VDE
Guide to Using VDE

Order Number: ??–?????–??

11-DEC-1998 17:47:29.55

This is a guide for the development environment package used within OpenVMS engineering.

This guide covers the common components and normal operations of the OpenVMS Development Environment system, including an introduction and overview of the OpenVMS Development Environment (VDE).

The OpenVMS Development Environment package is used within various Digital engineering organizations, including the OpenVMS group.

Revision/Update Information:	This manual supersedes the Guide to Using VDE, Version 1.4-n
Software Version:	VDE Version 1.5-0
Operating System Version:	OpenVMS Alpha Version 6.2 OpenVMS VAX Version 6.2

Digital Equipment Corporation
Maynard, Massachusetts

11-DEC-1998 17:47:29.55

Copyright ©December 1997

The following are trademarks of Digital Equipment Corporation: Bookreader, DEC Notes, DECdirect, DECwindows, Digital, OpenVMS, OpenVMS Cluster, VAX, VAX Document, VMScluster, VMS, and the DIGITAL logo.

The following are third-party trademarks:

Motif, OSF, OSF/1, OSF/Motif, and Open Software Foundation are registered trademarks of the Open Software Foundation, Inc.

All other trademarks and registered trademarks are the property of their respective holders.

Contents

PREFACE ix

CHAPTER 1 INTRODUCTION 1–1

1.1 WHAT IS VDE? 1–1

1.2 VDE AND THE OPENVMS VAX LIBRARY 1–2

1.3 MODULES AND FACILITIES 1–3

1.4 DEVELOPMENT STREAMS 1–3

1.5 GENERATIONS 1–4

1.6 ARCHITECTURES 1–4

1.7 PROPOGATION AND FOLDING 1–5

1.8 CUSTOMIZATION USING SCRIPTS 1–5

1.9 CONVENIENCE FEATURES 1–5
1.9.1 Kept Subprocess 1–5
1.9.2 Initialization Files 1–6
1.9.3 Selecting a Text Editor 1–7
1.9.4 Grouping Reservations into Sessions 1–7
1.9.5 Defining Defaults Using Contexts 1–8
1.9.6 Default Libraries, Mnemonics 1–8

CHAPTER 2 RESERVING AND REPLACING MODULES 2–1

2.1 IDENTIFYING MODULE GENERATIONS 2–1

Contents

2.2 RESERVING MODULES 2–2
2.2.1 Concurrent Reservations 2–7
2.2.2 Queued Reservations 2–9

2.3 CREATING NEW MODULES 2–9

2.4 FETCHING MODULES 2–11

2.5 UNRESERVING MODULES 2–13

2.6 DISPLAYING RESERVATIONS 2–13

2.7 REPLACING MODULES 2–15

2.8 CHANGE PROPAGATION AND FOLD RECORDS 2–23

2.9 DISPLAYING QUEUED REPLACEMENTS 2–24

2.10 CANCELLING QUEUED REPLACEMENTS 2–26

CHAPTER 3 DISPLAYING INFORMATION 3–1

3.1 DISPLAYING FACILITIES 3–1

3.2 DISPLAYING MODULES 3–2

3.3 DISPLAYING GENERATION HISTORIES 3–4

3.4 DISPLAYING DIFFERENCES BETWEEN GENERATIONS 3–9

3.5 DISPLAYING DEVELOPMENT STREAMS 3–11

3.6 DISPLAYING USERS 3–13

CHAPTER 4 REVIEWING QUEUED REPLACEMENTS 4–1

4.1 REVIEWING MODULES QUEUED FOR REPLACEMENT 4–1

4.2 ACCEPTING OR REJECTING A QUEUED REPLACEMENT 4–7

CHAPTER 5 PROJECT LEADER OPERATIONS 5–1

5.1 PERFORMING QUEUED REPLACEMENTS 5–1

5.2 INSERTING GENERATIONS 5–3

5.3 REMOVING GENERATIONS 5–4

5.4 DELETING GENERATIONS 5–6

5.5 MARKING MODULE GENERATIONS OBSOLETE 5–7

5.6 SPECIFYING REVIEWER AND NOTIFICATION LISTS 5–7

5.7 CREATING NEW STREAMS 5–8
5.7.1 Creating Build Streams 5–9
5.7.2 Creating Release Streams 5–10
5.7.3 Example Stream Creations 5–12

5.8 TAILORING PROCESSING WITH SCRIPTS 5–13

CHAPTER 6 ESTABLISHING A VDE LIBRARY 6–1

6.1 CREATING A VDE LIBRARY 6–1
6.1.1 The VDE$UPLOAD CMS Upload Utility 6–2
6.1.2 The CREATE LIBRARY Command 6–2
6.1.3 Examples 6–4

Contents

6.2 OPTIONAL LIBRARY ATTRIBUTES AND SETTINGS 6–7
6.2.1 Database Placement 6–7
6.2.2 Standard User Privileges 6–8
6.2.3 Automatic Addition of Users to Database 6–9
6.2.4 Conversion of Library Format 6–10
6.2.5 Sending Mail Notifications 6–11
6.2.6 Allowing Deletion of Library Entities 6–12
6.2.7 CMS History and Notes Defaults 6–12
6.2.8 Configuring Architectures 6–13
6.2.9 Miscellaneous Attributes 6–13

6.3 LIBRARY DIRECTORY STRUCTURE 6–13
6.3.1 Delta File Directories 6–14
6.3.2 Staging Area Directories 6–15
6.3.3 Development Stream Directories 6–17

6.4 ADDING USERS TO THE LIBRARY 6–21
6.4.1 The CREATE USER Command 6–21
6.4.2 Automatic Addition of Users 6–22

6.5 CREATING STREAMS 6–23
6.5.1 Stream Directory Tree 6–24
6.5.2 Scripts to Populate New Streams 6–24
6.5.3 Stream Successor Relationships 6–24

6.6 CREATING FACILITIES 6–24

6.7 POPULATING A FACILITY WITH MODULES 6–24
6.7.1 Getting Source Modules from a Directory 6–25
6.7.2 Getting Source Modules from a CMS Library 6–25
6.7.3 Example CMS Upload Procedures 6–26
6.7.4 Adding Individual Source Modules 6–27

CHAPTER 7 LIBRARY MAINTENANCE OPERATIONS 7–1

7.1 CREATING SCT NOTES CONFERENCES 7–1

7.2 TAKING VDE OFFLINE 7–1

7.3 RESOLVING DATABASE PROBLEMS 7–2
7.3.1 User Process Deadlocks 7–2
7.3.2 Freeze in Rdb 7–3
7.3.3 Rdb process dumps 7–3
7.3.4 Rdb run-unit journal errors 7–3
7.3.4.1 Missing run-unit journal area • 7–4
7.3.4.2 Rdb run-unit journal diskquota errors • 7–4
7.3.4.3 Rdb recovery process dumps • 7–5
7.3.4.4 Missing run-unit journal files • 7–6

7.4 DATABASE BACKUP AND RECOVERY PROCEDURES 7–7
7.4.1 Overview of Backup Procedures 7–7
7.4.2 Placement of Backup Files and Directories 7–8
7.4.3 After-Image Journalling 7–8
7.4.4 Creating the Database Backup Procedure 7–8
7.4.5 Creating the Database Restore Procedure 7–9
7.4.6 Backing up the VDE Database 7–9
7.4.7 Restoring the VDE Database 7–11
7.4.8 Recovering from a Lost Journal File 7–13
7.4.9 Associated Rdb commands 7–15
7.4.10 Recovering Lost CMS Libraries 7–15
7.4.11 Moving to Larger CMS Disks 7–17
7.4.12 Moving to Larger Rdb Disks 7–18

7.5 PURGING THE STAGING AREA DIRECTORIES 7–19

7.6 OFFLOADING LARGE FILES 7–20

7.7 DEFRAGMENTING THE CMS LIBRARY 7–23

7.8 RELOCATING RDB COMPONENTS 7–24

7.9 RECREATING RDB COMPONENTS 7–24

CHAPTER 8 BUILDING YOUR SOFTWARE SYSTEM 8–1

8.1 BUILD DEPENDENCY GRAPHS 8–1

8.2 CENTRAL BUILD ALGORITHM 8–1

8.3 CREATING BUILD JOBS 8–1

8.4 DISPLAYING BUILD JOBS 8–1

8.5 RUNNING BUILD JOBS 8–1
8.5.1 Starting Build Jobs 8–1
8.5.2 Stopping Build Jobs 8–2
8.5.3 Suspending Build Jobs 8–2
8.5.4 Waiting for Build Jobs 8–2

8.6 SCRIPTS FOR BUILD STEP EXECUTION 8–2

8.7 PERFORMING INDIVIDUAL BUILD STEPS 8–2
8.7.1 Copy Steps 8–2
8.7.2 Compile Steps 8–2
8.7.3 Link Steps 8–2

8.8 INITIALIZING THE DEPENDENCY GRAPH 8–2

CHAPTER 9 ARCHITECTURE SUPPORT 9–1

9.1 ARCHITECTURES IN THE SOURCE CODE LIBRARY ROOTS 9–1

9.2 COMMANDS TO SPECIFY MULTIPLE ARCHITECTURES 9–2

9.3 CREATING ARCHITECTURE-SPECIFIC MODULES 9–2

9.4 OPERATION OF EXISTING COMMANDS 9–3

9.5 CHANGE PROPAGATION AND FOLD RECORDS 9–5

CHAPTER 10 THE GRAPHICAL USER INTERFACES 10–1

10.1 MOTIF INTERFACE 10–1
10.1.1 Motif Version Information 10–1
10.1.2 Motif Library Selection 10–2
10.1.3 Motif Library Default Selection 10–3
10.1.4 Motif Resources 10–4

10.2 WWW/MOSAIC INTERFACE 10–4

APPENDIX A TROUBLESHOOTING INFORMATION A–1

A.1 PERIODIC MAINTENANCE A–1

A.2 CMS TROUBLESHOOTING A–1
A.2.1 CMS Library Locked A–1
A.2.2 CMS 3.8-2 and Indexed Files A–2
A.2.3 Recovering Lost CMS Libraries A–2
A.2.4 Moving to Larger CMS Disks A–2

A.3 VMS_MENU TROUBLESHOOTING A–2

A.4 VDE SPAWNED PROCESSING TROUBLESHOOTING A–3

A.5 RDB TROUBLESHOOTING A–3
A.5.1 User Process Deadlocks A–3
A.5.2 Freeze in Rdb A–3
A.5.3 Resolving Rdb process dumps A–3
A.5.4 Rdb run-unit journal errors A–3
A.5.5 Missing run-unit journal area A–3
A.5.6 Rdb run-unit journal diskquota errors A–4
A.5.7 Rdb recovery process dumps A–4
A.5.7.1 Missing run-unit journal files • A–4
A.5.8 Recovering from a Lost Journal File A–4
A.5.9 Moving to Larger Rdb Disks A–4

Contents

APPENDIX B COMPARISON OF VDE, VSC, AND VTSC B–1

INDEX

FIGURES

10–1 VDE Motif Facility Display 10–2

TABLES

2–1 Qualifiers to the RESERVE command 2–4
2–2 Qualifiers to the CREATE MODULE command 2–10
2–3 Qualifiers to the FETCH command 2–11
2–4 Qualifiers to the SHOW RESERVATION command 2–13
2–5 Qualifiers to the REPLACE command 2–19
2–6 Qualifiers to the SHOW REPLACEMENT command 2–24
2–7 Qualifiers to the CANCEL REPLACEMENT command 2–27
3–1 Qualifiers to the SHOW FACILITY command 3–1
3–2 Qualifiers to the SHOW MODULE command 3–3
3–3 Qualifiers to the SHOW GENERATION command 3–6
3–4 Qualifiers to the DIFFERENCES command 3–10
3–5 Qualifiers to the SHOW STREAM command 3–11
3–6 Qualifiers to the SHOW USER command 3–13
4–1 Qualifiers to the REVIEW REPLACEMENT command 4–2
4–2 Commands Accepted in the REVIEW REPLACEMENT Subsystem 4–2
4–3 Qualifiers to the ACCEPT REPLACEMENT and REJECT REPLACEMENT commands 4–8
5–1 Qualifiers to the PERFORM REPLACEMENT command 5–2
5–2 Qualifiers to the INSERT GENERATION command 5–4
5–3 Qualifiers to the REMOVE GENERATION command 5–5
5–4 Qualifiers to the DELETE GENERATION command 5–6
5–5 Script Commands 5–14
6–1 Delta File Disk Directories 6–14
6–2 Staging Area Disk Directories 6–16
6–3 Stream Disk Directories 6–18
B–1 VDE, VSC and VTSC B–1

Preface

Intended Audience

This manual is intended for users who want to learn more about the OpenVMS Development Environment, or about the VDE utility.

Document Structure

This document contains a guide to the OpenVMS Development Environment (VDE) package, and to the VDE command interface utility. It provides an overview of VDE, and the various commands used to fetch, reserve, replace and otherwise access and maintain the modules in a VDE library.

Note: For a basic understanding of the VDE command interface utility, and the operations commonly performed by users, one need only read Chapter 1, Chapter 2, and Chapter 3.

This guide contains the following information:

- Chapter 1 provides an introduction to the OpenVMS (OpenVMS Development Environment) system.
- Chapter 2 describes how to perform typical operations, including module reservation, module creation, and module replacement.
- Chapter 3 tells how to display various kinds of information from the source control database.
- Chapter 4 explains how to review queued module replacements.
- Chapter 9 describes operations around multiple architecture support.
- Chapter 5 describes special operations performed by project leaders and maintainers of VDE libraries.
- Chapter 7 describes various backup and recovery operations typically performed by the maintainers of VDE libraries.
- Chapter 10 describes the X Windows Motif and World Wide Web Mosaic interfaces.
- Appendix A describes how to troubleshoot some of the problems that might be encountered while using VDE.

Associated Documents

- VDE Installation and New Features Guide
- VDE Reference Manual

Conventions

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

The following conventions are used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
GOLD *x*	A sequence such as GOLD *x* indicates that you must first press and release the key defined GOLD, then press and release another key. GOLD key sequences can also have a slash (/), dash (-), or underscore (_) as a delimiter in EVE commands.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification, or in the syntax of a substring specification in an assignment statement.)
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in Bookreader versions of the manual.

italic text	Italic text emphasizes important information, indicates variables, and indicates complete titles of manuals. Italic text also represents information that can vary in system messages (for example, Internal error *number*), command lines (for example, /PRODUCER=*name*), and command parameters in text.
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
numbers	All numbers in text are assumed to be decimal, unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.
mouse	The term *mouse* refers to any pointing device, such as a mouse, a puck, or a stylus.
MB1, MB2, MB3	MB1 indicates the left mouse button, MB2 indicates the middle mouse button, and MB3 indicates the right mouse button. (The buttons can be redefined by the user.)
PB1, PB2, PB3, PB4	PB1, PB2, PB3, and PB4 indicate buttons on the puck.
SB, SB	SB and SB indicate buttons on the stylus.

1 Introduction

One of the most important tasks found in any large software engineering effort involves the coordination, tracking, maintenance and protection of the source files. OpenVMS engineering uses the OpenVMS Development Environment (VDE) package for these purposes, and for arcane reasons refers to the aggregate collection of OpenVMS source files and source libraries as the *Master Pack*.

The OpenVMS Development Environment (VDE) package is the set of tools that the OpenVMS development group uses to control and to track changes to the multiple VDE libraries that comprise the OpenVMS Master Pack.

1.1 What is VDE?

VDE is tailored for medium to large-scale source code control environments, and it provides capabilities and controls useful within the OpenVMS development environment. Among other uses, VDE is used to coordinate all access to the libraries containing the OpenVMS source code.

VDE provides various user interfaces including the VDE[1] command interface, and the Mosaic and Motif graphical user interfaces. (Information on the Motif and WWW/Mosaic graphical user interfaces is included in Chapter 10; this chapter describes the command interface.)

The VDE command interface can be invoked directly at the DCL prompt. VDE can also execute one or more semicolon-seperated commands directly at DCL, and will return back to DCL after command completion. Longer command lines can be broken across lines using the hyphen character; the command continuation character. test suite library, respectively.)

An example of a single VDE command follows:

```
$ VDE RESERVE/STREAM=FRED [ACC]FOO.MAR "Remark string"
```

The above example reserves module FOO.MAR in the ACC facility.

An example of a series of simple VDE commands follows:

[1] In certain existing installations of VDE, VDE users have access to three command interfaces: VDE, VSC, and VTSC. The command sets, the operations, and the personalities are all effectively identical; the salient differences among the three command interfaces include the library chosen for initial access, and the default setting of the /KEPT_PROCESS qualifier.
Unless otherwise indicated, the VDE, VSC, and VTSC command sets and capabilities are completely interchangeable. Any applicable differences will be explicitly called out.
See Appendix B.

```
$ VDE
VDE> RESERVE/STREAM=FRED [ACC]FOO.MAR "Remark string"
VDE> SHOW GENERATION [ACC]*
VDE> EXIT
$
```

The above command reserves module FOO.MAR in stream FRED, and displays some generation-related information. The caller then exits back to DCL.

When either command is prompting for commands, there are a variety of functions available. One can use the keyboard arrow keys for line editing and command line recall, and one can also define keyboard keys to invoke specific commands.

VDE[2] A kept subprocess starts the VDE utility in a subprocess the first time one uses the VDE command in a given process, and each subsequent VDE command (with /KEPT_PROCESS specified or defaulted) attaches to the existing subprocess to execute VDE commands. As a result, the work of activating the VDE utility and connecting to the source control database (which is a relatively slow operation) is done only for the first VDE command one enters in a given process. Subsequent VDE commands that access the database are then much faster. In addition, the kept subprocess retains state information—such as default settings, and even the initial process default directory—as long as that subprocess is active. The kept subprocess is implicitly terminated when the parent process is terminated. The kept subprocess is explicitly terminated via the VDE EXIT/KEPT_PROCESS command. Further information on this topic is included in Section 1.9.1.

1.2 VDE and the OpenVMS VAX Library

The remainder of this chapter discusses some high-level concepts needed when working with the OpenVMS VAX library, and some features of VDE that make this easier.

Section 1.3 discusses the basic unit of source control: the module. Modules are organized into facilities. Section 1.4 discusses how parallel development is organized, while Section 1.5 and Section 1.7 describe how changes to modules are propogated.

Facilities are grouped into Streams, and Section 1.4 describes the concept of a stream.

VDE can be extended for user requirements using a scripting mechanism. Some scripts, such as those needed to build a module, are required. Others, such as the script that triggers wheneever a module is replaced, are optional. A brief overview of scripts is located in Section 1.8.

VDE has a number of convenience features. For information on convenience features such as contexts, session initialization files, user-specified key definitions, and user-specified default text editors, see Section 1.9.

[2] VSC defaults to the use of a kept subprocess. This can be overridden with the /NOKEPT_PROCESS qualifier.

1.3 Modules and Facilities

The OpenVMS VAX library contains over 22,000 source modules. As this is far too many modules for anyone to reasonably keep track of the various associations, the library is broken down into over 200 *facilities*, and each module belongs to one of those facilities. The VDE system uses the following syntax to specify a module and the facility to which it belongs:

[fac-name]mod-name.type-name

For example, [FACIL]MOD.TYP refers to module MOD.TYP in facility FACIL. This syntax is accepted on VDE commands and is displayed in VDE output.

If you omit the bracketed facility name, as in MOD.TYP, VDE assumes that the module belongs to the default facility, provided a default facility is defined. You specify the default facility for your VDE session with the SET FACILITY command:

```
VDE> SET FACILITY SYS
```

This command makes facility SYS the default facility. Your default facility is normally undefined at the start of each VDE session. To see your default facility, use the SHOW DEFAULTS command.

