Functional Features of the Command Interface

The functional features of the command interface are highlighted in the following paragraphs.

Symbolic Debugging

Because the OpenVMS Debugger is a symbolic debugger, you can refer to program locations by the symbols you used for them in your program—the names of variables, routines, labels, and so on. You do not need to specify memory addresses or machine registers when referring to program locations, although you can, if you want.

Support for All Data Types

The debugger understands all compiler generated data types, such as integer, floating point, enumeration, record, array, and so on. It displays the values of program variables according to their declared type.

Flexible Data Format

The debugger permits a variety of data forms and types for entry and display. By default, the source language of the program determines the format used for the entry and display of data. You can also impose other formats. For example, by using a type or radix qualifier with the EXAMINE command, you can display the contents of a program location in ASCII, word-integer, or floating-point format.

Starting or Resuming Program Execution

You start or resume program execution with the GO or STEP commands. The GO command causes the program to execute until a breakpoint is reached, a watchpoint is modified, an exception is signaled, or the program terminates. The STEP command enables you to execute a specified number of lines or instructions, or up to the next instruction of a specified class.

Breakpoints

By setting breakpoints with the SET BREAK command, you can suspend program execution at specified locations and check the current status of your program. Rather than specify a location, you can also suspend execution on certain classes of instructions or on every source line. Also you can suspend execution on certain kinds of events, such as exceptions and tasking (multithread) events.

Tracepoints

By setting tracepoints with the SET TRACE command, you can monitor the path of program execution through specified locations. When a tracepoint is triggered, the debugger reports that the tracepoint was reached and then continues execution. As with the SET BREAK command, you can also trace through classes of instructions and monitor events.

Watchpoints

By setting a watchpoint with the SET WATCH command, you can cause execution to stop whenever a particular variable or other memory location has been modified. When a watchpoint is triggered, the debugger suspends execution at that point and reports the old and new values of the variable.

Manipulation of Variables and Program Locations

With the EXAMINE command, you can determine the value of a variable or program location. The DEPOSIT command enables you to change that value. You can then continue execution to see the effect of the change, without having to recompile, relink, and rerun the program.

Evaluation of Expressions

With the EVALUATE command, you can compute the value of a source-language expression or an address expression. You specify expressions and operators in the syntax of the language to which the symbolic debugger is currently set.

Control Structures

You can use logical control structures (FOR, IF, REPEAT, WHILE) in commands to control the execution of other commands.

Shareable Image Debugging

You can debug shareable images (images that are not directly executable). The SET IMAGE command enables you to reference the symbols declared in a shareable image.

Multiprocess Debugging

You can debug multiprocess programs (programs that run in more than one process). The SHOW PROCESS and SET PROCESS commands enable you to display process information and control the execution of images in individual processes.

Task Debugging

You can debug tasking programs (also known as multithread programs). These programs use DECthreads or POSIX 1003.4a services, or use language-specific tasking services (for example, Ada tasking programs). The SHOW TASK and SET TASK commands enable you to display task information and control the execution of individual tasks.

Vector Debugging (VAX Only)

VAX

On VAX systems, you can debug vectorized programs, that is, programs that use VAX vector instructions. You can control and monitor execution at the vector instruction level, examine and deposit vector instructions, manipulate the contents of vector registers, use a mask to display specific vector elements, and control synchronization between the scalar and vector processors.♦

Terminal and Workstation Support

The debugger supports all VT-series terminals and MicroVAX workstations.

9.1.4 Convenience Features of the Command Interface

The convenience features of the command interface are highlighted in the following paragraphs.

Online Help

Online help is always available during a debugging session. Online help contains information about all debugger commands and selected topics.

Source Code Display

You can display lines of source code for all supported languages during a debugging session.

Screen Mode

In screen mode, you can display and capture various kinds of information in scrollable windows that can be moved around the screen and resized. Automatically updated source, instruction, and register displays are available. You can selectively direct debugger input, output, and diagnostic messages to displays. You can also create DO displays that capture the output of specific command sequences.

Running and Rerunning a Program

With the RUN and RERUN commands, you can run a new program or rerun the same program from the current debugging session without first exiting and restarting the debugger. When you rerun a program you can choose to either activate or deactivate any previously set breakpoints, tracepoints, and watchpoints.

Keypad Mode

When you start the debugger, several commonly used debugger command sequences are assigned by default to the keys of the numeric keypad (if you have a VT52, VT100, or LK201 keyboard). Therefore, you can enter these commands with fewer keystrokes than if you were to type them at the keyboard. You can also create your own key definitions.

Source Editing

As you find errors during a debugging session, you can use the EDIT command to invoke any editor available on your system. You specify the editor you want with the SET EDITOR command. If you use the DEC Language-Sensitive Editor /Source Code Analyzer (LSE/SCA), the editing cursor is automatically positioned within the source file whose code appears in the screen-mode source display.

Command Procedures

You can direct the debugger to execute a command procedure (a file of debugger commands) to recreate a debugging session, to continue a previous session, or to avoid typing the same debugger commands many times during a debugging session. You can pass parameters to command procedures.

Initialization Files

You can create an initialization file containing commands to set your default debugging modes, screen display definitions, keypad key definitions, symbol definitions, and so on. When you invoke the debugger, those commands are executed automatically to tailor your debugging environment.

Log Files

You can record in a log file the commands you enter during a debugging session and the debugger's responses to those commands. You can use log files to keep track of your debugging efforts, or you can use them as command procedures in subsequent debugging sessions.

Symbol Definitions

You can define your own symbols to represent lengthy commands, address expressions, or values in abbreviated form.

9.1.5 Convenience Features of the DECwindows Interface

The following paragraphs highlight some of the convenience features of the debugger's default DECwindows interface.

Source-Code Display

The debugger is a source-level debugger. The source-code display in the source window is automatically updated to show where program execution is paused currently. You can enable and disable the display of compiler-generated line numbers.

A source browser feature lists the modules and routines of your program and lets you display source code in arbitrary modules and set breakpoints on routines. By double clicking on program names and module names, you can list the underlying hierarchy of modules and routines.

Call-Stack Navigation

A menu on the source window lists the sequence of routine calls currently on the call stack. By clicking on a routine name, you can set the context (scope) for source display, instruction display (in the instruction window), and symbol searches to any routine on the stack.

Breakpoints

You set, deactivate, and activate breakpoints by clicking on buttons next to the source lines in the source window. You can set conditional breakpoints or action breakpoints. The latter execute one or more debugger commands when the breakpoint triggers. The source window buttons and the Breakpoint View give a visual indication of activated, deactivated, and conditional breakpoints.

Push Buttons

Push buttons in the control panel control common operations: by clicking on a button, you can start execution, step to the next source line, display the value of a variable selected in a window, interrupt execution, and so on.

You can add, modify, and remove buttons and the associated debugger commands.

Displaying and Manipulating Data

To display the value of a variable, you select its name from the source window and click on a button.

The Monitor View automatically displays the updated values of specified variables whenever the debugger regains control from your program.