1.4 Development Streams

Streams are central to the source control system. Each stream holds a particular generation of a module, and each stream has information associated that controls the propogation of changes among releases, the selection of the appropriate CMS generation numbers, the list of users that review changes, the list of users that are notified of changes, and various other attributes.

The OpenVMS Development Group usually has several versions of OpenVMS under development at any one time. These versions are called *development streams*, or simply *streams*. Examples of development streams include past versions of OpenVMS (such as V4.7 or V5.0), versions currently under development (often with code names such as DRAGON or EAGLE), off-to-the-side development streams (often called ‘‘variant streams’’ or ‘‘variant threads’’), and base levels.

As a VDE user, you must always reserve modules from a specified stream, and you must then later replace the updated modules into the same stream. As a result, many VDE commands require you to specify the development stream to which the commands should apply.

Many such commands allow you to specify the development stream with the /STREAM qualifier. If you omit this qualifier or the command does not accept it, the command acts on the default stream, provided a default stream is defined. One specifies the default development stream for the VDE session with the SET STREAM command:

```
VDE> SET STREAM EPSILON
```

This command makes stream EPSILON the default stream. The default stream is normally undefined at the start of each VDE session. To display the current default stream, use the SHOW DEFAULTS command.

1.5 Generations

VDE and VDE can store multiple instances of each source module in a library. Each instance is called a *generation* of the module.

Module generations are analogous to a railroad network—some tracks can branch off and run parallel to the mainline track, some sidings can diverge off the mainline, and some sidings can reconnect back into the mainline.

Each generation can represent a current or past instance of the module as it was at some point during its historical development. Different generations can also represent variants of the module that belong to different development streams being developed concurrently. Each generation of a module is created by reserving the previous generation, editing that generation to produce the new generation, and replacing the new generation into the library. Each generation (except the first) thus has a predecessor. A given generation may have zero or more successors. If there are multiple successors, these successor generations represent variants of the module for different development streams.

For each module that is represented in a given stream, there is exactly one generation that stores the current contents of that module for that stream. This generation is called the *latest generation* of the module for the stream. The source control database records what generation of each module belongs to each stream. A module need not have a latest generation for every stream, however. For example, a recently added module will not be used in older versions of OpenVMS and will therefore not have latest generations for those streams.

1.6 Architectures

VDE provides a mechanism by which source code for multiple architectures can be maintained. As it is possible that the same module name is used in each architecture, VDE allows the facilities and the module names to overlap. By default, VDE assumes all source code is architecture-independent.

VDE allows a suitably privileged user to create one or more architecture-dependent architecture roots, or *arch-roots*. Each arch-root stores zero or more architecture-dependent modules in an architecture-dependent facility—in effect, the structure of each arch-root parallels the structure of the main source pool.

For further information on (arch-roots) and on architecture support, see Chapter 9.

1.7 Propogation and Folding

VDE supports *propogation* and *folding*. Propogation is an automatic process that causes the replacement of a source module in the specified replacement stream, and into all successor streams; into all streams that have descended from the specified stream.

Folding is a manual form of propogation and is used when the VDE automatic propogation is not feasible. A fold record is created as a reminder that a module must be manually propogated into parallel and successor streams.

Folding records can be used to coordinate files across multiple arch-roots. Automatic propogation of source code changes across multiple arch-roots is not supported as it is expected the architecture-specific modules will differ rather substantially.

The automatic propogation occurs in successor streams only if the module has not been changed in the successor stream—the change is propogated only if the module is of the same generation. If the module has changed—if the source code has diverged—in any successor stream, a fold record is created for each successor stream. Propogation is based on module generation numbers.

1.8 Customization Using Scripts

VDE allows extensive user tailoring using *scripts*. A script allows you to easily perform a user-specified action as a part of a script-specific VDE operation. For instance, VDE invokes the replace script when a module is replaced in the VDE database.

Various types of scripts are optional and allow for site-specific processing. Specification of certain scripts is mandatory, as these scripts are used within VDE. For instance, the VDE VDE$BUILD utility requires specific information when compiling and linking modules and it retrieves this information from the (mandatory) user-specified compile and link scripts.

For further information on scripts and script creation, see Section 5.8, Table 5–5, and the *Script* topics in the *VDE Reference Manual*.

1.9 Convenience Features

VDE supports a number of convenience features.

1.9.1 Kept Subprocess

VDE can utilitize a kept subprocess. The kept subprocess is created when the command interface is first activated, and it greatly improves overall performance. With the kept subprocess, all of the processing required to initially connect to the database is performed only once, not each time the utility is activated.

Note: All security and quota related values associated with the kept subprocess are inherited from the parent process when the kept subprocess is spawned. This may seem like a minor detail, but it will cause all changes in the parent process's privilege mask and rightslist identifiers to be *ignored* by the kept subprocess.

To force an update of the security and quota attributes in the kept subprocess, issue the following commands:

```
VDE> EXIT/KEPT_PROCESS
$  VDE/KEPT_PROCESS
VDE>
```

The above commands cause the kept subprocess to exit and, as any subsequent invocation of VDE (with /KEPT_PROCESS) determines that there is no kept subprocess present, it will then spawn one.

1.9.2 Initialization Files

One can use an initialization file to set up VDE command abbreviations (with DEFINE/COMMAND), keypad definitions (with DEFINE/KEY), and session environment defaults (with SET FACILITY and SET STREAM). An initialization file can contain zero or or more VDE commands.

VDE supports a user-specified initialization file. Each utility processes an initialization file in a similar fashion, though each uses a unique name for the file. The initialization file is automatically invoked at the start of each VDE session.

To establish an initialization file for the VDE utility, define the logical name VDE$INIT[3] before starting the VDE session. This logical name must reference the file name of the VDE initialization file. If the file name and/or the file extension is omitted from the logical name translation, the VDE utility uses the default value VDE$INIT.VDE for the initialization file.

An initialization file is usually resident in the user's SYS$LOGIN: directory.

If desired, the exclamation point (!) can be used as a comment character in an VDE or an VDE initialization file. All text located after a exclamation point is ignored up to and including the end of the line, if the exclamation point is not located within a quoted string. The space-hyphen (–) can be used as a line continuation. The space-hyphen character must be the last character of a line; it causes the subsequent command line to be considered part of the current line.

[3] VSC and VTSC use the logical names VDE$INIT and VTSC$INIT, respectively, and the default specifications of VSC$INIT.VDE and VTSC$INIT.VDE, respectively.

1.9.3 Selecting a Text Editor

VDE lets one establish a text editor for the VDE session. VDE invokes the specified editor when certain commands are entered. For example, VDE lets one use an editor to review code changes on behalf of another user. The editor is invoked in a spawned subprocess, and can be invoked in read-write or read-only mode, depending on the context. To specify the editor for the current VDE session, use the SET EDITOR command. For example:

```
VDE> SET EDITOR "LSEDIT"
```

This command tells VDE to spawn the LSEDIT command, which invokes the VAX Language-Sensitive Editor, whenever a VDE command requires an editor to be invoked. If you prefer some other editor, you should specify the DCL command for that editor instead. You can also include any qualifiers you want in the quoted string. VDE simply starts with the string you specify, appends a space and the name of the file to edit, and spawns the resulting DCL command string. If the editor is invoked in read-only mode, VDE appends the /READ_ONLY qualifier (which most OpenVMS editors accept), a space, and a file name to the string you specify, and then spawns that. (If your editor does not accept the /READ_ ONLY qualifier, see the HELP text to see how you can specify a separate read-only editor string.)

The /DEFAULT qualifier on the SET EDITOR command allows one to establish the default editor strings for future VDE sessions. These editor strings are stored in the source control database. The SHOW EDITOR command displays the current editor strings.

VDEalso supports a number of standard utility commands. These commands include @*file-spec* (which executes a file containing VDE commands), SPAWN, ATTACH, EXIT, HELP, DEFINE/KEY and SET KEY, DEFINE/COMMAND, SET LOG and CANCEL LOG, SET VERIFY and CANCEL VERIFY, SET TERMINAL, and all associated SHOW commands. See the HELP text for descriptions of these commands.

1.9.4 Grouping Reservations into Sessions

VDE and VDE module reservations can be grouped together using *sessions*. One can create a session that contains a group of modules involved in a particular modification or fix—all modules contained in a session can be referenced in a single command.

```
VDE> REPLACE/SESSION=WIN32_VMS
```

The above command replaces all modules included in the WIN32_VMS session.

Information on how to *reserve* and *replace* modules is included in Chapter 2.

1.9.5 Defining Defaults Using Contexts

A *context* allows you to easily switch an entire group of related VDE default settings among a set of possible values. You can keep multiple sets of VDE defaults around.

Each context is comprised of a set of defaults: a default facility, a default stream, and a default architecture. (For information on architectures, see Chapter 9.) You can specify one context for use when you are making bugfixes to a particular stream and facility, and another context for when you are adding new features to a different facility in a newer stream. And using key definitions (DEFINE/KEY), you can switch among your various contexts using a single keystroke.

Contexts can be private and user-defined, or they can be publically established and available for whole groups of users. Project leaders can establish a set of contexts for use by a particular group of users.

1.9.6 Default Libraries, Mnemonics

VDE utilitizes a set of logical names to determine the names, contents and locations of commonly-accessed libraries. To see which libraries have been established by default (if any), issue the command:

```
VDE> SET LIBRARY
```

In addition to telling you that the command needs another parameter, this command will display any default libraries.

For information on the VDE$LIBRARY_DEFAULTS logical names, see *VDE Reference Manual*, or see Section 10.1.3.

2 Reserving and Replacing Modules

This chapter describes how to use the VSC utility to identify specific modules in a VDE library. It also shows you how to create, reserve and replace modules, how to fetch module generations without reserving them, how to create new modules, how to cancel reservations, and how to display reservation information from the source control database.

This section also describes some convenience features present in VSC and VDE. Examples of these features include the ability to reserve groups of modules, called *sessions*, and treat these modules as a unit. Also covered is the ability to queue a reservation request when a module is reserved by another user, and how to receive notification when the queued reservation request is eventually granted.

2.1 Identifying Module Generations

Each generation of a given module can be referenced using either of two names. One generation name is the *generation number*. This number is assigned and maintained by VSC. The other generation name is the *generation expression*. The generation expression is assigned by CMS. One can use either generation name to specify a specific generation in VSC commands.

VSC assigns generation numbers in sequential order to the generations for each development stream. Generation 1 is thus the first generation for the stream, generation 2 is its successor, and so on. Generation 2 of stream A can be the same as generation 2 of stream B (if the streams have not yet diverged) or they may be two different generations (assuming the streams have diverged).

When one refers to a generation by its generation number on a VSC command, one specifies a semicolon and the generation number after the module name. The generation number is relative to some stream that one specifies explicitly (usually with the /STREAM qualifier) or by default. For example, MOD.TYP;5 refers to generation 5 for the specified stream. One can also specify a zero or negative generation number; such a generation number is relative to the latest generation for the stream. MOD.TYP;0 thus refers to the latest generation and MOD.TYP;–1 refers to the next most recent generation, and MOD.TYP;–0 refers to the oldest.

CMS generation expressions are assigned by different means. The first generation for a module has generation expression 1. It can have multiple successors (for different streams) with generation expressions 2, 1A1, 1B1, 1C1, and so on. Generation 1B1 can in turn have successors 1B2, 1B1A1, 1B1B1, and so on. Each generation thus has a unique generation expression that encodes its genealogy. This expression is independent of stream. To specify a generation by its generation expression, one must use the /GENERATION qualifier for those commands that accept it.

For example, MOD.TYP/GENERATION=1B1 refers to the generation of module MOD.TYP that has generation expression 1B1.

If one omits both the generation number and the /GENERATION qualifier, most VSC commands reference the latest generation of the module for the specified stream. This causes the command to operate on the most recent generation by default—this is normally the desired effect.

Because generation numbers are always relative to some stream, they are used to show lines of descent for a given stream and to reference generations relative to a specific stream. CMS generation expressions are independent of stream and thus give absolute names to the generations of a module. These two kinds of generation names are primarily used with the SHOW GENERATION and FETCH commands. For example, to fetch the next most recent generation of module FOO.BAR for stream FRED, one would request generation number –1, as shown in this example:

```
VSC> FETCH /STREAM=FRED FOO.BAR;-1
```

However, one can also request the generation with generation expression 10A3 (which is an absolute name; entirely independent of stream), as in this example:

```
VSC> FETCH FOO.BAR /GENERATION=10A3
```

For most commands, including FETCH, one should not specify a generation. The default references the latest generation for a specified (or defaulted) stream, and this is nearly always the generation one typically wishes to reference. When reserving a module, one never specifies a generation to reserve; VSC always retrieves the latest generation for the specified stream.

Audit Trail Identifiers

When modifying a source module, one typically adds code comments that explain the nature of the changes. These comments should be labelled by an *audit trail identifier*. The audit trail identifier is simply the expected CMS generation expression of the new generation one creates when replacing the changed module into the VDE library. VDE displays the expected CMS generation expression when the module is initially reserved. In rare circumstances, the actual generation expression used when the module is replaced may differ from the expected generation expression displayed when the module is reserved. If this happens, VSC displays an informational message. One must then update the source module to correct the audit trail identifier in the code comments.

2.2 Reserving Modules

To reserve a module from a VDE library, use the RESERVE command. This command reserves a specified module for a given development stream and fetches that module into your default directory. This example illustrates the syntax of the RESERVE command:

```
VSC> RESERVE/STREAM=FOO [FACIL]MOD.TYP "Remark string"
```

This command reserves module MOD.TYP in facility FACIL from a VDE library. The latest generation of the module for stream FOO is copied into your default directory.

With the RESERVE command, you can specify a list of module names separated by commas. This syntax—or the session support—allows you to manipulate a number of modules with only a few commands. For example:

```
VSC> SET FACILITY SYS
VSC> RESERVE/STREAM=FOO A.MAR,B.MAR,C.MAR "Reduce memory-scrambling"
```

This example reserves modules A, B, and C in facility SYS from stream FOO. Note that you do not have to include the facility names with the RESERVE command if the modules belong to the default facility established with a previous SET FACILITY command.

You can use wildcard characters in the module names. The percent sign (%) in a name matches any single character in the position it occupies and the asterisk (*) matches zero or more characters in the position it occupies. All modules that match the wildcard pattern are reserved.

VDE also allows you to associate a number of modules from various RESERVE and CREATE MODULE commands together, by using the /SESSION qualifier. The /SESSION qualifier allows you to pick a name for a group of modules, and to avoid having to specify each module on the REPLACE command. The use of /SESSION also provides the reviewers and the project leader with a clear indication that a group of replacement modules are related.

The reservation remark string can be specified as a parameter at the end of the command, as shown in the previous example. The remark parameter has the same effect as the /REMARK qualifier described in Table 2–1 and applies to all reserved modules unless overridden by /REMARK qualifiers on the individual module-name parameters.

The RESERVE command displays the expected CMS generation expression for the new generation that will be created when the module is later replaced. You should use this generation expression as the ‘‘audit trail identifier’’ for the comments in your code that explain your new changes. This example illustrates the RESERVE command’s output:

```
VSC>RESERVE/STREAM=BIRCH [FACIL]MOD.TYP "Add gasket modulator"
%VDE-I-FETCHED, generation [FACIL]MOD.TYP;3(3) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]MOD.TYP;
%VDE-I-RESERVED, generation [FACIL]MOD.TYP;3(3) reserved from stream BIRCH
%VDE-I-EXPGENEXPR, expected generation expression at replacement is 3A1
%VDE-I-COMMIT, database transaction has successfully committed
VSC>
```

In this case, you should use 3A1 as your audit trail identifier. This expected CMS generation expression might differ from the one actually assigned when you later replace the module, but only in certain unusual circumstances. VSC prints a message when the replacement is performed if the expected and actual generation expressions differ. If so, you need to rereserve the module to correct the audit trail identifier.

The source control system keeps track of successor relationships between streams. If source changes replaced into a stream A should normally also be propagated to another stream B, then stream B is a *successor* of stream A. Successor relationships between streams are defined by the maintainers of the VDE libraries, and reside in the source control database. These relationships ensure that bug fixes and other changes made to one version of OpenVMS are propagated to all later versions of OpenVMS.

Successor relationships determine what streams are affected by a module reservation. Normally, a reservation *covers* the development stream specified with the /STREAM qualifier and all successors of that stream, including the successors of the successors, and so on recursively. If you omit the /STREAM qualifier, the reservation covers your default stream and its successors. This means that you expect your changes to be propagated to those streams when you later replace the module. However, the reservation does not cover any stream in which the module has already diverged from the original stream because VSC cannot automatically propagate source changes to such a stream.

The RESERVE command determines what streams your reservation will cover and checks if any existing reservation for the module already covers one of those streams. If so, the RESERVE command rejects your reservation attempt unless you explicitly request a concurrent reservation. You can limit change propagation, and therefore restrict the streams covered by your reservation, by using the /PROPAGATE and /NOPROPAGATE qualifiers. By covering fewer streams (when appropriate), you make reservation conflicts less likely.

The RESERVE command accepts a number of qualifiers. The help text and the reference manual have the details on all of the qualifiers accepted by the command. Table 2–1 lists some of the more commonly-used qualifiers.

Table 2–1 Qualifiers to the RESERVE command

/CONFIRM
/NOCONFIRM

/CONFIRM asks you to confirm that you want each module reserved. This qualifier is particularly useful if you use wildcard characters in the module name. /NOCONFIRM reserves each module without asking for confirmation. /NOCONFIRM is the default.

/IDENTIFICATION=res-ident

Specifies that the reservation be identified by the name given by the **res-ident** parameter. This name can be an arbitrary string that is meaningful to you. (The name must consist of letters, numbers, dollar signs, and underscores and can be up to 39 characters long.) If this qualifier is omitted, the source control system automatically assigns a numeric reservation identifier. Reservation identifiers distinguish between multiple concurrent reservations for the same module by the same user.

/LOG
/NOLOG

Table 2–1 (Cont.) Qualifiers to the RESERVE command

Specify whether informational log messages are printed after each module is reserved. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.

/NOCONCURRENT

Specifies that concurrent reservations not be allowed for any of the modules reserved by this command. If this qualifier is omitted, other users are allowed to concurrently reserve the modules.

/OUTPUT=filespec
/NOOUTPUT

/OUTPUT specifies that the reserved module be written to the file or directory given by the **filespec** parameter. /NOOUTPUT specifies that the reserved module not be written to an output file at all. If neither qualifier is specified, VSC creates the output file in your current default directory and gives it the same name and type as the module.

/OVERRIDE=CONCURRENT

Specifies that the RESERVE command should reserve each specified module even if the module is already reserved by another user. This qualifier is required to create concurrent reservations. However, VSC creates a concurrent reservation for a module only if all previous, conflicting reservations of that module allow concurrent reservations. If this qualifier is omitted, the RESERVE command reserves the module only if it is not already reserved. When this qualifier creates a concurrent reservation, the VSC utility automatically sends a mail message to each developer who already holds a reservation for the module in order to notify him or her of the new concurrent reservation.

/PROPAGATE=stream-name
/NOPROPAGATE

Determine what streams are covered by the current reservation and how module changes are propagated to successor streams when the module is later replaced into the library. /PROPAGATE specifies that the reservation should be propagated to all successors of the initial development stream up to and including the stream specified by the **stream-name** parameter. /NOPROPAGATE specifies that the reservation not be propagated past the initial stream. If neither /PROPAGATE nor /NOPROPAGATE is specified, the reservation is propagated to all successors (recursively) of the initial stream. These qualifiers are used when a change is appropriate for only a subset of the streams in the successor chain. For example, a bug fix might apply only to the initial stream because it is replaced by a new algorithm in later streams.

For OpenVMS VAX, most reservations should use the default propagation behavior and you should seldom need these qualifiers.

/QUEUE

Specifies that a reservation request should be queued if you cannot reserve the module because it is already reserved by another user. If the requested module can be reserved now, the module is reserved and this qualifier has no effect. If the module is not available, VSC queues a reservation request for it. When the module later becomes available, you are notified by a mail message and you can then reserve the module.

Table 2–1 (Cont.) Qualifiers to the RESERVE command

When you queue a reservation request, the source control system records the queued reservation in its database. The SHOW RESERVATION command displays queued reservations. When the developer who owns the current reservation later replaces the module into the library or cancels the reservation, VSC sends a mail message to each user who has queued a reservation request for the module. The mail message says that the module is available and lists all users who have requested it in the order that they requested it. Those users must then decide among themselves who should reserve the module first. To actually reserve the module, you must enter another RESERVE command.

/REMARK="string"

Specifies a one-line remark string to be associated with the reservation. The string can be up to 132 characters long.

/SESSION=session-name

/SESSION allows you to associated various reserved and created modules together into a session; sessions allow you to later replace a whole series of reserved and created modules via a single REPLACE command. When used with the queued-replacement environment, the use of sessions also provides a clear indication to the reviewers and to the project leader that the modules being replaced are associated. The particular session name used is typically a mnemonic name chosen by the user, and usually reflects the particular set of changes being made.

/STREAM=stream-name

Specifies the development stream from which the specified modules are reserved. The **stream-name** parameter gives the name of the stream. If this qualifier is omitted, the modules are reserved from your default development stream, provided one is defined.

/USERNAME=username

Specifies that the module should be reserved on behalf of the user whose OpenVMS user name is given by the **username** parameter. To use this qualifier, you must have a VSC privilege (USERNAME); the qualifier is meant for release project leaders and others who manage the VDE libraries, and is intended to be used only in special or unusual situations.

Examples

This is an example of the output of the RESERVE command:

```
VSC> RESERVE/STREAM=BIRCH [FACIL]MOD.TYP "Fix QAR 55"
%VDE-I-FETCHED, generation [FACIL]MOD.TYP;3(3) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]MOD.TYP;
%VDE-I-RESERVED, generation [FACIL]MOD.TYP;3(3) reserved from stream BIRCH
%VDE-I-EXPGENEXPR, expected generation expression at replacement is 3A1
%VDE-I-COMMIT, database transaction has successfully committed
VSC>
```

In this example, module MOD.TYP in facility FACIL is reserved from the OpenVMS VAX library. The module is reserved for stream BIRCH and all successor streams that have the same generation of MOD.TYP as BIRCH. One of the log messages indicates that generation 3 in that stream was fetched out. The new generation created when you later replace the module is expected to have the generation expression 3A1; you should use

this string as the audit trail identifier in the code comments explaining your changes.

The following example shows how VSC displays multiple log messages when you reserve multiple modules with the same RESERVE command:

```
VSC> SET FACILITY SYS
VSC> RESERVE/STREAM=PHOENIX A.MAR,B.MAR,C.MAR "Add scramble feature"
%VDE-I-FETCHED, generation [FACIL]A.MAR;5(2A3) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]A.MAR;
%VDE-I-FETCHED, generation [FACIL]B.MAR;1(1) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]B.MAR;
%VDE-I-FETCHED, generation [FACIL]C.MAR;3(3) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]C.MAR;
%VDE-I-RESERVED, generation [FACIL]A.MAR;5(2A3) reserved from stream PHOENIX
%VDE-I-EXPGENEXPR, expected generation expression at replacement is 2A4
%VDE-I-RESERVED, generation [FACIL]B.MAR;1(1) reserved from stream PHOENIX
%VDE-I-EXPGENEXPR, expected generation expression at replacement is 2
%VDE-I-RESERVED, generation [FACIL]C.MAR;3(3) reserved from stream PHOENIX
%VDE-I-EXPGENEXPR, expected generation expression at replacement is 4
%VDE-I-COMMIT, database transaction has successfully committed
VSC>
```

2.2.1 Concurrent Reservations

A module reservation is said to ‘‘cover’’ all development streams to which you expect the module to be propagated when you later replace it. This means that, if the stream that you reserve a module from has a successor stream and if the module has not yet diverged between the two streams, the reservation covers both streams. If the successor stream also has a successor, that stream is covered too (if the module has not diverged), and so on. If two reservations of the same module cover at least one common development stream, then the two reservations are said to be *concurrent reservations* of the module.

The source control system does not by default create a concurrent reservation for you when you reserve a module. If someone else’s existing reservation conflicts with the reservation you are trying to create, it is best for you to wait for the other developer to replace (or unreserve) the module before you reserve it and start to modify it. If both of you modify a module at the same time for the same stream, one of you will have to manually merge the parallel sets of changes to make the module right. This extra step in effect requires someone to enter his or her changes twice, which is more work and increases the risk of errors. Avoiding concurrent reservations is therefore your best policy.

However, there are exceptions. Sometimes you might need to change a module to fix a bug in the current build, for example, even though someone else (or even you yourself) already has reserved the module for more extensive changes. In such cases, you can use the /OVERRIDE=CONCURRENT qualifier on the RESERVE command to concurrently reserve the module. When you have completed your change, you queue the module for replacement with the REPLACE command. When the release project leader performs your replacement, the other developer’s reservation is cancelled and he or she is notified of this fact by a mail message. That other developer must then rereserve the module, merge his or her changes with your changes, and then continue his or her work. When you create a concurrent reservation for a module and then

replace the module first, you thus create work for another developer, so you need a good reason for doing so.

The following example illustrates what happens when you try to reserve a module that is already reserved:

```
VSC> RESERVE [RMS]MODULE_E.B32 "Emergency bug fix"
Module [BUILD_TEST]E.B32 is currently reserved:
   Reservation 1 by user HOFFMAN on 18-JAN-1990 15:45:51.77
      Remark: FIRST RESERVATION
%VDE-E-MODALRRES, module [RMS]MODULE_E.B32 already reserved by another user
 attempt to reserve module has failed
%VDE-I-ROLLBACK, database transaction rolled back
VSC>
```

In this example, the RESERVE command fails because the module is already reserved by another user. The command's output identifies the conflicting reservation so you can see who owns it (user HOFFMAN in this example). The reservation conflicts with the one you wanted to make because it covers some common stream for the same module.

If you decide that you must create a concurrent reservation for the module after all, you enter this command:

```
VSC> RESERVE/OVERRIDE=CONCURRENT [RMS]MODULE_E.B32 "Emergency bug fix"
Module [BUILD_TEST]E.B32 is currently reserved:
   Reservation 1 by user HOFFMAN on 18-JAN-1990 15:45:51.77
      Remark: FIRST RESERVATION
%VDE-I-FETCHED, generation [RMS]MODULE_E.B32;1(1) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]MODULE_E.B32;
%VDE-I-RESERVED, generation [RMS]MODULE_E.B32;1(1) reserved from stream BIRCH
%VDE-I-EXPGENEXPR, expected generation expression at replacement is 2
%VDE-I-COMMIT, database transaction has successfully committed
VSC>
```

This RESERVE command creates a concurrent reservation for the module because it contains the /OVERRIDE=CONCURRENT qualifier. It also sends mail to user HOFFMAN to notify HOFFMAN that you just concurrently reserved the module.

When you reserve a module, you can specify that you do not want other developers to create concurrent reservations for that module. To do so, use the /NOCONCURRENT qualifier on the RESERVE command:

```
VSC> RESERVE/NOCONCURRENT [RMS]MODULE_E.B32 "Remark string"
```

This qualifier causes the source control system to reject future attempts to create reservations that are concurrent with this one until you replace the module into the library or unreserve it. If you are about to make complicated changes to a module, you can thus prevent others from concurrently reserving that module. Normal usage, however, is to omit this qualifier and to allow concurrent reservations in case they are needed in an emergency. You can change a reservation's noconcurrent attribute with the MODIFY RESERVATION command.

2.2.2 Queued Reservations

VSC and VSC support queued reservations. When you queue a reservation request, the source control system records the queued reservation in its database, pending the availability of the specified module. When the developer who owns the current reservation later replaces the module into the library or cancels the reservation, VSC sends a mail message to each user who has queued a reservation request for the module.

The queued reservation request mail message indicates that the module is now available for reservation, and it lists all users who have requested it, in the order the requests were posted. The users listed in the message must decide among themselves who should reserve the module first.

To actually reserve the module, one must enter an explicit RESERVE command.

In addition to displaying other reservation information, the SHOW RESERVATION command displays all queued reservations.

2.3 Creating New Modules

To create a new source module, use the CREATE MODULE command:

```
VSC> CREATE MODULE [FACIL]MOD.TYP /STREAM=BIRCH /REMARK="Remark string"
```

This command records the new module in the source control database and creates a reservation for the module in the specified stream. It does not create the first generation of the module, however. You must use a subsequent REPLACE command to actually create the initial generation of the module.

You can create multiple modules with a single CREATE MODULE command by specifying a list of module names separated by commas.

VDE also allows you to associate a number of modules from various RESERVE and CREATE MODULE commands together, by using the /SESSION qualifier. The /SESSION qualifier allows you to pick a name for a group of modules, and to avoid having to specify each module on the REPLACE command. The use of /SESSION also provides the reviewers and the project leader with a clear indication that a group of replacement modules are related.

If you should mistakenly create a module under an incorrect module name, or should you create a module that becomes unnecessary, you can—*prior to module replacement*—issue a DELETE MODULE command. The DELETE MODULE command will cancel the reservation, and will correctly back out the CREATE MODULE command. For further details on this, see the *VDE Reference Manual*.

The CREATE MODULE command accepts a number of qualifiers. The help text and the reference manual have the details on all of the qualifiers accepted by the command. Table 2–2 lists some of the more commonly-used qualifiers.

Table 2–2 Qualifiers to the CREATE MODULE command

/LOG
/NOLOG

Specify whether informational log messages are printed after each module is created. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.

/MARKER
/NOMARKER

Specify that the module is a "marker file". A marker file is any binary, image, saveset, or library file that should not stored via the normal *delta* mechanism used within the CMS library, but should be stored as a seperate module in the marker file directory (see SHOW DIRECTORY/DELTA). This setting avoids having extensive delta-related processing and storage overhead between one generation of a binary module and the next.

/REMARK="string"

Specifies a one-line remark string to be associated with the module. This remark typically explains the purpose or function of the module. If you omit this qualifier, VSC prompts you for a remark.

/SESSION=session-name

/SESSION allows you to associated various reserved and created modules together into a session; sessions allow you to later replace a whole series of reserved and created modules via a single REPLACE command. When used with the queued-replacement environment, the use of sessions also provides a clear indication to the reviewers and to the project leader that the modules being replaced are associated. The particular session name used is typically a mnemonic name chosen by the user, and usually reflects the particular set of changes being made.

/STREAM=stream-name

Specifies the development stream from which the new module is reserved. The **stream-name** parameter gives the name of the stream. If this qualifier is omitted, the module is reserved from your default development stream, provided one is defined.

Examples

This is an example of the CREATE MODULE command:

```
VSC> SET STREAM PHOENIX
VSC> CREATE MODULE [FACNAM]NEWMOD.C /REMARK="Update widget table"
%VDE-I-RESERVED, generation [FACNAM]NEWMOD.C reserved from stream PHOENIX
%VDE-I-MODADDED, module [FACNAM]NEWMOD.C added to the database
%VDE-I-COMMIT, database transaction has successfully committed
VSC>
```

This example makes stream PHOENIX the default development stream and then creates the new module NEWMOD.C in facility FACNAM. The CREATE MODULE command adds the module to the source control database and reserves it from the default stream. A subsequent REPLACE command (not shown) is needed to create the first generation of the module in the library.

2.4 Fetching Modules

To fetch an existing module without reserving it, use the FETCH command. For example:

```
VSC> FETCH/STREAM=FOO [FACIL]MOD.TYP
```

This command copies the latest generation for stream FOO of module MOD.TYP in facility FACIL to your default directory.

The FETCH command accepts multiple module names separated by commas and accepts the asterisk (*) and percent sign (%) wildcard characters in the module-name parameters. Those modules whose names match the wildcard patterns are fetched.

You can specify the specific generation you want to fetch in two ways. One way is to specify a generation number preceded by a semicolon after the module name. For example, MOD.TYP;5 refers to absolute generation 5 for the specified development stream. MOD.TYP;0 refers to the latest generation of the module in the specified stream and MOD.TYP;–1 refers to generation –1 relative to the latest generation in the stream (in other words, to the next most recent generation). The second method to specify the generation you want is to use the /GENERATION=*gen-expr* qualifier to specify the CMS ‘‘generation expression’’ of the desired generation. If you specify neither a generation number nor the /GENERATION qualifier, VSC fetches the latest generation for the specified (or default) development stream.

In addition to the /GENERATION qualifier, the FETCH command accepts a subset of the RESERVE command qualifiers. The FETCH command also accepts an optional quoted remark parameter for compatibility with CMS, but since there is no reason to record fetch operations, the remark string is simply discarded.

The FETCH command accepts a number of qualifiers. The help text and the reference manual have the details on the qualifiers accepted by the command. Table 2–3 contains some of the more commonly-used qualifiers.

Table 2–3 Qualifiers to the FETCH command

/CONFIRM
/NOCONFIRM
/LOG
/NOLOG
/OUTPUT
/STREAM

Same as for the RESERVE command.

/GENERATION=gen-expr

Table 2–3 (Cont.) Qualifiers to the FETCH command

	Specifies that the module generation be fetched that has the CMS generation expression given by the **gen-expr** parameter. This expression is also the generation's audit trail identifier. If this qualifier is omitted, the command fetches the generation given by the generation number on the module-name parameter. If both this qualifier and the generation number are omitted, the command fetches the most recent generation of the module in the specified development stream.
/HISTORY[="string"] **/NOHISTORY**	Determine whether CMS history information is included in the fetched source module. /HISTORY causes CMS history information to be included in the output file and /NOHISTORY suppresses such information. If neither qualifier is specified, the history attribute for each module determines whether history information is included. For OpenVMS VAX, history information is turned off by default.
/NOTES[="string"] **/NONOTES** **/POSITION=col-num**	Determine whether CMS notes information is included in the fetched source module. /NOTES causes CMS notes information to be included in the output file and /NONOTES causes such information to be omitted. If CMS notes are included, the /POSITION qualifier specifies the column number of the notes text. If neither /NOTES nor /NONOTES is specified, the notes attribute for each module determines whether notes are included. For OpenVMS VAX, notes are turned off by default.

Examples

This is an example of the FETCH command:

```
VSC> FETCH A.REQ
%VDE-I-FETCHED, generation [FACNAM]A.REQ;3(2A1) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]A.REQ;
VSC>
```

This example fetches generation 3 (for the default stream) of module A.REQ in facility FACNAM. The generation's CMS generation expression (and audit trail identifier) is 2A1. VSC places a file containing that generation in your default directory.

The following example fetches two modules, C.B32 in facility SYS and D.B32 in facility BAD, and copies them to directory [MYDIR]:

```
VSC> FETCH/OUTPUT=[MYDIR] [SYS]C.B32/STREAM=V4.4,[BAD]D.B32/GENERATION=3A2
%VDE-I-FETCHED, generation [SYS]C.B32;4(4) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]C.B32;
%VDE-I-FETCHED, generation [BAD]D.B32;5(3A2) fetched from library
%VDE-I-OUTFILE, output file is DEV$:[MYDIR]D.B32;
VSC>
```

The command fetches the latest generation of C.B32 for stream V4.4, and it fetches the generation of D.B32 that has CMS generation expression 3A2.

2.5 Unreserving Modules

To cancel a module reservation without replacing the module, use the UNRESERVE command. The following example illustrates its use:

```
VSC> UNRESERVE/STREAM=FOO [FACIL]MOD.TYP "Remark string"
```

This command cancels the reservation of module MOD.TYP in facility FACIL. The /STREAM qualifier specifies the stream for which the reservation was made. If you omit this qualifier, your default stream is assumed. If you have more than one reservation of the module for that stream, you must specify its reservation identifier with the /IDENTIFICATION qualifier. The command can cancel both queued reservation requests and real reservations.

The UNRESERVE command accepts wildcard characters in the module-name parameter. Those modules whose names match the wildcard pattern are unreserved. The command also accepts multiple module-name parameters separated by commas.

The optional remark string parameter specifies a remark that explains why the module reservation was cancelled. You can also specify the remark string with the /REMARK qualifier.

The UNRESERVE command accepts a subset of the REPLACE command qualifiers (see Table 2–5). The following qualifiers have the same meanings as for the REPLACE command: /CONFIRM, /NOCONFIRM, /IDENTIFICATION, /LOG, /NOLOG, /STREAM, and /USERNAME.

2.6 Displaying Reservations

Once a reservation has been entered into the source control system's database, you can display it with the SHOW RESERVATION command. This is a typical example:

```
VSC> SHOW RESERVATION [FACIL]MOD.TYP
```

This command shows all reservations and queued reservation requests for module MOD.TYP in facility FACIL. If the module-name parameter is omitted, the command displays all reservations for modules in your default facility.

The SHOW RESERVATION command accepts a list of module names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the module names. All reservations for modules that match the wildcard patterns are displayed.

The SHOW RESERVATION command accepts a number of qualifiers. The help text and the reference manual have the details on all of the qualifiers accepted by the command. Table 2–4 describes a number of the more commonly-used qualifiers.

Table 2–4 Qualifiers to the SHOW RESERVATION command

/BEFORE=date-time

Table 2–4 (Cont.) Qualifiers to the SHOW RESERVATION command

Displays only those reservations created before the specified date and time.

/BRIEF
/FULL

Determine the type of information displayed for each reservation. /BRIEF displays the reservation identifier, the name of the reserved module, the reserving user, the date of the reservation, and the stream in which the module is reserved. This display normally fits on one line. /FULL displays all database information about the reservation; this display requires several lines per reservation. /BRIEF is the default.

/NOQUEUE

Specifies that queued reservation requests not be displayed. If this qualifier is omitted, the command displays both queued reservation requests and normal reservations.

/OUTPUT=filespec

Directs the printed output of this command to a specified file. The **filespec** parameter specifies the name of the file. VSC creates a new file with that name, directs the command's print output to that file, and prints nothing on your terminal. If this qualifier is omitted, all output appears on the terminal.

/SINCE=date-time

Displays only those reservations created on or after the specified date and time.

/STREAM[=stream-name]

Displays only those reservations made for the development stream given by the **stream-name** parameter. If the **stream-name** parameter is omitted, the command only displays those reservations made for your default stream. If the /STREAM qualifier is omitted entirely, the command displays reservations for all streams.

/TOTAL

Shows the total number of reservations that match the specified module names, usernames, and other constraints. Only the total count is shown; other information about the individual reservations is not shown.

/USERNAME[=username]

Specifies the OpenVMS user name for the user whose reservations are to be shown. If this qualifier is specified, VSC only displays that user's reservations. If the **username** parameter is omitted, VSC only displays your own reservations. If the whole qualifier is omitted, VSC displays reservations for all users.

Examples

This is an example of the brief output of the SHOW RESERVATION command:

```
VSC> SHOW RESERVATION
Reservation 1 of [FACIL]FOO.MAR             SMITH        18-NOV-1990 BIRCH
Reservation 1 of [FACIL]FUMBLE.MAR          JONES         5-SEP-1990 PHOENIX
Queued reservation 2 of [FACIL]FUMBLE.MAR   MURGATROID   21-NOV-1990 PHOENIX
Reservation 1 of [FACIL]SCRAMBLE.B32        SMITH        31-OCT-1990 MUSHROOM
VSC>
```

The following example shows the full output of the SHOW RESERVATION command for one reservation:

```
VSC> SHOW RESERVATION/FULL [FACIL]FOO.MAR
Reservation 1 of [FACIL]FOO.MAR
   Reservation created on 18-NOV-1990 10:39:54.31
   Reservation created by user SMITH (Jane Smith)
   Generation 15 (15) reserved from stream BIRCH
   Expected generation expression at replacement is 16 (audit trail identifier)
   Concurrent reservations are allowed
   Remark: Fix QAR 23 again

VSC>
```

2.7 Replacing Modules

After you have reserved a module and changed it, you need to build it, test it, and replace it onto the OpenVMS VAX library. This chapter describes how you replace modules back onto the OpenVMS VAX library after you have changed, built, and tested them as appropriate.