To display the values of the individual elements of an aggregate variable (such as an array), you double click on the name of the variable in the Monitor View. To assign a new value to a variable, you edit the currently displayed value. To set a watchpoint, you click on a button next to its name.

To dereference a pointer variable (to display the value of the referenced object), you double click on the name of the pointer variable.

Run/Rerun Program

You can rerun the same program or run another program from the same debugging session without exiting the debugger. When rerunning a program, you can choose to save the current state (activated or deactivated) of breakpoints, tracepoints, and static watchpoints. The Run/Rerun feature is also available in the debugger's command interface.

Instruction and Register Views

The Instruction View shows the decoded instruction stream of your program—the code that is actually executing. This is useful if the program you are debugging has been optimized by the compiler so that the information in the source window does not exactly reflect the code that is executing.

The Register View displays the current contents of all machine registers. You can edit the displayed values to deposit other values into the registers.

Tasking Program Support

The Task View displays information about the current state of all tasks of a tasking program (also called a multithread program). You can modify task characteristics to control task execution, priority, state transitions, and so on.

Integration with Command Interface

The debugger's DECwindows interface is layered on, and closely integrated with, the command-driven debugger:

- When you use the DECwindows interface, the resulting commands are echoed in the message region so that you can correlate your input with the corresponding command line that the debugger processes.
- When you enter commands at the prompt, they update the DECwindows views accordingly.

Customization

You can modify the following and other aspects of the debugger's DECwindows interface and save the current settings in a resource file to customize your debugger startup environment:

- Configuration of windows and views
- Control-panel button labels and associated debugger commands, including adding and removing buttons and commands
- Character fonts for displayed text

Online Help

Online help is available for the debugger's DECwindows interface (context-sensitive help) and its command interface.

For more information about the debugger, see the *OpenVMS Debugger Manual*.

9.2 OpenVMS Delta/XDelta Debugger

The OpenVMS Delta/XDelta Debugger (DELTA/XDELTA) is a primitive debugger. It is used to debug code that cannot be debugged with the symbolic debugger, that is, any code that executes at interrupt priority levels (IPLs) above IPL0 or any code that executes in supervisor, executive, or kernel mode. Examples include user-written device drivers and the OpenVMS operating system.

Almost all the commands available on DELTA are also available on XDELTA. Furthermore, both DELTA and XDELTA use the same expressions. However, they are different in two ways: you use them to debug different kinds of code, and you invoke and exit from them in different ways.

You can use DELTA to debug programs that execute at IPL0 in any processor mode (user, supervisor, executive, and kernel). You can also debug user-mode programs with DELTA, but the debugger is more suitable. To run DELTA in a processor mode other than user mode, your process must have the privilege that allows DELTA to change to that mode—change-mode-to-executive (CMEXEC) or change-mode-to-kernel (CMKRNL) privilege. You cannot use DELTA to debug code that executes at an elevated IPL.

You can use XDELTA to debug programs that execute in any processor mode and at any IPL. To use XDELTA, you must have system privileges, and you must include XDELTA when you boot the system.

You can use DELTA/XDELTA commands to perform the following debugging tasks:

- Open, display, and change the value of a particular location
- Set, clear, and display breakpoints
- Set display modes in byte, word, longword, or ASCII
- Display instructions
- Execute the program in a single step with the option to step over a subroutine
- Set base registers
- List the names and locations of all loaded modules of the executive

For more information about using DELTA/XDELTA for debugging, see the *OpenVMS Delta/XDelta Debugger Manual*.

9.3 OpenVMS AXP System-Code Debugger (AXP Only)

AXP

OpenVMS AXP supports a new symbolic debugger that can be used to debug nonpageable system code and device drivers running at any IPL. The OpenVMS AXP System-Code Debugger lets you use the familiar OpenVMS Debugger interface to observe and manipulate system code interactively as it executes.

Using the OpenVMS AXP System-Code Debugger, you can perform the following tasks:

- Control the system software's execution—-stop at points of interest, resume execution, intercept fatal exceptions, and so on
- Trace the execution path of the system software
- Monitor exception conditions

- Examine and modify the values of variables
- In some cases, test the effect of modifications without having to edit the source code, recompile, and relink

Because the OpenVMS AXP System-Code Debugger is a symbolic debugger, you can specify variable names, routine names, and so on, precisely as they appear in the source code. Using the OpenVMS AXP System-Code Debugger, you can display the source code where the software is executing and step through the source code, line by line.

You can use this debugger to debug code with the following languages:

- C
- Bliss
- VAX MACRO

Note

A Bliss compiler is not available for OpenVMS AXP.

The OpenVMS AXP System-Code Debugger recognizes the syntax, data typing, operators, expressions, scoping rules, and other constructs of a given language. If your program is written in more than one language, you can change the debugging context from one language to another during a debugging session. For information about how to use the OpenVMS AXP System-Code Debugger and how it differs from the OpenVMS Debugger, see the *OpenVMS AXP Device Support: Developer's Guide*. For information about OpenVMS Debugger commands, see the *OpenVMS Debugger Manual*.♦

9.4 System Dump Analyzer

The System Dump Analyzer utility (SDA) helps you determine the cause of system failures. You invoke this utility specifying a system crash dump file, which is a copy of memory at the time of a system crash. SDA reads the dump file; then, it formats and displays the contents of the file. In addition to information contained in the dump file, SDA reads the system's symbol table file. You can specify that SDA read the symbols that define many of the system's data structures, including those in the I/O database.

You can also use SDA to analyze a running system. To do this, you need change-mode-to-kernel (CMKRNL) privilege. This option is useful for examining the stack and memory of a process stalled in a scheduler state.

If you are examining a dump file, SDA displays the immediate cause of the crash. You can then use SDA to diagnose how the error occurred. For example, you can use SDA commands to locate the line of code that signaled the bugcheck and to find the line of code (usually on the stack) that caused the error. Then, you can examine device drivers, linker maps, and system maps to locate the module where the line of code came from. Once the module has been identified, you can examine the module code to pinpoint the problem.

You can locate the error using SDA commands that allow you to view the following pieces of information:

- The location and contents of the four process stacks
- VAX On a VAX computer, the location and contents of the systemwide interrupt stack♦
- The active processes and the values of the parameters used in swapping and scheduling these processes
- The software and hardware context of any process
- The value of a symbol and the contents of the location to which the symbol points
- A formatted list of a block of memory
- The list of system page table entries
- The lookaside lists, the nonpaged dynamic storage pool, and the paged dynamic storage pool
- All locks in the system
- The names of the RMS data structures
- All data structures associated with a device
- The VMScluster or the system communications services cluster
- The active connections between systems communication services processes
- The dump file header
- The response identifications

The SDA commands also allow you to switch processes, direct output to a log file or terminal, scan memory locations, assign a value to a symbol, read global symbols to add them to the SDA symbol table, and repeat the execution of the last command.

For more information about the System Dump Analyzer on AXP systems, see the *OpenVMS AXP System Dump Analyzer Utility Manual*. For more information about the System Dump Analyzer on VAX systems, see the *OpenVMS VAX System Dump Analyzer Utility Manual*.

9.5 Crash Log Utility Extractor

The Crash Log Utility Extractor (CLUE) is a tool for recording a history of crash dumps and key parameters for each crash dump and for extracting and summarizing key information from each crash dump. Unlike crash dumps, which are overwritten with each system crash and are available only for the most recent crash, the crash history file on OpenVMS VAX and the summary crash history file (with a separate listing file for each crash) on OpenVMS AXP, are permanent records of system crashes.

By examining the key parameters of a crash, you can identify and resolve the issues that caused it.

The implementation differences between OpenVMS VAX and OpenVMS AXP are shown in Table 9–3.

Table 9–3 CLUE Differences Between OpenVMS VAX and OpenVMS AXP

	OpenVMS VAX	OpenVMS AXP
Attribute		
Access method	Invoked as a separate utility.	Accessed through SDA.
History file	A cumulative file that contains a one-line summary and detailed information from the crash dump file for each crash.	A cumulative file that contains only a one-line summary for each crash dump. The detailed information for each crash is put in a separate listing file.
Uses in addition to debugging crash dumps	None.	CLUE commands can be used interactively to examine a running system.
Where documented	*OpenVMS System Manager's Manual* and *OpenVMS System Management Utilities Reference Manual*	*OpenVMS System Manager's Manual* and *OpenVMS AXP System Dump Analyzer Utility Manual*

9.6 DEC Performance and Coverage Analyzer

DEC Performance and Coverage Analyzer (PCA) is a component of DECset. Each DECset tool provides a DECwindows Motif user interface and a consistent look and feel across platforms. PCA is designed to help software engineers analyze and improve the runtime functioning of application programs.

DEC PCA serves two functions:

- Helps pinpoint execution bottlenecks and other performance problems so that users can modify their programs to run faster.
- Provides test coverage analysis by measuring what program sections have or have not been executed by a specified set of test data. With this information, users can create new tests to exercise their programs more thoroughly.

DEC PCA is fully symbolic and uses the Debug Symbol Table (DST) information in the user's program to access the symbolic names of program locations. Consequently, applications written in any of the OpenVMS languages that produce DST information can be analyzed with DEC PCA.

DEC PCA consists of two parts—the Collector and the Analyzer. The Collector gathers performance or test coverage data on a running user program and writes that data to a performance data file. The Analyzer—a separate, interactive program—then reads the performance data file and presents the results as performance histograms and tabular displays.

DEC PCA can be used to collect and analyze the following kinds of data:

- CPU sampling data—The program counter (PC) can be sampled to determine which sections of an application use the most CPU time during program execution.
- Program counter sampling data—The program counter can be sampled at a specified interval (by default, every 10 milliseconds) to determine which sections of an application take the longest time to run.

- Ada multitasking data—Many types of multitasking data can be gathered to determine which tasks consume the most resources.
- Exact execution counts—Information about the exact number of times specified program locations are executed helps illuminate an application's dynamic functions.
- Test coverage data—Information that reveals which code paths are or are not executed when an application is tested enables users to create more complete tests.
- Event markers—Significant events in the execution of the program (for example, entering a routine) can be marked to permit later filtering of the data.

For more information about DEC PCA, see the *Guide to Performance and Coverage Analyzer for VMS Systems*.

9.7 DEC Test Manager

DEC Test Manager is a component of DECset. Like the other components of DECset, it provides a DECwindows Motif user interface and a consistent look and feel across platforms. It also provides a command-line user interface.

DEC Test Manager is based on the concept of regression testing. Regression testing is a method of ensuring that a program being developed runs correctly and that new features added to the program do not affect the correct execution of previously tested features.

In regression testing, users run established software tests and compare the actual test results with the results that were expected. If these actual results do not agree with the expected results, the software being tested may contain errors. If errors do exist, the software being tested is said to have regressed.

DEC Test Manager automates regression testing of software during the development and maintenance phases by executing user-supplied tests and automatically comparing the results with expected test results. Programmers supply and select the tests they want to run.

The use of DEC Test Manager by itself can improve programmer productivity and software reliability. You can organize, run, compare, and store test results efficiently. You can repeat tests and review results as often as needed. Integration of the DEC Test Manager with PCA and CMS provides a further enhanced testing environment with corresponding productivity gains.

For more information about the DEC Test Manager, see the *Guide to Test Manager for VMS Systems*.

10

Using Callable System Routines

The OpenVMS operating system includes the following callable system routines that perform various tasks:

- Run-time library (RTL) routines
- System services
- Utility routines
- OpenVMS RMS (hereafter referred to as RMS)

In this manual, a routine is a closed, ordered set of instructions that performs one or more specific tasks. Every routine has an entry point (the routine name), and optionally an argument list. Procedures and functions are specific types of routines: a procedure is a routine that does not return a value, whereas a function is a routine that returns a value by assigning that value to the function's identifier.

This chapter briefly describes the routines and references appropriate manuals for more information.

10.1 Deciding Which Routines to Use

You can use system routines in programs to complete programming tasks such as:

- I/O operations
- Security procedures
- File manipulation
- Memory management
- Screen management
- Mathematics operations
- Event synchronization
- Utility usage

The sections that follow suggest sets of routines to use for each of these general programming tasks.

10.1.1 I/O Operations

For I/O operations, you can use RMS, RTL routines, or system services. Use RMS for device-independent I/O, when you want more control over file access. Use RTL routines to get more functionality than language I/O statements. Use system services for device-dependent I/O when you want more control over the device. System services allow you to access devices not supported by RMS, to perform I/O operations not supported by a particular language, and to increase I/O performance.

10.1.2 Security Procedures

For security procedures, use system services to maintain rights database, to use access control lists and process rights lists, to check access protection, and to provide security erase patterns. To assign protection to a particular file, use RMS.

10.1.3 File Management

For complex file manipulation, you would generally use RMS. RMS can create complex file organizations; reorganize files; extend disk space for files; and get, locate, insert, update, and delete records. There are RMS and RTL routines for simple file manipulation such as opening, reading, deleting, renaming, and closing files.

10.1.4 Memory Management

For memory management tasks, both RTL routines and system services can acquire and free virtual memory. RTL memory management routines call system services. RTL routines maintain a processwide pool of free pages that are automatically reused. If you call system services directly, the program must keep track of free pages. Direct calls to system services should be used when the size requirements exceed 1000 pages for one request. RTL routines working with such large requests may result in fragmenting the virtual address space. System services give you more control because you can specify a specific virtual address and unlock pages in memory.

10.1.5 Screen Management

For screen management, use RTL routines. The screen management routines allow you to build terminal-independent screen management functions. They do not rely on particular hardware devices; input is read from a virtual keyboard and output is sent to a virtual display. With SMG$ routines, complex screens can be built with several regions defined. The program can then work within a region without regard to its position on the screen.