You use the REPLACE command to copy a new or changed module from your default directory (or another directory you specify) into the specified library. Before you can use the REPLACE command for a module, you must first reserve the module with the RESERVE command or create the module with the CREATE MODULE command. The REPLACE command replaces the module into the development stream you specify and automatically propagates the change to that stream's successors. The following example shows how to use the REPLACE command:

```
VSC> REPLACE/STREAM=FOO [FACIL]MOD.TYP "Remark string"
```

This example replaces module MOD.TYP in facility FACIL onto the OpenVMS VAX library. The module is replace into stream FOO, which means that the module must already be reserved for that stream and the input file becomes the latest generation of the module for that stream. If you omit the /STREAM qualifier, the module is replaced into your default stream.

With the REPLACE command, you can specify a list of module names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the module names. All modules that match the wildcard patterns are replaced.

VDE also allows you to associate a number of modules from various RESERVE and CREATE MODULE commands together, by using the /SESSION qualifier. The /SESSION qualifier allows you to pick a name for a group of modules, and you can avoid having to specify each individual module on the REPLACE command. The use of /SESSION also provides the reviewers and the project leader with a clear indication that a group of replacement modules are related.

Optionally, you can specify a remark string as a parameter at the end of the command, as shown in the previous example. The remark parameter has the same effect as the /REMARK qualifier described in Table 2–5 and applies to all modules being replaced unless overridden by /REMARK qualifiers on the individual module-name parameters. If you do not specify

a remark, VSC prompts you for it. If you then enter a null remark string, VSC uses the remark from your original RESERVE command.

The REPLACE command also asks you for the cause that is the reason or justification for the the replacement. The reason can be a fold record, a Source Control Tracking (SCT) entry, or other cause. If the reason is specified by an SCT entry, VSC allows several options: it can enter the SCT note for you, it can ask you for the number of a pre-existing SCT entry, and it allows you to edit the SCT note during the replacement. VSC also asks for any related QARs (Quality Assurance Reports) or CLDs (urgent customer problems). The following partial example shows how you specify the SCT entry and problem reports associated with a replacement:

```
VSC> REPLACE/STREAM=FOO [FACIL]MOD.TYP❶
Enter the replacement remark: example replacement command ❷
General type of replacement:
  FO : Fold; see original change for info,
  SC : SCT; See Source Code Tracking Note,
  OT : Other; replacement made for other reason(s)
Type of replacement? sct❸
Reason(s) for making this change (from the SCT note):
  BF : Bug Fix,
  NF : New Feature/Enhancement,
  PI : Performance Improvement,
  CS : Common Sources,
  CU : Cleanup/Retrofit/Regression,
  NH : New Hardware Support,
  FE : Functional Equivalence,
  OT : Other
What was the reason for the change: bf❹
How many defects -- errors from any and all sources: design, coding,
spelling, logic, etc. -- did this replacement repair?
Number of defects: 2❺
Enter name of SCT conference [SCT-FOO]:❻
Enter SCT project name or [RETURN] for list: VORLON❼
Enter file to load into SCT: (filename/EDIT/NOTE) sys$login:sct.txt❽
%VDE-I-NOTEID, note 21.0 entered in conference❾
Please specify related QARs and CLDs, if applicable:
Enter name of QAR database (or [RETURN] if none): QAR-DB-NAME❿
Enter QAR number: 13⓫
Enter name of QAR database (or [RETURN] to end list):⓬
Enter the CLD number (or [RETURN] if none): CLD-12345⓭
Enter the CLD number (or [RETURN] to end list): ⓮
```

❶ The replacement command specifies the target stream, FOO, and the facility and module to be replaced. Multiple modules can be specified explicitly by including their names as a comma-seperated list on the command, or implicitly via the /SESSION support.

❷ The replacement remark is entered. If VSC is later asked to enter an SCT note, the specified project name and the text of this remark will be used as the title of the SCT note.

❸ The particular type of replacement. SCT replacements are the most common. The particular type of replacement chosen will alter the questions asked. If you had entered FOLD or had specified /FOLD, VSC would have canceled the outstanding fold records for you. If you had specified OTHER or /OTHER, VSC would ask you for the reason why this change was not made via an an SCT or due to a FOLD.

❹ The particular reason for making the replacement. The reason is used for various purposes within VSC. Can be specified on the REPLACE command using /REASON.

❺ The user-specified number of defects corrected. This value is used for statistical quality control purposes. Can be specified on the REPLACE command using /DEFECT.

❻ The SCT notes conference for this stream. By default, the stream is SCT-*stream-name*. If you are not sure of the stream name or the name of the SCT conference, see your project leader. If /SCT or /NOTE is specified on the REPLACE command, this prompt is skipped and the default SCT notes conference is used.

❼ The project name this replacement is associated with. Press RETURN for a list of projects associated with the stream. If you are not sure of the correct project name, see your project leader. You can specify this information using the /PROJECT qualifier, and this prompt will be skipped.

❽ By the specification of a filename, VSC has been asked to enter the specified file directly into the SCT notes conference. The remark is used as the note title. The keywords EDIT and NOTE are also accepted at this point, and cause VSC to invoke the editor for you, or cause VSC to prompt you for the number of a pre-existing and previously-entered SCT note. Can be specified by using /SCT or /NOTE on the REPLACE command. (The default for this prompt can be user-customized using the VDE$REPLACE_DEFAULT logical name. See the reference manual for details.)

❾ This replacement has created topic 21.0 in the SCT-FOO SCT notes conference.

❿ The name of the QAR database for the QAR associated with this replacement is QAR-DB-NAME.

⓫ This replacement is related to QAR number 13.0 in the QAR-DB-NAME QAR database. Specification of /NOQAR on the REPLACE command avoids the QAR and CLD prompts for information.

⓬ There are no other associated QARs, signified by pressing RETURN.

⓭ This replacement is associated with the CLD number CLD-12345. VSC checks CLD syntax.

⓮ There are no other associated CLDs, signified by pressing RETURN.

Queued Replacements

When you use the REPLACE command to replace a module into a normal OpenVMS VAX development stream, VSC does not immediately replace the module into the library, and it does not immediately cancel your reservation on the module. Instead, it queues the module for replacement at a later time when the release project leader for the stream allows the actual replacement to take place. The release project leader normally will not allow the actual replacement to proceed until your replacement has been reviewed.

When the REPLACE command queues your replacement (also called creating a *queued replacement*), it performs the following operations:

- The source control system records the queued replacement in its database and assigns it a name. This name can later be used with other commands to reference the queued replacement. By default, VSC creates the replacement name by concatenating your user name, a hyphen, and a small integer, where the integer makes the replacement name unique. If your username is SMITH, for example, your first replacement is named SMITH–1. You can override this default name with the /REPLACEMENT qualifier.
- The source control system records what modules belong to this queued replacement. All modules you list on the REPLACE command become members of the queued replacement. These modules are still marked as reserved in the database since they have not actually been replaced yet.
- The source control system copies each replaced module to a special disk directory under its own control so that the module no longer needs to remain in your directory. This special disk directory is called the replacement's *staging area*. (An added benefit of staging areas is that they allow lost generations to be recovered after a VDE failure; see Section 7.4.10.)
- The source control system asks you whether you want to edit an information file for the replacement to pass instructions or other information to the reviewers and the release project leader. If you answer YES, it invokes your preferred editor to create the file.

VSC queues the module replacement request into the VDE database for review, and notifies the release project leader. Once the changes are reviewed, the release project leader then uses the PERFORM REPLACEMENT command to accept the replacement. Once the replacement has been accepted, the pending modification request will be dequeued and the modification will be committed into the VDE database. You will typically receive a mail message from VSC when your replacement has been performed. See Chapter 4 and Section 5.1 for information on the VSC commands used by reviewers and by project leaders, respectively.

Immediate Replacements

For some development streams, the VSC utility performs *immediate* replacements. An immediate replacement updates the library—you guessed it—immediately. When you enter the REPLACE command, the module is copied into the library. OpenVMS VAX does not typically permit immediate replacements into its major development streams, but might allow them for selected off-to-the-side development streams used by small numbers of developers. Whether the REPLACE command performs a queued or immediate replacement is thus a property of the development stream you are replacing modules into. If any development stream affected by the replacement (directly or through change propagation) requires queued replacements, the replacement is queued.

A disadvantage of immediate replacements is that they do not create staging areas, which can prevent recovery of lost generations after a VDE failure.

Qualifiers

The REPLACE command accepts a number of qualifiers. The help text and the reference manual have the details on the qualifiers accepted by the command. Table 2–5 lists the more commonly-used qualifiers.

Table 2–5 Qualifiers to the REPLACE command

/CONFIRM
/NOCONFIRM

/CONFIRM asks you to confirm that you want each module replaced. This qualifier is particularly useful if you use wildcard characters in the module name. /NOCONFIRM replaces each module without asking for confirmation. /NOCONFIRM is the default.

/DEFECT=defect-count

Specifies the number of defects the particular replacement remedied. Defects counted include spelling errors, run-time errors, design errors, etc. This qualifier is applicable only to changes with bug-related reason codes.

/FOLD[=fold-ident]

Specifies that the fold record identified by the **fold-ident** parameter be deleted from the database when the present replacement completes. If this qualifier is omitted, no fold record is cancelled. (See Section 2.8 for a discussion of fold records.)

/IDENTIFICATION=res-ident

Specifies the reservation to be replaced. This qualifier is required only when you have multiple concurrent reservations of the same module. The **res-ident** parameter is the reservation identifier of the reservation to be replaced. This is the reservation identifier that you specified or the source control system assigned when you reserved the module; it is usually a small integer value.

/INFORMATION[=filespec]
/NOINFORMATION

/INFORMATION specifies that additional information about the current queued replacement is found in the text file specified by the **filespec** parameter. This information might include instructions or advice that the release project leader should read before performing the queued replacement. It might also include information for your reviewers. The information file is copied to the replacement's staging area and is discarded once the replacement is performed.

If the **filespec** parameter is omitted, you are placed in an editing session to create the information file. If the /NOINFORMATION qualifier is specified, no information file is attached to the replacement. If neither qualifier is specified for a queued replacement, you are asked whether you want to edit an information file. These qualifiers are ignored for immediate replacements.

/INPUT=filespec

Table 2–5 (Cont.) Qualifiers to the REPLACE command

Specifies the source file to be used as input for the replacement. If you omit the directory specification from the **filespec** parameter, the file is taken from your default directory. If you omit the file name or file type, the module name and module type are used by default. If you omit the /INPUT qualifier entirely, the file is taken from your default directory and must have the same file name and type as the module.

/LOG
/NOLOG

Specify whether informational log messages are printed after each module is replaced or queued for replacement. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.

/NOTE=sct-note-number

Specifies the number of a previously-entered SCT note in the SCT notes conference. When specified, VSC uses the default name for the SCT notes conference. If omitted, VSC will prompt for the SCT-related information.

/OTHER="other reason"

Specifies that the replacement is not associated with an SCT entry or a FOLD record, and includes the reason why.

/PROJECT=project-name

/PROJECT specifies the project name with which the replacement is associated. The project is used for statisitical purposes, and the project name is used in conjunction with the remark text as the title used for any SCT note entered by VSC.

/PROPAGATE=stream-name
/NOPROPAGATE

/PROPAGATE specifies that the module changes being replaced should be propagated to all successors of the initial development stream up to and including the stream specified by the **stream-name** parameter. Changes cannot be propagated further than specified on the original RESERVE command, however. /NOPROPAGATE specifies that the changes not be propagated past the initial development stream. If neither qualifier is specified, VSC propagates the changes as far as requested on the original RESERVE command, which by default propagates changes to all successors (recursively) of the current stream. See Section 2.8 for more detail on change propagation.

For OpenVMS VAX, most replacements should use the default propagation behavior and you should seldom need these qualifiers.

/REASON=reason-code

Specifies the reason code for the replacement. Typically associated with an SCT entry. Bug-related reason codes require the specification of the number of defects corrected.

/REMARK="string"

Specifies a one-line remark string to be associated with the replacement. If this qualifier and the remark parameter are omitted, you are prompted for a replacement remark. If you enter a null replacement remark, VSC uses the remark string from each module's reservation. The project name and the remark text are used as the title for any SCT note entered by VSC.

Table 2–5 (Cont.) Qualifiers to the REPLACE command

/REPLACEMENT=repl-name

Specifies the name of the queued replacement if the replacement is queued. (This qualifier is ignored for immediate replacements.) If a queued replacement by that name already exists, VSC appends the specified modules to that queued replacement. Otherwise, VSC creates a new queued replacement by that name. If this qualifier is omitted, VSC creates a new queued replacement and assigns it a unique replacement name based on your OpenVMS user name.

/REVIEWER=(username [, username...])

Specifies the users who should review this replacement. Each **username** parameter specifies the OpenVMS user name of one reviewer for the replacement. Each such reviewer is automatically notified with a mail message. These reviewers are expected to review the changes that make up the current replacement as described in Chapter 4.

/SCT=sct-file-name

Specifies the SCT note to be entered into the SCT notes conference. When specified, VSC uses the default name for the SCT notes conference. If omitted, VSC will prompt for the SCT-related information.

/SESSION=session-name

/SESSION allows you to replace a number of reserved and created modules together with a single command, and allows you to avoid specifying each module on the command. When used with the queued-replacement environment, the use of sessions also provides a clear indication to the reviewers and to the project leader that the modules being replaced are associated. The particular session name used is typically a mnemonic name chosen by the user when the CREATE MODULE or RESERVE command is issued, and usually reflects the particular set of changes being made.

/STREAM=stream-name

Specifies the development stream in which the modules to be replaced are currently reserved and into which they should be replaced. The **stream-name** parameter gives the name of the stream. If this qualifier is omitted, VSC replaces the modules into your default development stream.

/USERNAME=username

Specifies that the modules should be replaced on behalf of the user whose OpenVMS user name is given by the **username** parameter. You must have a VSC privilege (USERNAME) to use this qualifier; it is meant for release project leaders and others who manage the OpenVMS libraries, and is intended only be used in unusual or special situations.

Examples

In the following example of a queued replacement, user HOFFMAN replaces module FOO.MAR into facility SYS in development stream PHOENIX:

```
VSC> REPLACE/STREAM=PHOENIX [SYS]FOO.MAR
Enter the replacement remark: example replacement command
General type of replacement:
  FO : Fold; see original change for info,
  SC : SCT; See Source Code Tracking Note,
  OT : Other; replacement made for other reason(s)
Type of replacement? sctReason(s) for making this change (from the SCT note):
  BF : Bug Fix,
  NF : New Feature/Enhancement,
  PI : Performance Improvement,
  CS : Common Sources,
  CU : Cleanup/Retrofit/Regression,
  NH : New Hardware Support,
  FE : Functional Equivalence,
  OT : Other
What was the reason for the change: bf
How many defects -- errors from any and all sources: design, coding,
spelling, logic, etc. -- did this replacement repair?
Number of defects: 1
Enter name of SCT conference [SCT-PHOENIX]:
Enter SCT project name or RETURN for list: BUCKAROO
Enter file to load into SCT: (filename/EDIT/NOTE) sys$login:sct.txt
%VDE-I-NOTEID, note 41.0 entered in conference
Please specify related QARs and CLDs, if applicable:
Enter name of QAR database (or RETURN if none):
Enter the CLD number (or RETURN to end list):
%VDE-I-FILCOPSTAG, file DEV$:[HOFFMAN]FOO.MAR;3 copied to staging area
%VDE-I-REPLQUE, module [SYS]FOO.MAR queued for replacement HOFFMAN-1
%VDE-I-COMMIT, database transaction has successfully committed
Do you want to edit an information file for replacement HOFFMAN-1 ? [No]: NO
VSC>
```

As the log messages show, VSC queues the replacement and assigns it the name HOFFMAN–1. VSC also asks whether the user wants to edit an information file for the replacement. An information file passes information to the reviewers and the release project leader. In this example, the answer is NO, but had the answer been YES, VSC would have invoked an editor to create the information file. Replacement HOFFMAN–1 is then available for review. It will be reviewed by user JONES and by any automatic reviewers for the module and stream. After it has been reviewed, it is up to the release project leader to actually perform the replacement.

In the following example of an immediate replacement, module MOD.TYP in facility FACIL is replaced into development stream TOM:

```
VSC> REPLACE/STREAM=TOM [FACIL]MOD.TYP "Remark string"/NOTE=321
Type of entry to cause replacement (FOLD, SCT, OTHER): SCT
Please specify related QARs and CLDs if applicable:
Enter name of QAR database (or RETURN if none): V54-FT
Enter QAR number: 35
Enter name of QAR database (or RETURN to end list):
Enter the CLD number (or RETURN if none): APO-12345
Enter the CLD number (or RETURN to end list):
%VDE-I-FILCOPLIB, file DEV$:[MYDIR]MOD.TYP;12 copied to library
%VDE-I-REPLACED, generation [FACIL]MOD.TYP;4(4) replaced into stream TOM
%VDE-I-REPLACED, generation [FACIL]MOD.TYP;4(4) replaced into stream JERRY
%VDE-I-COMMIT, database transaction has successfully committed
VSC>
```

Because stream JERRY is a successor of stream TOM in this example and because this module has not diverged between those two streams, the change (the new module generation) is automatically propagated to stream JERRY. The new instance of this module thus becomes the latest generation of the module for those two streams. The /NOTE qualifier

indicates that additional information on this change was included in a previously-created SCT note, 321.0; this note was previously loaded into the SCT notes conference SCT-TOM.

2.8 Change Propagation and Fold Records

When you replace a module, you make your new source file the most recent generation of the module in the specified (or default) development stream. That instance of the module is thus the generation you get by default in a subsequent FETCH or RESERVE command and is the generation that is built for that stream.

If the specified stream has successors, VSC propagates the new module generation to all successor streams if possible. For example, if stream V5.4 is a successor of stream V5.3, then any module that is replaced into stream V5.3 is also replaced in stream V5.4 automatically. If stream V5.4 has additional successors (such as stream V5.5), the new module generation is propagated to those streams, and so on recursively.

Change propagation works as follows. If the latest generations of the module for the original development stream and the successor stream are the same, then the new module instance being replaced becomes the latest generation for both streams. Both streams are thus equally up-to-date with respect to the changes made to that module. However, if the latest generations of the module differ for the two streams, which means that the module has already diverged between those two streams, the source control system does not propagate the new module generation to the successor stream. Instead, it creates a *fold record* in the database to remind the developer that the change made to the module for the first stream must be manually ‘‘folded’’ into the successor stream.

You display fold records with the SHOW FOLD command. This example shows the brief output of the command:

```
VSC> SHOW FOLD *.MAR
Fold 1 of [FACIL]FOO.MAR;29(27B2) into stream V5.4     23-MAR-1989 SMITH
Fold 2 of [FACIL]BAR.MAR;2(1A1) into stream V5.5       24-MAR-1989 JONES
VSC>
```

Each fold record is identified by a *fold identifier*. This identifier is a unique, small integer value that VSC assigns when creating the fold record.