10.1.6 Math Operations Specific to OpenVMS

For math routines for OpenVMS systems, RTL routines can complete simple arithmetic as well as the following functions:

- Exponentiation
- Complex exponentiation
- Complex function evaluation
- Floating-point trigonometric function evaluation
- Absolute value
- Numeric data conversions

10.1.7 Digital Portable Mathematics Library (AXP Only)

AXP

For math routines on AXP systems, the Digital Portable Mathematics Library (DPML) provides a wide variety of mathematical routines including:

- Floating-point trigonometric function evaluation
- Exponentiation, logarithmic, power function evaluation
- Hyperbolic function evaluation

- Algebraic function evaluation
- Complex function evaluation
- Complex exponentiation
- Miscellaneous function evaluation

If you want to maintain compatibility with future libraries and create portable mathematical applications, Digital recommends that you use the DPML routines available through the high-level language of your choice (for example, FORTRAN or C) rather than using the call interface. This will guarantee the functioning of the routines across platforms. Because of the complex relationship between high-level languages and DPML routines, the behavior of direct calls to DPML routines may change in future releases. DPML routines also provide significantly higher performance and accuracy.

For more information about using DPML routines, refer to *Digital Portable Mathematics Library*.♦

10.1.8 Event Synchronization

For event synchronization, use RTL routines or system services. Use RTL routines to synchronize events with event flags. Use system services to synchronize events with event flags, with a resource lock, and with an asynchronous system trap (AST).

10.2 RTL Routines

The OpenVMS Common Run-Time Procedure Library (or the Run-Time Library) is a set of language-independent procedures that perform a wide variety of operations. These RTL routines follow the OpenVMS Calling Standard; they are part of the Common Run-Time environment. The Common Run-Time environment lets a program contain routines written in different languages, so that you can call RTL routines from any language, thus increasing program flexibility.

10.2.1 Organization of the Run-Time Library

The routines of the OpenVMS RTL are grouped according to the types of tasks they perform; these groups are referred to as facilities. Each group or facility has an associated prefix that is used in the routine name to identify that routine as a member of a particular facility. Table 10–1 lists all the RTL facility prefixes and the types of tasks each facility performs.

Table 10–1 Run-Time Library Facilities

Facility Prefix	Types of Tasks Performed
DTK$	DECtalk routines that are used to control Digital's DECtalk device
LIB$	Library routines that obtain records from devices, manipulate strings, convert data types for I/O, allocate resources, obtain system information, signal exceptions, establish condition handlers, enable detection of hardware exceptions, and process cross-reference data
MTH$	Mathematics routines that perform arithmetic, algebraic, and trigonometric calculations

(continued on next page)

Table 10–1 (Cont.) Run-Time Library Facilities

Facility Prefix	Types of Tasks Performed
OTS$	General purpose routines that perform tasks such as data type conversions as part of a compiler's generated code, and also some mathematical functions
PPL$	Parallel processing routines that simplify subprocess creation, interprocess communication, and resource sharing for parallel applications
SMG$	Screen management routines that are used in designing, composing, and keeping track of complex images on a video screen
STR$	String manipulation routines that perform such tasks as searching for substrings, concatenating strings, and prefixing and appending strings

10.2.2 Features of the RTL

The RTL provides the following features and capabilities:

- RTL routines perform a wide range of general utility operations. You can call an RTL routine from any OpenVMS language instead of writing your own code to perform the operation.

  Routines in the RTL are part of the OpenVMS Common Run-Time environment; therefore, they can be called from any OpenVMS language.

- Because many of the routines are shared, they take up less space in memory.
- When new versions of the RTL are installed, you do not need to revise your calling program, and generally do not need to relink.
- All RTL routines are fully reentrant unless the description of the facility or the routine specifies otherwise.

  The term reentrant means that the routine executes correctly regardless of how many threads of execution are executing at the same time. Currently, reentrancy is supported only when those multiple threads are executing on the same processor. The term AST-reentrant means that a routine may be interrupted and reentered from itself or an AST-level thread of execution only. In particular, an AST-reentrant routine may not execute properly if more than one non-AST-level thread of execution is executing the routine at once.

  Because the Run-Time Library routines are reentrant (unless otherwise noted), they can be called from multiple threads of execution. For example, a routine may be called from both an AST-level thread and a non-AST-level thread of an image, as well as from the multiple tasks of an Ada program.

10.3 System Services

System services are procedures that the operating system uses to control resources available to processes; to provide for communication among processes; and to perform basic operating system functions, such as the coordination of input/output operations.

Although most system services are used primarily by the operating system on behalf of logged-in users, they are also available for general use and provide mechanisms that you can use in application programs. For example, when you log in to the operating system, the Create Process ($CREPRC) system service is called to create a process on your behalf. You may, in turn, write a program

that calls the $CREPRC system service to create a subprocess to perform certain functions for an application.

System services can be divided into functional groups. Table 10–2 lists each group of system services and its function.

Table 10–2 Functional Groups of System Services

Services Group	Function
AST	Process execution can be interrupted by events (such as I/O completion) for the execution of designated subroutines. These software interrupts are called asynchronous system traps (ASTs) because they occur asynchronously to process execution. System services are provided so that a process can control the handling of ASTs.
‡Cluster Event Notification	Cluster event notification services manage notification requests of cluster configuration events.
Condition-Handling	Condition handlers are procedures that can be designated to receive control when a hardware or software exception condition occurs during image execution. Condition-handling services designate condition handlers for special purposes.
DECdtm	DECdtm services provide for complete and consistent executions of distributed transactions. DECdtm services coordinate distributed transactions by using the two-phase commit protocol, and by implementing special logging and communication techniques. DECdtm services do the following: • Start transactions • End transactions • Abort transactions
Event Flag	A process can use event flags to synchronize sequences of operations in a program. Event flag services clear, set, and read event flags, and place a process in a wait state pending the setting of an event flag or flags.
File Management	File management services provide searching and parsing operations of file specifications and management of the default directory specification.

‡AXP specific

(continued on next page)

Table 10–2 (Cont.) Functional Groups of System Services

Services Group	Function
Input/Output	I/O services perform input and output operations directly, rather than through the file handling services of RMS. I/O services do the following: • Perform logical, physical, and virtual input and output operations • Format output lines converting binary numeric values to ASCII strings and substituting variable data in ASCII strings • Perform network operations • Send messages to system processes
Lock Management	Lock management services let cooperating processes synchronize their access to shared resources.
Logical Names	Logical name services provide a generalized technique for maintaining and accessing character string logical name and equivalence name pairs. Logical names can provide device independence for system and application program input and output operations.
Memory Management	Memory management services provide ways to use the virtual address space available to a program. Included are services that do the following: • Allow an image to increase or decrease the amount of virtual memory data available • Control the paging and swapping of virtual memory • Create and access files in memory that contain shareable code or data
Process Control	Process control services let you create, delete, and control the execution of processes.
Process Information	Process information services let you obtain information about processes.
Security	The security services provide various mechanisms that you can use to enhance the security of OpenVMS operating systems.
Timer and Time Conversion	Timer services schedule program events for a particular time of the day or after a specified interval of time has elapsed. The time conversion services provide a way to obtain and format binary time values for use with the timer services.

Table 10–3 summarizes the system services according to their functions.

Table 10–3 Summary of System Services

Service	Function
AST Services	
$DCLAST	Queues an AST for the calling access mode or for a less privileged access mode.
$SETAST	Enables or disables the delivery of asynchronous system traps (ASTs) for the access mode from which the service call is issued.
$SETPRA	Establishes a routine to receive control after a power recovery is detected.
$SYNCH	Checks the completion status of a system service that completes asynchronously.
Cluster Event Notification Services	
‡$CLRCLUEVT	Removes one or more notification requests previously established by a call to $SETCLUEVT.
‡$SETCLUEVT	Establishes a request for notification when a cluster configuration event occurs.
‡$TSTCLUEVT	Simulates the occurrence of a cluster configuration event to test the functionality of the notification AST.
Condition-Handling Services	
$DCLCMH	Specifies the address of a routine to receive control when a Change Mode to User or Change Mode to Supervisor instruction trap occurs.
‡$GOTO_UNWIND	Unwinds the call stack.
†$RELEASE_VP	Terminates the current process's status as a vector consumer.
†$RESTORE_VP_EXCEPTION	Restores the saved exception state of the vector processor.
†$RESTORE_VP_STATE	Allows an AST routine or condition handler to restore the vector state of the mainline routine.
†$SAVE_VP_EXCEPTION	Saves the pending exception state of the vector processor.
$SETEXV	Assigns a condition handler address to the primary, secondary, or last chance exception vectors, or removes a previously assigned handler address from any of these three vectors.
$UNWIND	Unwinds the procedure call stack.
DECdtm Services	
$ABORT_TRANS (and $ABORT_TRANSW)	Ends a transaction by aborting it (and wait).
$END_TRANS (and $END_TRANSW)	Ends a transaction by attempting to commit it, and returns the outcome of the transaction (and wait).

†VAX specific
‡AXP specific

(continued on next page)

Table 10–3 (Cont.) Summary of System Services

Service	Function
DECdtm Services	
$START_TRANS (and $START_TRANSW)	Starts a new transaction (and wait).
Event Flag Services	
$ASCEFC	Associates a named common event flag cluster with a process to execute the current image and to be assigned a process-local cluster number for use with other event flag services.
$CLREF	Clears (sets to 0) an event flag in a local or common event flag cluster.
$DACEFC	Releases the calling process's association with a common event flag cluster.
$DLCEFC	Marks a permanent common event flag cluster for deletion.
$READEF	Returns the current status of all 32 event flags in a local or common event flag cluster and indicates whether the specified event flag is set or clear.
$SETEF	Sets an event flag in a local or common event flag cluster.
$WAITFR	Tests a specific event flag and returns immediately if the flag is set. Otherwise, the process is placed in a wait state until the event flag is set.
$WFLAND	Allows a process to specify a set of event flags for which it wants to wait.
$WFLOR	Allows a process to specify a set of event flags for which it wants to wait.
File Management Services	
$FILESCAN	Searches a string for a file specification and parses the components of that file specification.
$SETDDIR	Allows you to read and change the default directory string for the process.
I/O Services	
$ALLOC	Allocates a device for exclusive use by a process and its subprocesses.
$ASSIGN	Provides a process with an I/O channel so that input /output operations can be performed on a device, or establishes a logical link with a remote node on a network.
$BRKTH (and $BRKTHW)	Sends a message to one or more terminals (and wait).
$CANCEL	Cancels all pending I/O requests on a specified channel.
$CREMBX	Creates a virtual mailbox device named MBA*n* and assigns an I/O channel to it.
$DALLOC	Deallocates a previously allocated device.
$DASSGN	Deassigns (releases) an I/O channel previously acquired using the Assign I/O Channel ($ASSIGN) service.

(continued on next page)

Table 10–3 (Cont.) Summary of System Services

Service	Function
I/O Services	
$DELMBX	Marks a permanent mailbox for deletion.
$DEVICE_SCAN	Returns the names of all devices that match a specified set of search criteria.
$DISMOU	Dismounts a mounted volume or volume sets.
†$DNS (AND $DNSW)	Allows client applications to store resource names and addresses (and wait).
$FAO	Converts a binary value into an ASCII character string in decimal, hexadecimal, or octal notation and returns the character string in an output string, and inserts variable character string data into an output string.
$FAOL	Provides an alternate method for specifying input parameters when calling the $FAO system service.
$GETDVI (and $GETDVIW)	Returns information related to the primary and secondary device characteristics of an I/O device (and wait).
$GETMSG	Returns message text associated with a given message identification code into the caller's buffer.
$GETQUI (and $GETQUIW)	Returns information about queues and the jobs initiated from those queues (and wait).
$INIT_VOL	Formats a disk or magnetic tape volume and writes a label on the volume.
$MOUNT	Mounts a tape, disk volume, or volume set and specifies options for the mount operation.
$PUTMSG	Writes informational and error messages to processes.
$QIO (and $QIOW)	Queues an I/O request to a channel associated with a device (and wait).
$RMSRUNDWN	Closes all files opened by OpenVMS RMS for the image or process and halts I/O activity.
$SNDERR	Writes a user-specified message to the system error log file, preceding it with the date and time.
$SNDJBC (and $SNDJBCW)	Creates, stops, and manages queues and the batch and print jobs in those queues (and wait).
$SNDOPR	Performs the following functions: • Sends a user request to operator terminals • Sends a user cancellation request to operator terminals • Sends an operator reply to a user terminal • Enables an operator terminal • Displays the status of an operator terminal • Initializes the operator log file

†VAX specific

(continued on next page)

Table 10–3 (Cont.) Summary of System Services

Service	Function
Lock Management Services	
$DEQ	Dequeues (unlocks) granted locks; dequeues the sublocks of a lock; or cancels an ungranted lock request.
$ENQ (and $ENQW)	Queues a new lock or lock conversion on a resource (and wait).
$GETLKI (and $GETLKIW)	Returns information about the lock database on a system (and wait).
Logical Name Services	
$CRELNM	Creates a logical name and specifies its equivalence names.
$CRELNT	Creates a process-private or shareable logical name table.
$DELLNM	Deletes all logical names with the specified name at the specified access mode or outer access mode, or it deletes all the logical names with the specified access mode or outer access mode in a specified table.
$TRNLNM	Returns information about a logical name.
Memory Management Services	
$ADJSTK	Modifies the stack pointer for a less privileged access mode.
$ADJWSL	Adjusts a process's current working set limit by the specified number of pages (on VAX systems) or pagelets (on AXP systems) and returns the new value to the caller.
$CRETVA	Adds a range of demand-zero allocation pages (on VAX systems) or pagelets (on AXP systems) to a process's virtual address space for the execution of the current image.
$CRMPSC	Allows a process to associate (map) a section of its address space with (1) a specified section of a file (a disk file section) or (2) specified physical addresses represented by page frame numbers (a page frame section).
$DELTVA	Deletes a range of addresses from a process's virtual address space.
$DGBLSC	Marks an existing permanent global section for deletion.
$EXPREG	Adds a specified number of new virtual pages to a process's program region or control region for the execution of the current image.
$LCKPAG	Locks a page or range of pages in memory.
$LKWSET	Locks a range of pages in the working set; if the pages are not already in the working set, it brings them in and locks them.
$MGBLSC	Establishes a correspondence between pages (maps) in the virtual address space of the process and physical pages occupied by a global section.