The following example shows the full output of the SHOW FOLD command for one fold record:

```
VSC> SHOW FOLD/FULL FOO.MAR
Fold 1 of [FACIL]FOO.MAR;29(27B2)
   Module to be folded from stream V5.3 into stream V5.4
   Reservation performed by user SMITH (Jane Smith)
   Reservation performed on 23-MAR-1989 18:09:51.80
   Replacement performed on 25-MAR-1989 20:29:05.23
   Remark: Fix QAR 23 again

VSC>
```

The normal way to remove fold records is to specify the /FOLD qualifier with the REPLACE command that replaces the folded source file. This is an example:

```
VSC> REPLACE/STREAM=V5.4 [FACIL]FOO.MAR/FOLD=2
```

This example replaces module FOO.MAR into stream V5.4 to propagate a change that VSC could not automatically propagate. The parameter to the /FOLD qualifier (the value ''2'') is the fold identifier of a fold record to delete as part of the replacement. This parameter can be omitted if the module only has one outstanding fold record. For queued replacements, VSC deletes the fold record when the release project leader performs your replacement.

You can also delete a fold record that is no longer needed (possibly because the change is no longer required) with an explicit CANCEL FOLD command:

```
VSC> CANCEL FOLD [FACIL]FOO.MAR/IDENTIFICATION=2/STREAM=BIRCH
```

This command deletes fold record 2 (as given by the /IDENTIFICATION qualifier) for module FOO.MAR in facility FACIL. This fold record is a reminder to fold a change into stream BIRCH. If there is only one fold record for the given module and stream, you can omit the /IDENTIFICATION qualifier.

2.9 Displaying Queued Replacements

Once the REPLACE command has entered a queued replacement into the source control system's database, you can display the replacement with the SHOW REPLACEMENT command. This example illustrates the command's syntax:

```
VSC> SHOW REPLACEMENT SMITH-1
```

This command displays all queued replacements specified by the replacement name parameter (SMITH–1 in the example). If the replacement name is omitted, the command displays all queued replacements.

With the SHOW REPLACEMENT command, you can specify a list of replacement names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters. All queued replacements that match the wildcard patterns are displayed.

The SHOW REPLACEMENT command accepts a number of qualifiers. The help text and the reference manual have the details on the qualifiers accepted by the command. Table 2–6 lists some of the more commonly-used qualifiers.

Table 2–6 Qualifiers to the SHOW REPLACEMENT command

/ACCEPTED

Specifies that only those replacements that have been accepted by all their reviewers be displayed.

Table 2–6 (Cont.) Qualifiers to the SHOW REPLACEMENT command

/BEFORE=date-time

Displays only those replacements queued before the specified date and time.

/BRIEF
/FULL

Determine the type of information displayed for each replacement. /BRIEF displays the replacement name, the date of the replacement, the replacing user, and the replacement status. This display normally fits on one line. /FULL displays all information about the replacement stored in the database; this display requires multiple lines per replacement.

/IN_REVIEW

Specifies that only those replacements that are in review be displayed. A replacement is in review if it has not been rejected by any reviewer, has been accepted by at least one reviewer, and has not yet been accepted by all reviewers.

/NOT_REVIEWED

Specifies that only those replacements that have not yet been accepted or rejected by any reviewers be displayed.

/OUTPUT=filespec

Directs the printed output of this command to a specified file. The **filespec** parameter specifies the name of the file. VSC creates a new file with that name, directs the command's print output to that file, and prints nothing on your terminal. If this qualifier is omitted, all output appears on the terminal.

/REJECTED

Specifies that only those replacements that have been rejected by at least one reviewer be displayed.

/REVIEWER[=username]

Specifies that only those replacements for which the specified user is a reviewer be displayed. The **username** parameter specifies the OpenVMS user name of the reviewer whose replacements are to be shown. If the **username** parameter is omitted, those replacements for which you are a reviewer are shown. If the qualifier is omitted entirely, replacements for all reviewers are shown.

/SINCE=date-time

Displays only those replacements queued on or after the specified date and time.

/STREAM[=stream-name]

Displays replacements for the development stream given by the **stream-name** parameter. If this parameter is omitted, the command displays replacements for your default development stream. If the /STREAM qualifier is omitted entirely, the command displays replacements for all development streams.

/TOTAL

Shows the total number of replacements that match the specified replacement names and other selection criteria. Only the total count is shown; other information about the individual replacements is not shown.

/USERNAME[=username]

Table 2–6 (Cont.) Qualifiers to the SHOW REPLACEMENT command

	Specifies the OpenVMS user name of the user whose replacements are to be shown. If this qualifier is specified, only that user's replacements are shown. If the **username** parameter is omitted, only your own replacements are shown. If the /USERNAME qualifier is omitted entirely, replacements for all users are shown.

Examples

This is an example of the brief output of the SHOW REPLACEMENT command:

```
VSC> SHOW REPLACEMENT SMITH-5
Queued replacement SMITH-5           23-MAR-1989 SMITH        Rejected
Queued replacement JONES-3           24-MAR-1989 JONES        Not reviewed
VSC>
```

The following example shows the full output of the SHOW REPLACEMENT command for one queued replacement. Note that this output lists all modules that belong to the replacement and all reviewers for the replacement.

```
VSC> SHOW REPLACEMENT/FULL SMITH-5
Queued replacement SMITH-5
   Replacement queued by user SMITH (Jane Smith)
   Replacement queued on  4-OCT-1990 16:55:02.06
   Replacement staging directory is VSC$STAGEDISK:[VMS.STAGE.VDE$STG_1.VDE$REP_115]
   Replacement review status: Rejected
   Replacement is for stream PHOENIX
   Unique replacement number is 115
   Reservations in replacement:
      Reservation 1 of [FACIL]FOO.MAR for stream PHOENIX
      Reservation 1 of [RMS]FUMBLE.B32 for stream PHOENIX
   Replacement contains 2 reservations
   Reviewers for replacement:
      Accepted by user HOFFMAN (Bill Hoffman) on  4-OCT-1990 17:08:03.08
      Rejected by user JONES (Jim Jones) on  4-OCT-1990 17:08:03.08
         Remark: Bad choice of variable name
      Not reviewed by  SNURD (Mortimer Snurd)
   Remark: Fix bad-status bug

VSC>
```

2.10 Cancelling Queued Replacements

If you decide that a replacement you queued should not be performed, you can cancel that replacement with the following command:

```
VSC> CANCEL REPLACEMENT SMITH-1
```

This command removes the queued replacement specified by the replacement name parameter (SMITH–1) from the source control system's database. The original module reservations associated with the cancelled replacement remain in the database. The typical reason for cancelling a replacement is that you or a reviewer found something wrong with the code to be replaced. After fixing the erroneous code, you can queue a new replacement with another REPLACE command.

With the CANCEL REPLACEMENT command, you can specify a list of replacement names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the names. The command deletes all queued replacements that match the wildcard pattern.

The CANCEL REPLACEMENT command accepts a number of qualifiers. The help text and the reference manual contain the details on the qualifiers accepted by the command. Table 2–7 lists the more commonly-used qualifiers.

Table 2–7 Qualifiers to the CANCEL REPLACEMENT command

/CONFIRM **/NOCONFIRM** /CONFIRM asks you to confirm that you want each replacement deleted. This qualifier is particularly useful if you use wildcard characters in the replacement name. /NOCONFIRM deletes each replacement without asking for confirmation. /NOCONFIRM is the default.
/LOG **/NOLOG** Specify whether informational log messages are printed after the replacement is deleted. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.

3 Displaying Information

This chapter tells you how to display information stored in the OpenVMS source control database. Earlier sections have already described how you display module reservations (Section 2.6), fold records (Section 2.8), and queued replacements (Section 2.9). This chapter describes how to display information about facilities, modules, module generations, development streams, and the users of the source control system.

3.1 Displaying Facilities

The SHOW FACILITY command lists information about the facilities into which the OpenVMS sources are divided. For example:

```
VSC> SHOW FACILITY FACNAM
```

This example displays information about facility FACNAM. If the facility name is omitted, the command displays information about all facilities in the current library. With this command, you can specify a list of facility names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the facility names. All facilities that match the wildcard patterns are displayed.

Table 3–1 lists the qualifiers for the SHOW FACILITY command and describes their functions.

Table 3–1 Qualifiers to the SHOW FACILITY command

/BRIEF
/NORMAL
/FULL

Determine the type of information displayed for each facility. /BRIEF displays the name of each facility that matches the facility-name parameters. /NORMAL displays the name and remark string for each facility; this is the default output. /FULL displays all attributes of each facility, including its name, creation date and time, and remark string.

/OUTPUT=filespec

Directs the printed output of this command to a specified file. The **filespec** parameter specifies the name of the file. VSC creates a new file with that name, directs the command's print output to that file, and prints nothing on your terminal. If this qualifier is omitted, all output appears on the terminal.

/TOTAL

Shows the total number of facilities that match the specified facility-name parameters. Only the total count is shown; other information about the individual facilities is not shown.

Examples

This is an example of the normal output of the SHOW FACILITY command:

```
VSC> SHOW FACILITY B*
Facility BACKUP          "BACKUP facility
Facility BAD             "Michael Jackson facility"
Facility BADBLK          "Bad-block faciility"
Facility BASRTL          "BASIC Run-Time Libary facility"
Facility BLAS1           "Blasphemous facility"
Facility BOOTS           "OpenVMS boot facility"
VSC>
```

The following is an example of the brief output of the SHOW FACILITY command:

```
VSC> SHOW FACILITY/BRIEF B*
Facility BACKUP
Facility BAD
Facility BADBLK
Facility BASRTL
Facility BLAS1
Facility BOOTS
VSC>
```

The following example shows the full output of the SHOW FACILITY command for one facility:

```
VSC> SHOW FACILITY/FULL SYS
Facility SYS
   Facility added to database on 19-OCT-1990 10:38:05.44
   Facility added to database by user SMITH (Jane Smith)
   Facility cannot be deleted
   Remark: OpenVMS Executive modules

VSC>
```

This example shows the effect of the /TOTAL qualifier:

```
VSC> SHOW FACILITY/TOTAL *
Number of facilities matching name *: 228
VSC>
```

3.2 Displaying Modules

The SHOW MODULE command lists information about the modules under source control. For example:

```
VSC> SHOW MODULE [FACIL]MOD.TYP
```

In this example, VSC displays information about module MOD.TYP in facility FACIL. If the facility name is omitted, the current default facility is assumed. If the whole module-name parameter is omitted, information about all modules in the default facility is shown.

With the SHOW MODULE command, you can specify a list of module names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the module names. All modules that match the wildcard patterns are displayed.

Table 3–2 lists the qualifiers for the SHOW MODULE command and describes their functions.

Table 3–2 Qualifiers to the SHOW MODULE command

/BRIEF
/NORMAL
/FULL

Determine the type of information displayed for each module. /BRIEF displays the name of each module that matches the module-name parameters. /NORMAL displays the name and remark string for each module; this is the default output. /FULL displays all attributes of each module, including its name, creation date and time, CMS history and notes attributes, reviewer and notify lists, and remark string.

/OUTPUT=filespec

Directs the printed output of this command to a specified file. The **filespec** parameter specifies the name of the file. VSC creates a new file with that name, directs the command's print output to that file, and prints nothing on your terminal. If this qualifier is omitted, all output appears on the terminal.

/STATISTICS=(keyword [, keyword...])

Computes and displays certains statistics about the modules that match the module-name parameters. The **keyword** parameters specify what statistics to display and how to sort them. The qualifier accepts the following keyword parameters:

- ALPHABETICALLY—Sort the display alphabetically by facility name or type name. If this keyword is omitted, VSC sorts the display in descending order by module count.
- FACILITY—Compute and display the number of modules in each facility.
- TYPES—Compute and display the number of modules that have each module type.

If you specify only one of these keywords, the surrounding parentheses can be omitted.

When this qualifier is used, other information about the individual modules is not shown. Use wildcard characters in the module-name parameters to select the module population for which you want statistics.

/TOTAL

Shows the total number of modules that match the specified module-name parameters. Only the total count is shown; other information about the individual modules is not shown.

Examples

This is an example of the normal output of the SHOW MODULE command:

```
VSC> SHOW MODULE [ADARTL]PPLSS_*
Module [ADARTL]PPLSS_DISPATCH.MAR     "SS dispatcher"
Module [ADARTL]PPLSS_KERNEL_WORKER.B32
                                      "Kernel worker"
Module [ADARTL]PPLSS_RESET.MAR        "Cleanup routine"
Module [ADARTL]PPLSS_SERVICES.MAR     "Service entry points"
Module [ADARTL]PPLSS_TABLE.B32        "Sys Service definitions"
Module [ADARTL]PPLSS_TABLE.REQ        "Sys Service definitions"
Module [ADARTL]PPLSS_USER_WORKER.B32  "User mode worker routines"
VSC>
```

The following is an example of the brief output of the SHOW MODULE command:

```
VSC> SHOW MODULE/BRIEF
Module [FACIL]FOO.BAR
Module [FACIL]MOD1.FOR
Module [FACIL]MOD2.FOR
Module [FACIL]MOD3.FOR
Module [FACIL]MOD4.FOR
Module [FACIL]MOD5.FOR
Module [FACIL]MOD6.FOR
VSC>
```

The following example shows the full output of the SHOW MODULE command for one module:

```
VSC> SHOW MODULE [FACIL]MOD1.FOR/FULL
Module [FACIL]MOD1.FOR
   Module added to database on  4-APR-1989 14:58:39.66
   Module added to database by user SMITH (Jane Smith)
   Module is a source module
   Module cannot be deleted
   Module may be concurrently reserved
   History information is not provided by default
   History string: C  #B
   Notes information is not provided by default
   Notes string: !  #G
   Note position: 81
   Reviewers: JONES, SNURD
   Users to notify: MERMAN
   Remark: Test module

VSC>
```

This example shows the effect of the /TOTAL qualifier:

```
VSC> SHOW MODULE/TOTAL [RMS]*
Number of modules matching name [RMS]*.*: 236
VSC>
```

3.3 Displaying Generation Histories

The SHOW GENERATION command lists historical and differential information about the module generations present in the current library. For example:

```
VSC> SHOW GENERATION [FACIL]MOD.TYP
```

This example displays information about the latest generation of module [FACIL]MOD.TYP in your default development stream. If the facility name is omitted, your default facility is assumed. If the whole parameter is omitted, the most recent generations for the default stream of all modules in the default facility are shown.

With the SHOW GENERATION command, you can specify a list of module names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the module names. The SHOW GENERATION command displays generations for all modules that match the wildcard patterns.

Each generation for a given module is named in two ways. One name is the *generation number* that VSC assigns and the other is the *generation expression* that CMS assigns. You can use either name to specify the generation to display.

VSC assigns generation numbers in sequential order to the generations for each development stream. Generation 1 is thus the first generation for the stream, generation 2 is its successor, and so on. Generation 2 of stream A can be the same as generation 2 of stream B (if the streams have not yet diverged) or they can be two different generations (if the streams have diverged).

When you refer to a generation by its generation number, you specify a semicolon and the generation number after the module name. For example, MOD.TYP;5 refers to generation 5 for the default stream (or the stream specified by the /STREAM qualifier). You can also specify a zero or negative generation number; such a generation number is relative to the latest generation for the stream. MOD.TYP;0 thus refers to the latest generation and MOD.TYP;–1 refers to the next most recent generation. If you specify the asterisk (*) wildcard character in place of the generation number (as in MOD.TYP;*), the SHOW GENERATION command displays all generations for the stream starting with the most recent generation.

CMS generation expressions are assigned a different way. The first generation for a module has generation expression 1. It can have multiple successors (for different streams) with generation expressions 2, 1A1, 1B1, 1C1, and so on. Generation 1B1 can in turn have successors 1B2, 1B1A1, 1B1B1, and so on. Each generation thus has a unique generation expression that encodes its genealogy. This expression is independent of stream. To specify a generation by its generation expression, you must use the /GENERATION qualifier. For example, MOD.TYP/GENERATION=1B1 refers to the generation of module MOD.TYP that has generation expression 1B1.

If you omit both the generation number and the /GENERATION qualifier, VSC displays the latest generation in the default or specified stream.

The SHOW GENERATION command also accepts a number of qualifiers that select subsets of the specified generations. For example, you can select all generations created before or since a certain date and time, you can select the generations created by a certain user, and you can select those generations that differ between two streams.

Table 3–3 lists the qualifiers for the SHOW GENERATION command and describes their functions.

Table 3–3 Qualifiers to the SHOW GENERATION command

/BEFORE=date-time

Displays only those generation created before the specified date and time.

/BRIEF
/FULL

Determine the type of information displayed for each generation. /BRIEF displays the module name, generation number, generation expression, creating user, and creation date and time of each generation that matches the generation-specification parameters. /FULL displays all database information for each generation.

/GENERATION=gen-expr

Specifies that the module generation with the CMS generation expression given by the **gen-expr** parameter be displayed.

/IF_SAME=stream-name
/IF_DIFFERENT=stream-name

Displays only those generations that are the same (or different) in two streams. These qualifiers select the latest generation of each specified module in the stream given by the /STREAM qualifier (or your default stream). The /IF_SAME qualifier displays this generation only if it is also the latest generation of the module in the stream given by the **stream-name** parameter. The /IF_DIFFERENT qualifier displays this generation only if it differs from the latest generation of the module in the stream given by the **stream-name** parameter. These qualifiers thus show for what modules two developments streams are still the same and for what modules the streams differ.

The latest generations of two streams are considered to be the same only if they are the identical generation with the same generation expression. VSC determines this from the contents of the source control database, not by comparing the text of the two generations.

/OUTPUT=filespec

Directs the printed output of this command to a specified file. The **filespec** parameter specifies the name of the file. VSC creates a new file with that name, directs the command's print output to that file, and prints nothing on your terminal. If this qualifier is omitted, all output appears on the terminal.

/SINCE=date-time

Displays only those generation created on or after the specified date and time.

/STATISTICS=(keyword [, keyword...])

Table 3–3 (Cont.) Qualifiers to the SHOW GENERATION command

Computes and displays certains statistics about the selected generations. The **keyword** parameters specify what statistics to display and how to sort them. The qualifier accepts the following keyword parameters:

- ALPHABETICALLY—Sort the display alphabetically by facility name or type name. If this keyword is omitted, VSC sorts the display in descending order by generation count.
- FACILITY—Compute and display the number of generations in each facility.
- TYPES—Compute and display the number of generations that have each module type.

If you specify only one of these keywords, the surrounding parentheses can be omitted.

When this qualifier is used, other information about the individual generations is not shown. Use selection qualifiers and wildcard characters in the generation specification parameters to select the generation population for which you want statistics.

/STREAM=stream-name

Specifies that the generation numbers you specify on the command parameters are for the stream specified by the **stream-name** parameter. If this qualifier is omitted, generation numbers are relative to your default stream.

/TOTAL

Shows the total number of generations that match the specified generation-specification parameters. Only the total count is shown; other information about the individual generations is not shown.

/TREE

Displays the full generation tree for each specified module. The generations are ordered by creation date (the most recent generations are shown first). For each generation that is the latest generation of some development stream, the name of that stream is displayed. This display is equivalent to the display that the CMS SHOW GENERATION /DESCENDANTS /MEMBER command gives.

/USERNAME[=username]

Displays only those generations created by the user specified by the **username** parameter. If the **username** parameter is omitted, VSC displays only those generations that you created. If the whole qualifier is omitted, VSC displays generations created by any user.