(continued on next page)

Table 10–3 (Cont.) Summary of System Services

Service	Function
Memory Management Services	
$PURGWS	Removes a specified range of pages from the current working set of the calling process to make room for pages required by a new program segment.
$SETPRT	Allows a process to change the protection on a page or range of pages.
$SETSTK	Allows a process to change the size of its supervisor, executive, and kernel stacks by altering the values in the stack limit and base arrays held in P1 (per-process) space.
$SETSWM	Allows a process to control whether it can be swapped out of the balance set.
$ULKPAG	Unlocks pages that were previously locked in memory by the Lock Pages in Memory ($LCKPAG) service.
$ULWSET	Unlocks pages that were previously locked in the working set by the Lock Pages in Working Set ($LKWSET) service.
$UPDSEC (and $UPDSECW)	Writes all modified pages in an active private or global section back into the section file on disk (and wait).
Process Control Services	
$CANEXH	Deletes an exit control block from the list of control blocks for the calling access mode.
$CANWAK	Removes all scheduled wakeup requests for a process from the timer queue, including those made by the caller or by other processes.
$CREPRC	Creates a subprocess or detached process on behalf of the calling process.
$DCLEXH	Declares an exit handling routine that receives control when an image exits.
$DELPRC	Allows a process to delete itself or another process.
$EXIT	Initiates image rundown when the current image in a process completes execution.
$FORCEX	Causes an Exit ($EXIT) service call to be issued on behalf of a specified process.
$HIBER	Allows a process to make itself inactive but to remain known to the system so that it can be interrupted; for example, to receive ASTs.
$PROCESS_SCAN	Creates and initializes a process context that is used by $GETJPI to scan processes on the local system or across the nodes in a VMScluster system.
$RESCHED	Requests reschedule of a process.
$RESUME	Causes a process previously suspended by the Suspend Process ($SUSPND) service to resume execution or cancels the effect of a subsequent suspend request.
$SCHDWK	Schedules the awakening (restarting) of a process that has placed itself in a state of hibernation with the Hibernate ($HIBER) service.

(continued on next page)

Table 10–3 (Cont.) Summary of System Services

Service	Function
Process Control Services	
$SETPRI	Changes the base priority of the process.
$SETPRN	Allows a process to establish or to change its own process name.
$SETPRV	Enables or disables specified privileges for the calling process.
$SETRWM	Allows a process to specify what action system services should take when system resources required for their execution are unavailable.
$SETSHLV	Controls whether a process automatically unshelves files.
$SUSPND	Allows a process to suspend itself or another process.
$WAKE	Activates a process that has placed itself in a state of hibernation with the Hibernate ($HIBER) service.
Process Information Services	
‡$CHECK_FEN	Indicates whether floating point is enabled.
$GETJPI (and $GETJPIW)	Returns information about one or more processes on the system or across the VMScluster system (and wait).
$GETSYI (and $GETSYIW)	Returns information about a local OpenVMS system or other OpenVMS systems in a VMScluster (and wait).
‡$GET_ALIGN_FAULT_DATA	Obtains data from user image alignment fault buffer.
‡$GET_ARITH_EXCEPTION	Returns information about the exception context for a given arithmetic exception.
‡$GET_SYS_ALIGN_FAULT_DATA	Obtains data from system image alignment fault buffer.
‡$IEEE_SET_FP_CONTROL	Modifies the IEEE floating-point control register and, optionally, returns the previous register value.
‡$INIT_SYS_ALIGN_FAULT_REPORT	Initializes system process alignment fault reporting.
‡$PERM_DIS_ALIGN_FAULT_REPORT	Disables user process alignment fault reporting.
‡$PERM_REPORT_ALIGN_FAULT	Initializes user process alignment fault reporting.
‡$START_ALIGN_FAULT_REPORT	Initializes user image alignment fault reporting.
‡$STOP_ALIGN_FAULT_REPORT	Disables user image alignment fault reporting.
‡$STOP_SYS_ALIGN_FAULT_REPORT	Disables systemwide alignment fault reporting.
System Security Services	
$ADD_HOLDER	Adds holder record to rights database.
$ADD_IDENT	Adds identifier to rights database.
†$ADD_PROXY	Adds a new proxy to, or modifies an existing proxy in, the proxy database.
$ASCTOID	Translates identifier name to binary value.
$AUDIT_EVENT (and $AUDIT_EVENTW)	Appends an error message to the audit log file (and wait).

†VAX specific
‡AXP specific

(continued on next page)

Table 10–3 (Cont.) Summary of System Services

Service	Function
System Security Services	
$CHECK_ACCESS	Invokes system access protection check on behalf of another user.
$CHECK_PRIVILEGE (and $CHECK_ PRIVILEGEW)	Determines whether the caller has the specified privileges or identifiers (and wait).
$CHKPRO	Invokes system access protection check.
$CMEXEC	Changes the access mode of the calling process to executive mode.
$CMKRNL	Changes the access mode of the calling process to kernel mode.
$CREATE_RDB	Initializes a rights database.
$CREATE_USER_PROFILE	Returns and encoded security profile for a user.
†$DELETE_INTRUSION	Searches for and deletes all records in the intrusion database matching the caller's specifications.
†$DELETE_PROXY	Deletes an existing proxy or removes the default user or the local user from an existing proxy in the proxy database.
†$DISPLAY_PROXY	Returns information about one or more existing proxies.
$ERAPAT	Generates a security erase pattern.
$FIND_HELD	Returns identifiers held by a holder in rights database.
$FIND_HOLDER	Returns holders of an identifier in rights database.
$FINISH_RDB	Deallocates record stream and clears context value when searching the rights database.
$FORMAT_ACL	Formats ACE into a text string.
$FORMAT_AUDIT	Converts a security auditing event message from binary to ASCII.
$GETUAI	Returns authorization information about a specified user.
$GET_SECURITY	Returns information about security characteristics of a selected object.
$GRANTID	Adds identifier to process or system rights list.
$HASH_PASSWORD	Applies a hash algorithm to an ASCII password string and returns a quadword hash value that represents the encrypted password.
$IDTOASC	Translates identifier value to its identifier name.
$MOD_HOLDER	Modifies holder record in rights database.
$MOD_IDENT	Modifies identifier record in rights database.
$MTACCESS	Controls magnetic tape access.
$PARSE_ACL	Converts text ACE into binary format.
$REM_HOLDER	Deletes holder record from identifier's list of holders in rights database.
$REM_IDENT	Deletes identifier and all holders of that identifier from rights database.