Examples

This example shows the brief output of the SHOW GENERATION command. Each generation's module name is followed by a semicolon and the VCS generation number which in turn is followed by the CMS generation expression (audit trail identifier) in parentheses. The output also shows who created the generation and when it was created:

```
VSC> SHOW GENERATION *.*
Generations for development stream MUSHROOM:
Generation [RMS]MODULE_A.REQ;2(2)             SMITH        12-DEC-1989 08:44:12
Generation [RMS]MODULE_B.REQ;25(10A12B3)      JONES         9-DEC-1989 20:34:00
Generation [RMS]MODULE_C.B32;14(9H5)          BROWN         9-DEC-1989 20:34:45
Generation [RMS]MODULE_D.B32;1(1)             MURGATROID    9-DEC-1989 20:34:57
Generation [RMS]MODULE_E.B32;8(6A2)           JONES        29-DEC-1989 13:12:32
VSC>
```

This next example shows how to display all generations of a given module that gave rise to the latest generation in your default stream:

```
VSC> SHOW GENERATION MODULE_E.B32;*
Generations for development stream MUSHROOM:
Generation [RMS]MODULE_E.B32;8(6A2)           JONES        29-DEC-1989 13:12:32
Generation [RMS]MODULE_E.B32;7(6A1)           JONES        26-DEC-1989 14:00:00
Generation [RMS]MODULE_E.B32;6(6)             SMITH        19-DEC-1989 15:37:12
Generation [RMS]MODULE_E.B32;5(5)             PORTER       12-DEC-1989 13:10:01
Generation [RMS]MODULE_E.B32;4(4)             DINGDONG     12-DEC-1989 08:43:27
Generation [RMS]MODULE_E.B32;3(3)             SMITH        12-DEC-1989 08:37:31
Generation [RMS]MODULE_E.B32;2(2)             JONES        12-DEC-1989 08:35:54
Generation [RMS]MODULE_E.B32;1(1)             JONES         9-DEC-1989 20:34:02
VSC>
```

The following example shows the full output of the SHOW GENERATION command for a specific generation:

```
VSC> SHOW GENERATION [RMS]MODULE_E.B32;8/FULL
Generation [RMS]MODULE_E.B32;8
   Generation created on 29-DEC-1989 13:12:32.25
   Generation created by user JONES (Jim Jones)
   Module is a source module
   CMS generation expression is "6A2"
   Delta letters in use:  None
   Generation created in stream V5.4
   Latest generation for stream MUSHROOM, X4PM_BIRCH, V5.4
   Module is current for stream MUSHROOM
   Remark: Add the mumble-fratz feature

VSC>
```

This example shows the output of the /TREE qualifier:

```
VSC> SHOW GENERATION/TREE [EXEC]MODULE_X.MAR
Generation tree for module [EXEC]MODULE_X.MAR:
   (5B2)    19-DEC-1989 15:37:12.37 SMITH "This time it will work for sure!"
               Latest generation for stream MUSHROOM, X4PS_MUSHROOM
   (5B1)    19-DEC-1989 15:37:12.11 SMITH "Fix whizbang feature again"
               Latest generation for stream X4PR_MUSHROOM
   (5A1)    19-DEC-1989 15:37:12.00 SMITH "Fix whizbang feature"
               Latest generation for stream X4PQ_MUSHROOM
   (5)      12-DEC-1989 13:10:01.45 PORTER "Add whizbang feature"
               Latest generation for stream PHOENIX, X4PP_PHOENIX
   (4)      12-DEC-1989 08:43:27.29 DINGDONG "Implement look-aside lists"
   (3A2)    12-DEC-1989 13:10:01.00 PORTER "Fix some bad comments"
               Latest generation for stream X4PM, X4PN, X4PO
   (3A1)    12-DEC-1989 13:10:01.56 PORTER "Add comments"
   (3)      12-DEC-1989 08:37:31.18 SMITH "Implement look-down algorithm"
               Latest generation for stream X4PK, X4PL
   (2)      12-DEC-1989 08:35:54.47 JONES "Implement new look-up algorithm"
               Latest generation for stream V5.2, V5.3, V5.4, X4PJ_BIRCH
   (1)      9-DEC-1989 20:34:02.13 JONES "Initial generation of module"

VSC>
```

3.4 Displaying Differences between Generations

VSC provides a DIFFERENCES command that determines and displays the differences between the texts of any two module generations. This is in addition to the ability to display differences between module generations through the REVIEW REPLACEMENT subsystem. The DIFFERENCES command fetches two specified generations into a scratch directory and runs the DCL DIFFERENCES command in a subprocess to display the differences between the two generations on your terminal. The following example shows how to use the DIFFERENCES command:

```
VSC> DIFFERENCES [FACIL]MOD.TYP;5 [FACIL]MOD.TYP;2
```

This example fetches generation 5 of module MOD.TYP in facility FACIL and generation 2 of the same module, computes the differences between those two files, and displays the differences. These generations are for your default development stream. The second parameter can be abbreviated to just a semicolon and the generation number (such as ‘‘;2’’) if the module name is the same for both generations. If the second parameter is omitted entirely, the specified generation is compared with the immediately preceding generation in the same development stream. If the second parameter is omitted in the previous example, generation 5 is thus compared with generation 4. Zero and negative generation numbers are also allowed; these generation numbers are relative to the latest generation of the module for the stream. If the generation number is omitted, generation 0 (the latest generation) is assumed by default.

The generations can also be identified by their CMS generation expressions (such as 10A2) with the /GENERATION qualifier. If both generations are generations of the same module (the usual case), this form of the qualifier is allowed:

```
VSC> DIFFERENCES [FACIL]MOD.TYP/GENERATION=(10A2,11B1)
```

This command compares the generations of [FACIL]MOD.TYP identified by the CMS generation expressions 10A2 and 11B1.

If you do not specify the /GENERATION qualifier, VSC uses the specified (or default) generation numbers and assumes the generation numbers to be relative to your default development stream. You can specify that you want to compare generations from other streams with the /STREAM qualifier:

```
VSC> DIFFERENCES [FACIL]MOD.TYP/STREAM=PHOENIX
```

This command compares the two most recent generations of the specified module in stream PHOENIX. To compare generations from different streams, you can either specify the /STREAM qualifier on both command parameters or you can use this special form of the qualifier with just one parameter:

```
VSC> DIFFERENCES [FACIL]MOD.TYP/STREAM=(V5.4,PHOENIX)
```

This command computes the differences between the most recent generations of the module for streams V5.4 and PHOENIX.

The DIFFERENCES command also accepts all qualifiers that the DCL DIFFERENCES command accepts. These qualifiers are simply passed through to the subprocess that runs the OpenVMS DIFFERENCES utility.

To perform its action, the DIFFERENCES command fetches the specified generations into a scratch directory. If you have defined logical name VDE$SCRATCH, VSC uses the directory given by that logical name as the scratch directory. If this logical name is not defined, VSC creates a subdirectory VDE$SCRATCH.DIR (unless it exists already) in directory SYS$SCRATCH and stores the files in that subdirectory. (By default, SYS$SCRATCH is your login directory.) VSC then spawns a subprocess to run the DCL DIFFERENCES command and display the differences. VSC deletes the fetched files at the end of the command unless you specify the /KEEP qualifier. The difference output goes to your terminal by default, but you can direct it to a file with the /OUTPUT qualifier.

Table 3–4 lists the qualifiers for the DIFFERENCES command and describes their functions.

Table 3–4 Qualifiers to the DIFFERENCES command

/GENERATION=gen-expr

Specifies that the module generation that has the CMS generation expression given by the **gen-expr** parameter be compared. If this qualifier is omitted, the command uses the generation given by the generation number on the module-name parameter. If both this qualifier and the generation number are omitted, the command uses the latest generation of the module for the specified development stream.

/KEEP

Specifies that the fetched and compared module generations be retained after the command completes. If this qualifier is omitted, the files containing those generations are deleted after the differences are computed.

/STREAM=stream-name

Specifies that the generation to be compared should be taken from the development stream given by the **stream-name** parameter. If this qualifier is omitted, the generation is taken from your default stream unless you use the /GENERATION qualifier.

/CHANGE_BAR
/COMMENT_DELIMITER
/IGNORE
/MAXIMUM_DIFFERENCES
/MATCH
/MERGED
/MODE
/NUMBER
/NONUMBER
/OUTPUT
/PARALLEL
/SEPARATED
/SLP
/WIDTH
/WINDOW

Table 3–4 (Cont.) Qualifiers to the DIFFERENCES command

These qualifiers are passed through to the spawned OpenVMS DIFFERENCES command without change. See the description of the DIFFERENCES command in the *OpenVMS DCL Dictionary* manual for a description of these qualifiers and their parameters.

3.5 Displaying Development Streams

The SHOW STREAM command lists information about the development streams defined in the source control database. Streams include past versions of OpenVMS, versions currently under development, build base levels, and off-to-the-side development streams. The following example illustrates the syntax of the SHOW STREAM command:

```
VSC> SHOW STREAM PHOENIX
```

This example displays information about stream PHOENIX. If the stream name is omitted, the command shows information about all streams defined in the database. With this command, you can specify a list of stream names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the stream names. All streams that match the wildcard patterns are displayed.

Table 3–5 lists the qualifiers for the SHOW STREAM command and describes their functions.

Table 3–5 Qualifiers to the SHOW STREAM command

/BRIEF
/NORMAL
/FULL

Determine the type of information displayed for each stream. /BRIEF displays the name of each stream that matches the stream-name parameters. /NORMAL displays the name and remark string for each stream; this is the default output. /FULL displays all attributes of each stream, including its name, creation date and time, reviewers, and remark string.

/OPEN
/FROZEN
/CLOSED

Displays only those streams with the specified status. /OPEN displays only those streams that are open for replacements. /FROZEN displays only those streams that are "frozen" so that replacements are not currently possible. /CLOSED displays only those streams that are permanently closed against replacements. If you omit all of these qualifiers, VSC displays all streams regardless of status.

/OUTPUT=filespec

Directs the printed output of this command to a specified file. The **filespec** parameter specifies the name of the file. VSC creates a new file with that name, directs the command's print output to that file, and prints nothing on your terminal. If this qualifier is omitted, all output appears on the terminal.

Table 3–5 (Cont.) Qualifiers to the SHOW STREAM command

/SUCCESSOR

Shows the full stream successor chain for each displayed development stream. This information shows to what streams changes will be propagated from a given stream.

/TOTAL

Shows the total number of streams that match the specified stream-name parameters. Only the total count is shown; other information about the individual streams is not shown.

Examples

This is an example of the normal output of the SHOW STREAM command:

```
VSC> SHOW STREAM
Stream MUSHROOM          "Maintenance stream, formerly V5.4-1"
Stream PHOENIX           "V6.0 (?) release stream (née Thunderbolt)"
Stream V4.4              "SDC release of VMS V4.4"
Stream V5.0              "SDC release of VMS V5.0"
Stream V5.2              "SDC release of VMS V5.2"
Stream V5.4              "SDC release of VMS V5.4"
VSC>
```

The following is an example of the brief output of the SHOW STREAM command:

```
VSC> SHOW STREAM/BRIEF *
Stream MUSHROOM
Stream PHOENIX
Stream V4.4
Stream V5.0
Stream V5.2
Stream V5.4
VSC>
```

The following example shows the full output of the SHOW STREAM command for one stream:

```
VSC> SHOW STREAM/FULL MUSHROOM
Stream MUSHROOM
   Stream created on 19-OCT-1990 10:37:00.77
   Stream created by user SMITH (Jane Smith)
   Stream owner is user SNURD (Mortimer Snurd)
   Parent stream is V5.4
   Successor streams:  PHOENIX
   Stream is part of the main development thread
   Stream cannot be deleted
   CMS class is maintained for stream
   Replacements are queued for stream
   Stream is open for checkins
   Remark: V5.4-1 release class

VSC>
```

This example shows the effect of the /SUCCESSOR qualifier:

```
VSC> SHOW STREAM V5.3 /SUCCESSOR
Stream V5.3
   Successor chains:
      V5.3 -> V5.3-1 -> V5.3-2 -> V5.4 -> MUSHROOM -> PHOENIX

VSC>
```

3.6 Displaying Users

The SHOW USER command lists information about the users of the OpenVMS source control system. These users are the OpenVMS developers and others who are allowed to access the OpenVMS current library. The following example shows how to use the SHOW USER command:

```
VSC> SHOW USER SMITH
```

This example displays information about the user with OpenVMS user name SMITH. If the user name is omitted, the command shows information about all users recorded in the database. With this command, you can specify a list of user names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the user names. All users whose OpenVMS user names match the wildcard patterns are displayed.

The qualifiers available on the SHOW USER command are described in Table 3–6.

Table 3–6 Qualifiers to the SHOW USER command

/BRIEF **/FULL** Determine the type of information displayed for each user. /BRIEF displays the OpenVMS user name and the full name (first name and last name) of each user who matches the user name parameters. /FULL displays all database information about each user.
/OUTPUT=filespec Directs the printed output of this command to a specified file. The **filespec** parameter specifies the name of the file. VSC creates a new file with that name, directs the command's print output to that file, and prints nothing on your terminal. If this qualifier is omitted, all output appears on the terminal.
/TOTAL Shows the total number of users that match the specified username parameters. Only the total count is shown; other information about the individual users is not shown.

Examples

This is an example of the brief output of the SHOW USER command:

```
VSC> SHOW USER S*
User SMITH (Jane Smith)
User SNURD (Mortimer Snurd)
VSC>
```

The following example shows the full output of the SHOW USER command:

```
VSC> SHOW USER/FULL SNURD
User SNURD (Mortimer Snurd)
   User added to database on 19-OCT-1990 22:12:40.91
   User added to database by user JONES (Jim Jones)
   Remark: VMS Developer
   Default privileges:
      CREMOD RESREP
   Authorized privileges:
      CREMOD RESREP

VSC>
```

4 Reviewing Queued Replacements

This chapter tells you how to review modules on behalf of another developer who has queued those modules for replacement. When you are a reviewer for a replacement, it is your responsibility to review the code being replaced for errors, quality, and appropriateness for the development stream. Based on these factors, you must then record in the source control database that you either accept or reject the proposed replacement. This chapter tells you how to perform these operations.

4.1 Reviewing Modules Queued for Replacement

When you queue a replacement with the REPLACE command, you can use the /REVIEWER qualifier to that command to specify who should review your replacement. The source control system might assign additional reviewers to your replacement because some developers automatically review every change to certain modules or development streams. VSC records the reviewer list in the database and sends mail to all the reviewers to request that they review your replacement. The SHOW REPLACEMENT/FULL command will display the reviewer list for you.

Once a replacement has been queued, the reviewers for that replacement are allowed to review it and to vote to accept or reject it. Reviewing the replacement means examining the source changes about to be replaced into the source code library to determine if they appear correct in style, substance, and appropriateness. The REVIEW REPLACEMENT command helps you review the source changes that comprise a replacement. For example, if you are a reviewer for the replacement SMITH–1, you should enter the following command to review it:

```
VSC> REVIEW REPLACEMENT SMITH-1
```

This command places you in a VSC subsystem that presents the *VSC_REVIEW>* prompt. This subsystem accepts a limited set of commands that allow you to examine the source code being replaced, differences between the new source code and the previous code in the source code library, and text files written by the replacement's submittor or by other reviewers. When you are done reviewing the replacement, you enter the EXIT command or press Ctrl/Z to return to the normal *VSC>* prompt.

To perform its function, the REVIEW REPLACEMENT command first creates a *review list* for the specified replacement. The elements of the review list consist of the *information file* created by the user queing the replacement (if such a file was specified), the *comment files* created by previous reviewers (if such files were specified), and the modules queued for replacement. The review list will also contain a log file from the PERFORM REPLACEMENT command if that command was used but encountered errors when it tried to perform the replacement. One element of the review list is always designated as the *current element*; that is the

element whose contents you can display. Initially, the first element on the list is the current element, but various commands allow you to change the current element.

The commands in the REVIEW REPLACEMENT subsystem let you display the review list, navigate back and forth among the elements of the list to select the current element, and display the contents of the current element. To display an information or comment file, VSC simply types the file on your terminal. To diplay a module being reviewed, VSC can show the source being replaced, the original source currently in the library, and the differences between these two. VSC also lets you use an editor to examine files if you prefer.

The REVIEW REPLACEMENT command accepts several qualifiers that let you control the composition of the review list. Table 4–1 lists these qualifiers and describes their functions.

Table 4–1 Qualifiers to the REVIEW REPLACEMENT command

/COMMENT
Causes all comment files provided by previous reviewers to be included in the review list.
/INFORMATION
Causes the replacement owner's information file to be included in the review list.
/MODULE
Causes all source modules that are part of the replacement to be included in the review list.
You can use these qualifiers in combination to include the elements you want in the review list. If you omit all three qualifiers, the command automatically includes all three kinds of elements in the review list.

Subsystem Commands

As already mentioned, the REVIEW REPLACEMENT subsystem accepts a special set of commands and keeps prompting for such commands until you enter the EXIT command. Table 4–2 lists the REVIEW REPLACEMENT subsystem commands and describes their functions.

Table 4–2 Commands Accepted in the REVIEW REPLACEMENT Subsystem

EXIT
Exits from the subsystem and returns you to the *VSC>* prompt. Pressing Ctrl/*Z* performs the same function.
DIRECTORY
Displays the review list for the replacement being reviewed. The command displays one line for each information file, comment file, log file, and module that is part of the queued replacement.
CURRENT

Table 4–2 (Cont.) Commands Accepted in the REVIEW REPLACEMENT Subsystem

Displays the name of the current element in the review list. The current element is the file or module that VSC displays if you enter a SHOW or EDIT command.

NEXT

Makes the next element on the review list the current element and displays the name of that element. Pressing the KP0 key (the 0 key on the numeric keypad) is equivalent to entering a NEXT command. As a result, you can step from element to element by repeatedly pressing the KP0 key. Similarly, the COMMA (,) key on the numeric keypad enters the NEXT and SHOW commands with a single keystroke and the MINUS (−) key on the numeric keypad enters the NEXT and EDIT commands.

BACK

Makes the previous element on the review list the current element and displays the name of that element.

FIRST

Makes the first element on the review list the current element and displays the name of that element.

LAST

Makes the last element on the review list the current element and displays the name of that element.

SELECT mod-name

Makes the review-list element specified by the **mod-name** parameter the current element and displays the name of that element. To select a module, the **mod-name** parameter must specify the name of that module in the normal VSC module-name syntax. To select an information or comment file, the **mod-name** parameter must specify the user name of the developer who created that information or comment file. To select the log file that VSC created if an attempt to perform the replacement failed, the **mod-name** parameter must specify the file name LOG.

SHOW

Types the contents of the current review-list element on your terminal. The SHOW command takes an optional keyword parameter. If the current element is a module, the NEW keyword displays the new version of the module queued for replacement, the OLD keyword displays the old version of that module currently in the library, and the DIFFERENCES keyword displays the differences between the new and the old versions. If no keyword is given, VSC displays differences by default. If the current element is an information, comment, or log file, VSC displays the text of that file and ignores any keyword you specify.

EDIT

Table 4–2 (Cont.) Commands Accepted in the REVIEW REPLACEMENT Subsystem

Spawns an editing session to display the current review-list element. The EDIT command takes an optional keyword parameter. If the current element is a module, the NEW keyword displays the new version of the module queued for replacement, the OLD keyword displays the old version of that module currently in the library, and the DIFFERENCES keyword displays the differences between the new and the old versions. If no keyword is given, VSC displays differences by default. If the current element is an information, comment, or log file, VSC displays the text of that file and ignores any keyword you specify.

Use the SET EDITOR command at the *VSC>* prompt to specify your preferred editor. If you have not specified an editor, VSC uses the system default text editor by default.

ACCEPT ["remark-string"]

Records in the VSC database that you accept the replacement currently being reviewed. You must be a reviewer for the replacement to use this command. You can specify the optional remark string to give a reason for accepting the replacement. This command accepts the same qualifiers as the VSC-level ACCEPT REPLACEMENT command.

REJECT "remark-string"

Records in the VSC database that you reject the replacement currently being reviewed. You must be a reviewer for the replacement to use this command. The remark string should specify a reason for rejecting the replacement. If you omit the remark parameter, VSC prompts you for a remark. This command accepts the same qualifiers as the VSC-level REJECT REPLACEMENT command. Like REJECT REPLACEMENT, the REJECT command lets you specify a comment file to more fully explain your reasons for rejecting the replacement.

SPAWN

Spawns a subprocess. This command accepts the same parameters and qualifiers as the VSC-level SPAWN command.

ATTACH

Attaches to a specified subprocess. This command accepts the same parameters and qualifiers as the VSC-level ATTACH command.

The normal way to use the REVIEW REPLACEMENT subsystem is to step from one review-list element to the next with the NEXT command (or the KP0 key on the numeric keypad) and to then examine each element with the SHOW or EDIT command. To examine an information, comment, or log file, just use the SHOW command. To examine a module, you should generally start by examining the differences between the new and old versions of the module. The SHOW or EDIT command without a keyword parameter will create and display the differences file. If that does not give you enough information, use the EDIT NEW command to see the full text of the source module to be replaced. Use the EDIT OLD command if you want to see the old source code for comparison. When you are done reviewing the replacement, enter the EXIT command to return to the *VSC>* prompt.

To traverse and display the review-list elements slightly faster, you can press the COMMA (,) key on the numeric keypad to perform the NEXT and SHOW commands with a single keystroke. Similarly, you can press the MINUS (–) key on the numeric keypad to perform the NEXT and EDIT commands with a single keystroke.

The new version, old version, and differences file for a module are all stored in the queued replacement's staging area. VSC fetches the old version and generates the differences file the first time a reviewer displays differences. For subsequent reviewers, VSC uses the files that already exist in the staging area.

Examples

In the following example, Jane Smith has queued a replacement consisting of two modules, [FACIL]FOO.MAR and [RMS]FUMBLE.B32. She has also supplied an information file for the replacement. If you were a reviewer for this replacement, you might first enter the following command to display information about it:

```
VSC> SHOW REPLACEMENT/FULL SMITH-5
Queued replacement SMITH-5
   Replacement queued by user SMITH (Jane Smith)
   Replacement queued on  4-OCT-1997 16:55:02.06
   Replacement staging directory is VSC$STAGEDISK:[VMS.STAGE.VDE$STG_1.VDE$REP_115]
   Replacement review status: Rejected
   Replacement is for stream PHOENIX
   Unique replacement number is 115
   Reservations in replacement:
      Reservation 1 of [FACIL]FOO.MAR for stream PHOENIX
      Reservation 1 of [RMS]FUMBLE.B32 for stream PHOENIX
   Replacement contains 2 reservations
   Reviewers for replacement:
      Not reviewed by  BROWN (Bill Brown) on  4-OCT-1997 17:08:03.08
      Rejected by user JONES (Jim Jones) on  4-OCT-1997 17:08:03.08
         Remark: Bad choice of variable name
      Not reviewed by  SNURD (Mortimer Snurd)
   Remark: Fix bad-status bug

VSC>
```

You would then enter the following command to review it:

```
VSC> REVIEW REPLACEMENT SMITH-1
VSC_REVIEW>
```

At the subsystem prompt, you might want to see what modules and files are in the replacement's review list. The DIRECTORY command performs this function:

```
VSC_REVIEW> DIRECTORY
Modules and files being reviewed for replacement SMITH-1:
   Information file by user SMITH (Jane Smith)
   Comment file by reviewer JONES (Jim Jones) rejecting replacement
   Module [FACIL]FOO.MAR
   Module [RMS]FUMBLE.B32

VSC_REVIEW>
```

The review list includes SMITH's information file, a comment file from another reviewer JONES, and the two source modules. The information file is initially the current element since it is the first element on the list. To display that file, enter the SHOW command:

```
VSC_REVIEW> SHOW

Please perform this replacement after replacement SNURD-2 has run.
This replacement depends on a data cell defined by that code.

VSC_REVIEW>
```

The next element on the list is a comment file that user JONES created when he voted to reject the replacement. Such a comment file normally gives the reason for rejecting the replacement. To select and display that comment file, enter the NEXT command followed by the SHOW command:

```
VSC_REVIEW> NEXT

Comment file by reviewer JONES (Jim Jones) rejecting replacement

VSC_REVIEW> SHOW

The code looks correct, but the name of variable USER_IS_A_JERK should
be changed to something more appropriate, such as USER_ERROR_FLAG.

VSC_REVIEW>
```

To see the differences for the first module on the review list (module FOO.MAR), enter another NEXT command followed again by a SHOW command:

```
VSC_REVIEW> NEXT

Module [FACIL]FOO.MAR

VSC_REVIEW> SHOW
%VDE-I-FETCHED, generation [FACIL]FOO.MAR;5(5) fetched from library
%VDE-I-OUTFILE, output file is VSC$STAGEDISK:[VMS.STAGE.VDE$STG_1.VDE$REP_115
.FACIL]FOO.MAR_VDE$OLD;

************
File VSC$STAGEDISK:[VMS.STAGE.VDE$STG_1.VDE$REP_115.FACIL]FOO.MAR;1
   46       Init_Linked_List(LST_PTR);
   47       X = 5;
******
File VSC$STAGEDISK:[VMS.STAGE.VDE$STG_1.VDE$REP_115.FACIL]FOO.MAR_VDE$OLD;1
   46       X = 5;
************

Number of difference sections found: 1
Number of difference records found: 1

DIFFERENCES /IGNORE=()/MERGED=1/OUTPUT=VSC$STAGEDISK:[VMS.STAGE.VDE$STG_1.VDE$REP_115.FACIL]
FOO.MAR_VDE$DIFF;1-
    VSC$STAGEDISK:[VMS.STAGE.VDE$STG_1.VDE$REP_115.FACIL]FOO.MAR;1-
    VSC$STAGEDISK:[VMS.STAGE.VDE$STG_1.VDE$REP_115.FACIL]FOO.MAR_VDE$OLD;1

VSC_REVIEW>
```

If the difference file had been large, it might have been easier to display it with the EDIT command. Also, to see the actual code to be replaced, you could use the EDIT NEW command at this point.

To examine module FUMBLE.B32, you repeat the NEXT and SHOW commands, as shown in the previous example. If you enter yet another NEXT command, VSC tells you that there are no more modules on the review list. You can then enter the EXIT command to return to the *VSC>* prompt:

```
VSC_REVIEW> EXIT
VSC>
```

At this point, you can enter normal VSC commands again.

4.2 Accepting or Rejecting a Queued Replacement

Once you have reviewed the changes that make up a queued replacement, you should record whether the replacement is acceptable or not. If you find the replacement acceptable, you enter the following command:

```
VSC> ACCEPT REPLACEMENT SMITH-1 "Remark string"
```

This command records, in the source-control database, that you have reviewed the source changes in the specified queued replacement (SMITH–1) and have found them to be acceptable. Optionally, you can specify a remark string (using the /REMARK qualifier or a remark parameter as shown) or a comment file (using the /INPUT qualifier) to explain or comment on your decision to accept the replacement.

If you decide that the source changes in the queued replacement are not acceptable, you reject the replacement with the following command:

```
VSC> REJECT REPLACEMENT SMITH-1 "Remark string"
```

This command records, in the source-control database, that you have reviewed the source changes in the specified queued replacement (SMITH–1) and have rejected them for some reason. This command requires you to specify a one-line remark string that summarizes why you reject the replacement. You can specify the remark string with the /REMARK qualifier or with a remark parameter as shown. If you do not specify a remark, VSC prompts you for the remark string. You must always give a reason for rejecting a replacement so the owner of the replacement learns how to make it acceptable.

If your reasons for rejecting the replacement are too long to fit in the remark string, you should explain your reasons in a comment file. You can edit this file ahead of time and specify its file name with the /INPUT qualifier. Alternatively, you can specify the /EDIT qualifier, which causes VSC to invoke an editor to create the comment file. The owner of the replacement can read your comment file through the REVIEW REPLACEMENT subsystem. If you specify neither the /INPUT nor the /EDIT qualifier for the REJECT REPLACEMENT command, VSC asks you whether you want to edit a comment file. For the ACCEPT REPLACEMENT command, the default is to not provide a comment file.

If you have accepted or rejected a replacement and change your mind, you can enter another ACCEPT or REJECT REPLACEMENT command to reverse the earlier judgement. You can also enter another such command to modify your remark string or comment file.

You can accept or reject a replacement only if you are a reviewer for that replacement. Anyone can review a queued replacement using the REVIEW REPLACEMENT command, but only designated reviewers can vote for or against a replacement. Also, the votes of the reviewers only constitute advice to the release project leader to help him or her decide whether to perform the replacement or not. The project leader can decide

for or against a given replacement regardless of what the reviewers say. Normally, however, the project leader relies on the reviewers to certify the quality and correctness of the replacement and takes their advice accordingly.

Table 4–3 lists the qualifiers for the ACCEPT REPLACEMENT and REJECT REPLACEMENT commands and describes their functions.

Table 4–3 Qualifiers to the ACCEPT REPLACEMENT and REJECT REPLACEMENT commands

/INPUT=filespec

Specifies a text file with comments about the replacement, such as reasons for rejecting it or descriptions of possible improvements. The source control system copies this file to the replacement's staging area. If you omit the directory specification from this qualifier, the file is taken from your default directory.

/EDIT

Specifies that VSC should start an editing session for you to create a text file with comments about the replacement. These comments can give reasons for rejecting the replacement or descriptions of possible improvements. The editor creates the file in the replacement's staging area. If you specify neither the /INPUT nor the /EDIT qualifier for the REJECT REPLACEMENT command, VSC asks you whether you want to edit a comment file.

Use the SET EDITOR command to specify your preferred editor. If you have not specified an editor, VSC defaults to the system default text editor.

/LOG
/NOLOG

Specify whether informational log messages are printed after the replacement is accepted or rejected. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.

/REMARK="string"

Specifies a one-line remark string to be associated with your acceptance or rejection of the replacement. If you omit this qualifier from the REJECT REPLACEMENT command, the command prompts you for a remark string. (A replacement can thus be accepted without giving a reason but cannot be rejected without a stated cause.)

/USERNAME=username

Specifies that the replacement should be accepted or rejected on behalf of the user whose OpenVMS user name is given by the **username** parameter. You must have a VSC privilege (USERNAME) to use this qualifier; it is meant for release project leaders and others who manage the OpenVMS libraries, and is intended to be used only in special or unusual situations.

Use the SHOW REPLACEMENT command (see Section 2.9) to display the status of a replacement being reviewed. For each replacement, this command can list the reviewers and whether each reviewer has accepted, rejected, or not reviewed the replacement.

5 Project Leader Operations

This chapter explains how to perform those source control operations normally performed by release project leaders and by maintainers of the various VDE libraries. These operations include performing queued replacements, inserting generations into development streams, removing generations from streams, deleting generations, specifying reviewers for streams and modules, and creating build streams.

These operations are typically not performed by most developers. All commands here require VSC privileges. *Many of the commands documented in this section can have serious adverse effects on a VDE library.*

5.1 Performing Queued Replacements

If you are the project leader for a given development stream, you are responsible for performing all replacements queued for that stream. To see what replacements are queued for your stream, use the SHOW REPLACEMENT command (see Section 2.9) with the /STREAM qualifier. The /STREAM qualifier selects those replacements that are for your stream. You might also want to use the /FULL qualifier to display the list of reviewers for each replacement and their votes to accept or reject it. However, even the brief SHOW REPLACEMENT output summarizes the review status of a replacement. You should also use the REVIEW REPLACEMENT command to read the queuing user's information file since that file might have information meant for you.

Based on the advice of the reviewers and your own judgement, you must decide whether to perform each such replacement. Once you have decided to perform a given replacement, you do so by entering the PERFORM REPLACEMENT command. For example:

```
VSC> PERFORM REPLACEMENT SMITH-1 /STREAM=BIRCH
```

This command performs the queued replacement specified by the replacement name parameter (SMITH–1). This replacement must belong to stream BIRCH. All modules included in the replacement are copied from the source control system's staging area to the appropriate CMS libraries. The new module generations are inserted into the development stream for which the replacement was queued and all successor streams reachable through change propagation. The queued replacement and the associated module reservations are then deleted from the database.

The PERFORM REPLACEMENT command accepts a list of replacement names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the names. The command performs all queued replacements that match the wildcard patterns and belong to the specified stream or streams.

The /STREAM qualifier specifies that replacement should be performed only if they belong to a specified stream. The stream name can contain wildcard characters, in which case VSC performs all replacements that match both the specified replacement names and the specified stream name. If you omit the /STREAM qualifier, VSC performs only those specified replacements that belong to your default stream.

Since the PERFORM REPLACEMENT command can be time-consuming, you might want to include it in a DCL command file that you run as a batch job.

The log messages that the command prints indicate that each module has been replaced into the appropriate streams. If a module is replaced into multiple streams due to change propagation, you will see one message per stream. Other messages tell you if VSC created fold records. VSC also prints an informational message for each replacement that matches the specified replacement names but is not performed because it does not belong to a requested stream.

In some unusual circumstances, you might also see a message that tells you that a module generation did not get the same generation expression (such as 5A1) as VSC expected when the module was reserved. In this case, you should notify the developer since he or she might have used the expected generation expression as an audit trail identifier in code comments.

VSC also includes all log messages in a mail message that it sends to the developer who queued the reservation. The developer will therefore see what happened during the replacement and will be aware of any errors that arose. If errors occurred, you should work with the developer to resolve the errors so that replacement can be performed again.

The PERFORM REPLACEMENT requires the VSC privilege PERFREP.

Table 5–1 lists the qualifiers for the PERFORM REPLACEMENT command and describes their functions.

Table 5–1 Qualifiers to the PERFORM REPLACEMENT command

/CONFIRM
/NOCONFIRM

/CONFIRM asks you to confirm that you want each replacement performed. This qualifier is particularly useful if you use wildcard characters in the replacement name. /NOCONFIRM performs each replacement without asking for confirmation. /NOCONFIRM is the default.

/LOG
/NOLOG

Specify whether informational log messages are printed after the replacement is performed. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.

/STREAM=stream-name

Table 5–1 (Cont.) Qualifiers to the PERFORM REPLACEMENT command

	Specifies that only those replacements that belong to certain streams be performed. The **stream-name** parameter specifies a stream name, possibly containing the asterisk (*) and percent sign (%) wildcard characters. VSC performs a given replacement only if it belongs to a stream whose name matches the **stream-name** parameter. If you omit this qualifier, VSC performs only those replacements that belong to your default stream. For each replacement that matches a replacement name parameter but does not belong to a stream matching the **stream-name** parameter, VSC prints an informational message saying that the replacement was not performed.

Examples

This is an example of the output from the PERFORM REPLACEMENT command:

```
VSC> SET STREAM PHONEIX
VSC> PERFORM REPLACEMENT SMITH-1
%VDE-I-REPLACED, generation [FACIL]FOO.MAR;6(6) replaced into stream PHOENIX
%VDE-I-REPLACED, generation [RMS]FUMBLE.B32;2(2) replaced into stream PHOENIX
%VDE-I-REPPERF, replacement SMITH-1 performed
%VDE-I-COMMIT, database transaction has successfully committed
VSC>
```

5.2 Inserting Generations

In addition to providing automatic change propagation, the source control system gives you a manual way to insert a change already made to one development stream into another stream. The INSERT GENERATION command makes an existing generation of a module into the latest generation for a specified development stream. This command is mainly used to take bug fixes and other changes that have already been replaced into some development stream and insert them into a base level or variant stream that needs those changes to build correctly. This command requires the VSC privilege PERFREP. The following example shows how to use the INSERT GENERATION command:

```
VSC> INSERT GENERATION [FACIL]MOD.TYP/STREAM=PHOENIX P-BUILD "remark"
```

This example inserts the latest generation of module [FACIL]MOD.TYP in stream PHOENIX into stream P–BUILD. PHOENIX is thus the source stream and P–BUILD is the destination stream for the generation. The command also inserts that generation into the corresponding CMS class if the destination stream maintains a CMS class. The quoted remark string is optional; if specified, it should explain the reason for inserting the generation into the stream.

You can specify a generation number for the source stream if you want to select a generation other than the latest one. Zero and negative generation numbers are allowed; they specify generation numbers relative to the latest generation for the source stream. You can also use the qualifier /GENERATION=*gen-expr* to identify the generation by its CMS generation expression.

If you omit both the /STREAM qualifier and the /GENERATION qualifier, VSC prompts you for the source stream.

The inserted generation's position in the source stream is not affected by this command. In the previous example, the selected generation thus winds up being the latest generation for both stream PHOENIX and stream P–BUILD.

Table 5–2 lists the qualifiers for the INSERT GENERATION command and describes their functions.

Table 5–2 Qualifiers to the INSERT GENERATION command

/CONFIRM
/NOCONFIRM

/CONFIRM asks you to confirm that you want each generation inserted into the specified destination stream. This qualifier is particularly useful if you use wildcard characters in the module name. /NOCONFIRM inserts each generation without asking for confirmation. /NOCONFIRM is the default.

/GENERATION=gen-expr

Specifies that the module generation that has the CMS generation expression given by the **gen-expr** parameter be inserted into the specified destination stream. If this qualifier is omitted, the command selects for insertion the generation given by the generation number on the module-name parameter. If both this qualifier and the generation number are omitted, the command selects the latest generation of the module in the stream you specified with the /STREAM qualifier.

/LOG
/NOLOG

Specify whether informational log messages are printed after the generations are inserted. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.

/STREAM=stream-name

Specifies that the generation should be selected from the source stream given by the **stream-name** parameter. This qualifier thus determines the interpretation of the generation number on the module-name parameter. This qualifier is ignored if you use the /GENERATION qualifier to select the module generation. If you omit both the /STREAM qualifier and the /GENERATION qualifier, VSC prompts you for the source stream.

5.3 Removing Generations

Sometimes it is necessary to remove a generation from a given development stream so that the module no longer has a latest generation in that stream. For example, sometimes one needs to undo an INSERT GENERATION command directed to the wrong stream. After you have removed a module generation from a stream, the module no longer ''belongs'' to that stream and can no longer be fetched, reserved, or built for the stream. The REMOVE GENERATION command removes a generation from a stream. For example:

```
VSC> REMOVE GENERATION [FACIL]MOD.TYP/STREAM=FOO
```

This example removes the latest generation of module MOD.TYP in facility FACIL from stream FOO. Afterwards, no generation of MOD.TYP is the latest generation of the module for stream FOO; the module is no longer represented in that stream.

The removed generation remains in the VDE library, and it can still be the latest generation of the module for some other stream. The REMOVE GENERATION command thus breaks the association between the module generation and the specified stream but does not delete the generation.

The REMOVE GENERATION command accepts a list of module names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the module names. The latest generations of those modules that match the wildcard patterns are removed from the specified stream.

Keep in mind that the REMOVE GENERATION command may not be the command you actually want to use. It is not very common that you want a module to not belong to a given stream at all; it is much more common that you want some other generation of the module to be the latest generation for the stream. Before you use the REMOVE GENERATION command, you should thus make sure that INSERT GENERATION or DELETE GENERATION is not the appropriate command.

The REMOVE GENERATION command requires the VSC privilege PERFREP.

Table 5–3 lists the qualifiers for the REMOVE GENERATION command and describes their functions.

Table 5–3 Qualifiers to the REMOVE GENERATION command

/CONFIRM **/NOCONFIRM** /CONFIRM asks you to confirm that you want each generation removed from the specified stream. This qualifier is particularly useful if you use wildcard characters in the module names. /NOCONFIRM removes each generation without asking for confirmation. /NOCONFIRM is the default.
/LOG **/NOLOG** Specify whether informational log messages are printed after the generations are removed. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.
/STREAM=stream-name Specifies that the latest generations of the specified modules should be removed from the stream given by the **stream-name** parameter. If this qualifier is omitted, VSC prompts you for the stream name.

5.4 Deleting Generations

When it is necessary to undo a change to a module, you can delete the latest generation of the module in a specified stream with the DELETE GENERATION command. This command requires the VSC privilege PERFREP. The following example shows how to use the DELETE GENERATION command:

```
VSC> DELETE GENERATION [FACIL]MOD.TYP/STREAM=FOO
```

This example deletes the most recent generation of module MOD.TYP in facility FACIL from stream FOO. The previous generation then becomes the latest generation for that stream. The command also inserts the previous generation into the CMS class that corresponds to that stream. The deleted generation remains in the CMS library until it has been deleted from all streams that it belongs to.