†VAX specific

(continued on next page)

Table 10–3 (Cont.) Summary of System Services

Service	Function
System Security Services	
$REVOKID	Removes identifier from process or system rights list.
†$SCAN_INTRUSION	Scans the intrusion database for suspects or intruders during a login attempt, audits login failures and updates records, or adds new records to the intrusion database.
$SETDFPROT	Allows you to read and write the default file protection for the process.
$SETUAI	Modifies the user authorization file (UAF) record for a specified user.
$SET_RESOURCE_DOMAIN	Controls association between calling process and resource domains.
$SET_SECURITY	Modifies the security characteristics of a security object.
†$SHOW_INTRUSION	Searches for and returns information about records in the intrusion database matching the caller's specifications.
$SUBSYSTEM	Saves or restores the process image rights for the current protected subsystem.
†$VERIFY_PROXY	Verifies that a proxy exists and returns a valid local user for the caller to use to create a local login.
Timer and Time Conversion Services	
$ASCTIM	Converts an absolute or delta time from 64-bit system time format to an ASCII string.
$ASCUTC	Converts an absolute time from 128-bit UTC format to an ASCII string.
$BINTIM	Converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SETIMR) or Schedule Wakeup ($SCHDWK) service.
$BINUTC	Converts an ASCII string to an absolute time value in the 128-bit UTC format.
$CANTIM	Cancels all or a selected subset of the Set Timer requests previously issued by the current image executing in a process.
$GETTIM	Returns the current system time in a 64-bit format.
$GETUTC	Returns the current time in 128-bit UTC format.
$NUMTIM	Converts an absolute or delta time from 64-bit system time format to binary integer date and time values.
$NUMUTC	Converts an absolute 128-bit binary time into its numeric components. The numeric components are returned in local time.
$SETIME	Changes the value of, or recalibrates, the system time.
$SETIMR	Sets the timer to expire at a specified time.

†VAX specific

(continued on next page)

Table 10–3 (Cont.) Summary of System Services

Service	Function
Timer and Time Conversion Services	
$TIMCON	Converts Coordinated Universal Time (UTC) to 64-bit system format or 64-bit system format to UTC based on the value of the convert flag.

10.4 Utility Routines

Some OpenVMS utilities can be invoked either at the DCL command level or through a callable interface. Other utilities have only a callable interface. A utility with a callable interface means that a program can invoke the utility, execute utility-specific functions, and exit the utility. Table 10–4 summarizes the utility routine groups.

For complete information on the utility routines, and a routine-by-routine listing, refer to the *OpenVMS Utility Routines Manual*.

Table 10–4 Utility Routine Summary

Routine Prefix	Utility/Facility	Description
ACL$	Access Control List (ACL) Editor	Creates and maintains access control lists. ACLs control access to files, devices, global sections, logical name tables, or mailboxes.
CLI$	Command Definition utility (CDU)	Processes command strings using information from a command table; use in conjunction with new commands created by CDU.
CONV$	Convert and Convert/Reclaim (CONV) utility	Convert utility copies records from one or more files to an output file while changing format and file organization. Convert/Reclaim utility reclaims empty buckets so that new records can be written.
DCX$	Data Compression/Expansion (DCX) facility	Analyzes and compresses data records; expands data records that have been compressed.
EDT$	EDT Editor	Invokes EDT and either edits a file from the program or allows interactive editing.
FDL$	File Definition Language utility (FDL)	Specifies RMS options for a file, creates a file, opens a file, closes a file, connects a file, allocates RMS control blocks, fills in control blocks, and deallocates control blocks.
LBR$	Librarian utility (LBR)	Maintains any type of library.
PSM$	Print Symbiont Modification (PSM) facility	Modifies the OpenVMS print symbiont (or, if necessary, can be used to create user-written symbiont).
SMB$	Symbiont/Job-Controller Interface (SMB) facility	Provides the symbiont-job controller interface for user-written symbionts.
SOR$	Sort/Merge (SOR) utility	Integrates a sort or merge operation into a program application.
TPU$	DEC Text Processing utility (DECTPU)	Invokes and uses DECTPU functions within a program written in any VAX programming language.

10.5 OpenVMS Record Management Services

OpenVMS Record Management Services (RMS) assists user programs in processing and managing files and their contents. RMS is a collection of routines that give programmers a device-independent method for storing, retrieving, and modifying data. RMS allows you to create a new file, access an existing file, extend disk space for a file, close a file, obtain file characteristics as well as to get, locate, insert, update, and delete records.

Specifically, RMS provides the following:

- Disk file organizations—sequential, relative, and indexed
- Record formats—fixed length and variable length for each file organization
- Record access modes—sequential, by key value, by relative record number, by record file address

For information about using RMS, refer to the *OpenVMS Record Management Services Reference Manual*.

RMS supports unit-record devices such as terminals and printers, but it is designed primarily to provide a comprehensive software interface to mass-storage devices such as disk and magnetic tape drives.

10.5.1 RMS File Control Blocks

Control blocks are used to provide input to services and to accept output from services.

The following control blocks support services that manipulate files:

- File access block (FAB)

  The FAB control block includes file specification information, file characteristics (file organization, record type, allocation information, and so forth), and run-time access options (file processing information and addresses of other control blocks with additional information.)

- Optional name block (NAM)

  The NAM control block includes supplemental information to the FAB.

- Optional extended attribute block (XAB)

  The XAB control block includes file characteristics that supersede or supplement the FAB control block.

10.5.2 RMS Record Control Blocks

To support services that manipulate with records, there are two record control blocks, as follows:

- Record access block (RAB)

  The RAB control block includes the address of the related FAB control block, the address of input and output record buffers, general I/O buffer type and size, how the records will be accessed, and other record information.

- Extended attribute block (XAB)

  The XAB control block includes record characteristics that can supersede or supplement information in the RAB control block.

10.5.3 RMS Macros

RMS uses macros provided in the system macro library to perform the following tasks:

- Initialize control blocks at assembly time (allocates space within the program image for the control block, defines the symbolic names for a control block, initializes certain control block fields with internally used values, initializes specified control block fields with user-specified values, and initializes certain fields with system-supplied default values).
- Define control block symbolic names at assembly time (does not allocate or initialize the control block).
- Set specified fields with user-specified values at run time.
- Invoke services at run time.

Table 10–5 lists each control block and its associated macros.

Table 10–5 User Control Blocks

Control (Block)	Macro (Name)	Function
FAB		Describes a file and contains file-related information.
	$FAB	Allocates storage for a FAB and initializes certain FAB fields; also defines symbolic offsets for a FAB.
	$FABDEF	Defines symbolic offsets for a FAB.
	$FAB_STORE	Moves specified values into a previously allocated and initialized FAB.
NAM		Contains file specification information beyond that in the FAB.
	$NAM	Allocates storage for a NAM and initializes certain NAM fields; also defines symbolic offsets for a NAM.
	$NAMDEF	Defines symbolic offsets for a NAM.
	$NAM_STORE	Moves specified values into a previously specified and allocated NAM.
RAB		Describes a record stream and contains record-related information.
	$RAB	Allocates storage for a RAB and initializes certain RAB fields; also defines symbolic offsets for a RAB.
	$RABDEF	Defines symbolic offsets for a RAB.
	$RAB_STORE	Moves specified values into a previously specified and allocated RAB.
XABxxx[1]		Contains file attribute information beyond that in the FAB. For XABTRM, contains information beyond that in the RAB.
	$XABxxx	Allocates and initializes an XAB.
	$XABxxxDEF	Defines symbolic offsets for an XABxxx.

[1]The xxx is a 3-character mnemonic.

(continued on next page)

Table 10–5 (Cont.) User Control Blocks

Control (Block)	Macro (Name)	Function
	$XABxxx_STORE	Moves specified values into a previously specified and allocated XABxxx.

10.5.4 OpenVMS Record Management Services Utilities

The RMS utilities are as follows:

- Analyze/RMS_File utility (ANALYZE/RMS_FILE)
- Convert and Convert/Reclaim utilities (CONVERT and CONVERT/RECLAIM)
- Create/FDL utility (CREATE/FDL)
- Edit/FDL utility (EDIT/FDL)

You can use these independently of RMS, or in conjunction with RMS, to build data files and to maintain files.

ANALYZE/RMS_FILE

With ANALYZE/RMS_FILE, you can analyze the internal structure of a RMS file in the following manner:

- Examine the structure of a file, and interactively check the structure to assess if it is properly designed for the application
- Generate a statistical report on the file's structure and use
- Generate an FDL file from a data file
- Generate a summary report on the file's structure and use

The interactive feature of this utility includes several commands to traverse the structure of an RMS file and examine specific data buckets and bytes of a record. This utility can also check the file and generate a report listing any errors found in the file.

ANALYZE/RMS_FILE commands help you move around the RMS file easily. You can move the structure pointer to the beginning and end of the file structure, up and down levels, to the first and last nodes, and to a specific bucket (or record) of an indexed or relative file.

CONVERT and CONVERT/RECLAIM

CONVERT copies one or more records from a file to another file, while changing the record format and file organization. CONVERT/RECLAIM reclaims empty bucket space in the file to allow new records to be written to it.

CONVERT/RECLAIM works only with Prolog 3 indexed files. You should use CONVERT/RECLAIM when new records no longer need a primary key associated with the deleted record.

In conjunction with changing record format and file organization, you can use CONVERT to complete the following functions:

- Reformat indexed files where many records have been inserted and deleted. New record file addresses are established for the records.
- Create a new output file with the same or different file characteristics.
- Add new records to the end of an existing sequential file.

- Merge new records into an existing indexed file.
- Convert carriage control to one of four formats (CARRIAGE_RETURN, FORTRAN, PRINT, and NONE).

CONVERT/RECLAIM does not change file format or organization when it reclaims empty bucket space. It deletes the old pointers to a bucket and puts it on a list of free buckets. When new records that need a new bucket are added, RMS goes to the free bucket list and sets up pointers to a bucket from the list. CONVERT/RECLAIM preserves the file addresses of the records.

For a complete description of using CONVERT and CONVERT/RECLAIM, refer to the *OpenVMS Record Management Utilities Reference Manual*.

Command qualifiers allow you to modify CONVERT in the following ways:

- Append records to an existing file
- Create a new file with or without using an FDL file
- Access or insert records in an indexed file
- Pad short records or truncate long records
- Sort a file according to the primary key
- Check all read and write operations

CREATE/FDL and EDIT/FDL

The File Definition Language (FDL) helps you define specifications for data files. FDL is used within the context of the File Definition Language facility, and consists of the utilities CREATE/FDL and EDIT/FDL. An FDL file consists of a collection of file attributes grouped into related sections. EDIT/FDL invokes the FDL editor to create a new FDL file. The types of attributes you specify are the following:

- File processing operations specified using the following keywords: BLOCK_IO (enabling RMS read and write operations), DELETE, GET, PUT, RECORD_IO (enabling mixed record I/O and block I/O), TRUNCATE, UPDATE
- Allocation of area and key analysis sections (for indexed files only)
- Creation or manipulation of RMS specific areas in an indexed file
- Application-dependent run-time attributes
- Date and time for certain file characteristics
- File processing and file-related characteristics
- Key attributes
- Secondary attributes that define records specified using the following keywords: BLOCK_SPAN, CARRIAGE_CONTROL, CONTROL_FIELD, FORMAT, and SIZE
- Sharing of the data file
- System identification information

CREATE/FDL uses the specifications in an existing FDL file to create a new empty data file. The *OpenVMS Record Management Utilities Reference Manual* describes how to use the FDL utility and lists each of the commands.

With FDL commands, you can add, modify, or delete lines to a file; enable assistance with the design and optimization of a data file; specify the number of keys in an indexed file; specify the output file; divide an indexed file into a specified number of areas; and choose between smaller buffer and flatter files.

For complete information about RMS, see the *OpenVMS Record Management Utilities Reference Manual*.

11

Additional Programming Utilities

In addition to the utilities already described in this manual, the OpenVMS operating system also includes the following programming utilities that you can use to develop application programs:

- Patch utility
- National Character Set (NCS) utility

This chapter briefly describes the features of these programming utilities and references the appropriate manuals for more information about how to use them.

11.1 Patch Utility (VAX Only)

VAX

On VAX systems, the Patch utility (PATCH) allows you to make changes to an image file in the form of patches. You can then run the new version of the image without having to recompile (or reassemble) and relink the program. You can enter PATCH commands interactively or use them in a command procedure to execute interactively or in batch mode. You can use PATCH with any language supported by the OpenVMS operating system as long as the image was generated by the linker.

The input image can be a shareable image, a device driver image, or any other executable image. Consider the following restrictions when you use PATCH:

- You can specify only universal symbols when patching a shareable image.
- You can use the default patch area to patch position-independent shareable images.
- You must use a user-defined patch area to patch position-dependent images.

PATCH does not alter the input image. It creates a copy of the image, makes changes to the copy, and leaves the original image unaltered.

With the PATCH commands, you can modify the image as follows:

- Add or delete instructions or data
- Replace instructions or data
- Allocate space for the patch area
- Create a command procedure of PATCH commands
- Assign an engineering change order-level to the changes
- View the contents of a particular location
- Display the modules in the image
- Apply the patch to the image

For more information about the Patch Utility, refer to the *OpenVMS VAX Patch Utility Manual*.♦

11.2 National Character Set Utility

The National Character Set (NCS) utility allows you to define and use collating sequences and conversion functions. With collating sequences, you can alter the standard sorting sequence for a particular use (usually for a national character set). Using conversion functions, you can define case conversions or character representations that you subsequently use in the collating sequence.

The collating sequences and conversions are stored in an NCS library that you manage using NCS. The command qualifiers allow you to create the library; insert, replace, and delete modules; list module information; and view specified modules.

Eight NCS callable routines allow you to access the collating sequences and conversions stored in an NCS library from your program.

For more information about using the NCS utility and its callable routines, refer to the *OpenVMS National Character Set Utility Manual*.

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

X

Z