The DELETE GENERATION command can only delete the latest generation in a stream; it cannot delete generations in the middle of a stream's generation chain. The purpose of the command is only to allow a recent bad replacement to be undone when that is necessary. Usually, however, developers should fix bad replacements by reserving the module, fixing it, and replacing it again.

The DELETE GENERATION command accepts a list of module names separated by commas, and you can use the asterisk (*) and percent sign (%) wildcard characters in the module names. The latest generations of those modules that match the wildcard patterns are deleted from the VDE library.

Table 5–4 lists the qualifiers for the DELETE GENERATION command and describes their functions.

Table 5–4 Qualifiers to the DELETE GENERATION command

/CONFIRM **/NOCONFIRM** /CONFIRM asks you to confirm that you want each generation deleted from the specified stream. This qualifier is particularly useful if you use wildcard characters in the module names. /NOCONFIRM deletes each generation without asking for confirmation. /NOCONFIRM is the default.
/LOG **/NOLOG** Specify whether informational log messages are printed after the generations are deleted. /LOG causes such messages to be printed and /NOLOG suppresses them. /LOG is the default.
/STREAM=stream-name Specifies that the latest generations of the specified modules should be deleted from the stream given by the **stream-name** parameter. If this qualifier is omitted, VSC prompts you for the stream name.

5.5 Marking Module Generations Obsolete

When a module has become obsolete, it can be marked as such in the VSC database using the MODIFY GENERATION/OBSOLETE command. This command will prevent further propogation of the specified module into any successor streams. The following sequence shows the commands typically used to mark a module generation as obsolete:

```
VSC> MODIFY MODULE/NOLOG [TAPE]DEVDRV.MAR -
_VSC>  /REMARK="Old tape device driver"
VSC> MODIFY GENERATION [TAPE]DEVDRV.MAR -
_VSC>  /NOLOG /OBSOLETE
```

5.6 Specifying Reviewer and Notification Lists

VSC allows you to specify that certain developers should review or be notified of all changes to certain streams or modules. To specify that a certain developer should review all changes replaced into a certain stream, use the /REVIEWER qualifier with the MODIFY STREAM command. For example:

```
VSC> MODIFY STREAM BIRCH /REVIEWER=(SMITH,SNURD)
```

This example causes VSC to record that developers SMITH and SNURD should review all code changes replaced into stream BIRCH. VSC appends these developers to the module's existing reviewer list. Thereafter, the REPLACE command sends mail notification to SMITH and SNURD each time a developer replaces modules into stream BIRCH, and SMITH and SNURD are automatically recorded as reviewers for each such replacement. They are thus able to accept or reject these replacements with the ACCEPT and REJECT REPLACEMENT commands. The user replacing modules into the stream can also specify additional reviewers with the /REVIEWER qualifier to the REPLACE command, but he or she cannot omit the automatic reviewers for the stream.

To remove reviewers from a stream's reviewer list, you should use the /NOREVIEWER qualifier to the MODIFY STREAM command. For example:

```
VSC> MODIFY STREAM BIRCH /NOREVIEWER=SNURD
```

This example removes developer SNURD from the reviewer list for stream BIRCH. If you specify the /NOREVIEWER qualifier without user name parameters, VSC removes all reviewers for the stream.

VSC also allows you to specify that certain developers should review all changes to a given module. The /REVIEWER qualifier to the MODIFY MODULE command attaches reviewers to modules. For example:

```
VSC> MODIFY MODULE [BAD]BADMAIN.B32 /REVIEWER=BROWN
```

This example causes VSC to make developer BROWN an automatic reviewer for all replacements of module BADMAIN.B32 in facility BAD, regardless of stream. The /NOREVIEWER qualifier removes reviewers for a module.

In addition to specifying reviewers for streams and modules, you can specify that certain developers be notified of all replacements to a certain stream or of a certain module. VSC automatically notifies such developers of replacements for the stream or module, but does not add them to the reviewer lists. They can therefore not accept or reject the replacements. This example shows how to add user BROWN to the list of users to notify of all changes to stream BIRCH:

```
VSC> MODIFY STREAM BIRCH /NOTIFY=BROWN
```

Each parameter to the /NOTIFY qualifier can specify the user name of a OpenVMS developer already recorded in the VSC database, an OpenVMS mail address containing one or more node names (such as NODE::SMITH), or a logical name bound to a list of users to notify. If you specify a logical name, keep in mind that the logical name must be defined both when you specify it and when it is later used to send the notification messages. The logical should thus be a group or system logical name, or a mail forwarding alias in the OpenVMS VMSMAIL.DATA database.

Similarly, the following example shows how to specify that users SMITH and SNURD be notified of all changes to module [BAD]BADMAIN.B32:

```
VSC> MODIFY MODULE [BAD]BADMAIN.B32 /NOTIFY=(SMITH,SNURD)
```

To remove users from a stream's reviewer list, you should use the /NONOTIFY qualifier to the MODIFY STREAM command. For example:

```
VSC> MODIFY STREAM BIRCH /NONOTIFY=BROWN
```

This example removes developer BROWN from the notification list for stream BIRCH. If you specify the /NONOTIFY qualifier without parameters, VSC deletes the entire notification list. This qualifier is used the same way with the MODIFY MODULE command.

The MODIFY STREAM command requires the VSC privilege CRESTRM; MODIFY MODULE requires CREMOD.

5.7 Creating New Streams

If you are the project leader for a stream, you must create new build streams from time to time so you can build OpenVMS for your release. You may also have to create the stream you are project-leading. This section tells you how to perform these operations.

Streams are classified into *build* streams, and into *release* streams. Build streams are used for builds and establish the files involved with each build, and are usually of interest only to those engineers directly involved with source code builds. Release streams are of general interest to all VDE or VSC users, as they are the streams used for managing source code development.

5.7.1 Creating Build Streams

To create a build stream for the development stream that you project lead, you must perform two operations: Create the build stream in the VSC database and create a CMS class for that build stream. To create the build stream in the VSC database, use the CREATE STREAM command. If you are the project leader for stream BIRCH, for example, you could use the following command to create a build stream:

```
VSC> CREATE STREAM BIRCH_BUILD -
_VSC>  /PARENT=BIRCH /CMS_CLASS -
_VSC>  /REMARK="Build stream"
```

This example creates build stream BIRCH_BUILD and specifies that its parent is stream BIRCH. This means that stream BIRCH_BUILD will be created with the same module generations as stream BIRCH currently has. Once you have created the new stream, the two streams can diverge; the next replacement into BIRCH, for example, will make it different from stream BIRCH_BUILD.

The CREATE STREAM command does not create a CMS class for the stream, but the /CMS_CLASS qualifier specifies that a CMS class for each facility should be maintained for the new stream as that stream is modified with future VSC commands (such as INSERT or DELETE GENERATION). How you create the CMS class initially is discussed shortly.

Specification of a remark is required; if you omit the /REMARK qualifier, VSC prompts you for a remark string.

Note: Specification of a remark for the new stream is required. Digital strongly recommends the remark indicate that the stream is a build stream.

For the OpenVMS VAX VDE library, the CREATE STREAM command takes about 20 minutes. You should run this command after you have performed all replacements that you want included in the build stream. It is best to run this command at less-busy times as it locks parts of the VSC database and can therefore prevent access by other users for various operations.

To create and populate a CMS class in each facility for the new stream, you must run the SYSBLDCOM$:POPULATE_LIBRARIES.COM command procedure. This procedure submits a separate batch job for each OpenVMS facility in order to populate the desired class as fast as possible. The whole operation is still time-consuming; it takes about four hours for the OpenVMS VAX library. To run this command procedure, enter the following command at DCL level:

```
$ @SYSBLDCOM$:POPULATE_LIBRARIES.COM BIRCH_BUILD BIRCH
```

This command procedure should be run after normal working hours. While it runs, you cannot modify stream BIRCH by replacing modules or inserting generations into it. You must ensure that no PERFORM REPLACEMENT commands run that can affect stream BIRCH, directly or through change propagation. However, you can (and preferrably should) run the CREATE STREAM command for the new build stream at the

same time. It is important that the VSC database is updated and the CMS classes are created at the same time so that they are consistent with each other.

If you already have an existing CMS class you want to reuse, you can use the SYSBLDCOM$:POP_REUSED_CLASS.COM command procedure instead. Its invocation is similar to that of POPULATE_LIBRARIES; the comments in the procedure give the details.

5.7.2 Creating Release Streams

To create a development stream for a new OpenVMS release (in other words, a stream that most OpenVMS developers will use to reserve and replace modules), you must make a number of decisions:

- decide what the parent stream will be. Preservation and continuity of the mainline is particularly important to modules on the mainline. Proper specification of stream parentage is of critical and central importance when the /FROM_CMS_LIBRARY qualifer is to be used on any subsequent CREATE FACILITY commands.
- decide if the stream should become the latest stream on the mainline, or if it should be treated as a variant branch and be located somewhere off the mainline. Exactly *one* stream in the VSC database *is* the mainline stream, while there can be many streams feeding into the mainline stream—these other streams are sometimes confusingly refered to as being ‘‘on the mainline’’.
- Create the stream in the VSC database, create the CMS class associated with it, and specify the stream’s change propagation behavior.
- Decide if you want propogation of modules from the parent stream. By default, such propogation is enabled.
- You may also want to specify developers who should review replacements or be notified of replacements going into the new stream.

To create the stream, use the CREATE STREAM command, much the same as explained in Section 5.7.1 for build streams. To create a new development stream MAPLE, for example, you might enter the following command:

```
VSC> CREATE STREAM MAPLE -
_VSC>  /OWNER=SMITH -
_VSC>  /PARENT=BIRCH /MAIN -
_VSC>  /CMS_CLASS /REPLACEMENT=QUEUE -
_VSC>  /REMARK="New-feature stream"
```

This command specifies that the parent of the new mainline stream MAPLE is the existing stream BIRCH. The newly created MAPLE mainline stream initially inherits all current generations from the parent BIRCH stream.

The use of the /OWNER qualifier specifies the owner of the new stream; this person is typically the stream’s project leader. If you use this qualifier, developers can later use the SHOW STREAM command to determine the stream’s owner.

The /MAIN qualifier causes the new stream to become the *last* stream on the main line of descent, and it causes VSC to use whole-number CMS generation expressions for the stream. /MAIN also causes VSC to update all successor stream links to link to this new mainline stream.

The /PARENT qualifier is informational in some cases, but can be critical during certain VSC operations such as CMS facility uploads. It should always be specified, and it should always specify the correct parent stream.

Note: The /[NO]MAIN and /PARENT qualifiers are central and necessary to the CMS generation number selection process used within VSC, and Digital recommands that these qualifiers be specified with care. In particular, the /MAIN qualifier can and should be used *only* on mainline streams, while /NOMAIN should be specified for non-mainline streams. Incorrect specification of /[NO]MAIN and/or /PARENT can and will cause longer CMS generation numbers, and has been shown to cause both confusion and consternation among developers accustomed to the typical (cardinal) progression of CMS generations that occurs for modules on the mainline.

Just as for build streams, the /CMS_CLASS qualifier specifies that a CMS class in each facility should be maintained for the stream.

The /REPLACEMENT qualifier specifies that the new stream requires all replacements into the stream to be queued (as opposed to immediate) replacements. Queued replacements go through the VSC review mechanism and give you control over when replacements are actually performed. The /REMARK qualifier specifies a remark string for the stream; if you omit this qualifier, VSC prompts you for a remark string.

Note: Specification of a remark for the new stream is required. Digital strongly recommends the remark indicate that the stream is a release or new-feature stream.

As explained in Section 5.7.1, the CREATE STREAM command does not create and populate the CMS class for the stream. Instead, you must submit the SYSBLDCOM$:POPULATE_LIBRARIES.COM or SYSBLDCOM$:POP_REUSED_CLASS.COM command procedure at DCL level to create and populate a CMS class for the stream in each facility. Section 5.7.1 describes how you use these procedures. One of these procedures should be run at the same time as you run the CREATE STREAM command, and you must ensure that no VSC commands that update the parent stream run at that time.

When you create a new development stream, you often want to specify that future replacements into some previously existing stream should be propagated to the new stream. For example, if stream MAPLE is created from stream BIRCH, you may want all future changes to BIRCH to be propagated to MAPLE so that MAPLE will contain all features and bug fixes that BIRCH contains. As a later stream, MAPLE will presumably also receive new features that are omitted from BIRCH. To specify this sort of stream successor relationship, you use the /SUCCESSOR qualifier to the MODIFY STREAM command. This is an example:

```
VSC> MODIFY STREAM BIRCH /SUCCESSOR=MAPLE
```

This command specifies that stream MAPLE is a successor to stream BIRCH. As a result, all code changes replaced into BIRCH will automatically be propagated to MAPLE if possible. If automatic change propagation is not possible for a given module replacement, VSC will create a fold record in its database.

You can specify multiple successors for a given stream if that is the behavior you want. To do so, specify a parenthesized list of stream names separated by commas with the /SUCCESSOR qualifier. Alternatively, specify each successor stream with a separate MODIFY STREAM command. To cancel successor relationships, use the /NOSUCCESSOR qualifier to the MODIFY STREAM command. (The HELP text has the details.)

To confirm that you have established the successor relationships correctly, use the /SUCCESSOR qualifier to the SHOW STREAM command. This example illustrates its output:

```
VSC> SHOW STREAM/SUCCESSOR BIRCH, MAPLE
Stream BIRCH
   Successor chains:
      BIRCH -> MAPLE

Stream MAPLE
   Successor chains:
      MAPLE (no successors)

VSC>
```

This output shows that stream MAPLE is a successor to stream BIRCH and that stream MAPLE does not itself have any successors. Because this is the desired successor relationship, VSC will perform change propagation correctly for these streams.

When you create a new release stream, you should also determine whether you want certain developers to review or be notified of all replacements into the stream. In particular, you probably want to be notified of all such replacements yourself. To specify the reviewers and developers to notify, use the /REVIEWER and /NOTIFY qualifiers to the CREATE STREAM or MODIFY STREAM command. Section 5.6 describes these qualifiers.

5.7.3 Example Stream Creations

The following code example shows how an OpenVMS development stream is created and populated. The example is typical of an OpenVMS *mainline* development stream.

The steps outlined below create a new *mainline* stream int the OpenVMS database, and a matching mainline stream in the OpenVMS test database. It temporarily disables propogation from the parent stream, as populating the stream can require some time. It then populates the new mainline streams, disables propogation into the new mainlines to prevent unintentional overwriting, and finally resets the proper propogation attribute on the parent stream.

These operations should be performed only when the VSC database is quiescent, as one typically wishes all modifications to the parent database to be propogated into the new stream.

```
$ VDE
Set Library VMS$
Create Stream -
    /CMS_CLASS/NOINIT_CMS_CLASS -
    /noGENERATION -
    /DEFER -
    /MAIN -  ! warning: alters generation selection
    /Populate -
    /Frozen -
    /Conference=YELLOW$NOTES:DRAGON-YELLOW.NOTE -
    /PARENT=CORAL -
    /NoSuccessor -
    /REPLACEMENT=QUEUE -
    /REMARK="Post-CORAL release stream"-
    DRAGON
Modify Stream DRAGON -
    /Reviewer=(projectleader) -
    /NOTIFY=(projectleader,DRAGONbld) -
    DRAGON
Modify Stream CORAL/NoSuccessor
Populate Stream DRAGON/From_Stream=CORAL
Modify Stream DRAGON/NoPopulate
Modify Stream CORAL/Successor=DRAGON
Exit
$!
$ VDE
Set Library VMSTEST$
Create Stream/CMS_CLASS/NOINIT_CMS_CLASS -
    /noGENERATION -
    /DEFER -
    /MAIN - ! warning: alters generation selection
    /Populate -
    /Frozen -
    /conference=YELLOW$NOTES:VMSTEST-YELLOW.NOTE -
    /PARENT=CORAL -
    /noSuccessor -
    /REPLACEMENT=QUEUE -
    /REMARK="Post-CORAL release stream"-
    DRAGON
$!
Modify Stream DRAGON -
    /reviewer=(projectleader,testleader) -
    /NOTIFY=(projectleader,testleader,DRAGONbld)
Modify Stream CORAL/NoSuccessor
Populate Stream DRAGON/From_Stream=CORAL
Modify Stream DRAGON/NoPopulate
Modify Stream CORAL/Successor=DRAGON
Exit
$ EXIT
```

The above example creates a mainline stream, which effects

5.8 Tailoring Processing With Scripts

As a project leader, you can tailor how VSC or VDE responds to certain situations using script files. Script files allow a number of common operations to be customized to site-specific requirements.

Table 5–5 contains a list of the available script types, and a brief description of each. Script creation uses the CREATE SCRIPT command, and requires the CRESCRP privilege.

Table 5–5 Script Commands

BUILD

The build script sets up optional build-job definitions. This script is executed at the start of each build-job process.

COMPILE

The compile script contains the command(s) used to compile a module. The compile script is called by a build job, and by a COMPILE command.

COPY

copy command copies a module. The script is called as part of a build job, and is called by the COPY command.

DELETE_GENERATION

The delete-generation script performs optional processing for the DELETE GENERATION command.

DIFFERENCES

The differences script is used to determine the differences between two generations of a module. The script is called by the DIFFERENCES command.

FETCH

The fetch script performs optional processing for the FETCH command.

INSERT_GENERATION

The insert-generation script performs optional processing for the INSERT GENERATION command.

LINK

The link script links a module. This script is called by build jobs and by the LINK command.

NEW_STREAM

The new-stream script copies files for a particular facility from the parent stream into a new stream. This script is called when a new stream is created.

NOKEEP

The nokeep script performs optional processing for the REPLACE command. Specifically, the nokeep script is called when the REPLACE/NOKEEP command is specified.

REPLACE

The replace script performs optional processing for the REPLACE command.

If stream-specific reference directories are required, this script can be used to update the reference directory on each replacement.

RESERVE

The reserve script performs optional processing when a RESERVE command is entered.

Should notification of a reservation be required, a site-specific reserve script can be coded to send mail.

Table 5–5 (Cont.) Script Commands

SET_FACILITY	The set-facility script performs optional processing for the SET FACILITY command.
SET_STREAM	The set-stream script performs optional processing for the SET STREAM command.
STAGING	The staging script copies a source file to a staging area for the REPLACE command when the replacement is queued.
UNRESERVE	The unreserve script performs optional processing for the UNRESERVE command.

6 Establishing a VDE Library

This chapter contains preliminary information; sections are currently under construction... And other sections are currently undergoing demolition...

This chapter describes how to set up a new VDE library by creating its database, setting up its disk directory structure, and populating it with users, development streams, facilities, and source modules. These commands must be performed before VDE can be used to manage the source code or or build the software system.

Commands and operations described in this section are normally performed only by the maintainers of the source code control libraries, by the database maintainance staff, or by the maintainers of VDE. These commands are also used by those users that wish to maintain private libraries using VDE.

Information pertaining to the tailoring and to the performance of VDE database backup operations is included in Section 7.4. Once you have created the library, you should establish the appropriate backup procedures, and determine the appropriate backup scheduling.

Note: The VDE utility and the VDE> are shown in this chapter. Unlike VSC, the VDE utility is intended to be used with VDE databases other than the OpenVMS source code control libraries.

6.1 Creating a VDE Library

A VDE library consists of various components: a root directory, a VDE database, and a disk directory tree. The *root directory* is the disk directory that contains the VDE database and that normally serves as the root directory for the remaining disk directories in the library. The *VDE database* is a Oracle Rdb7 database which stores all information that VDE needs about your software system. Rdb is a relational database management system, and VDE reads and writes the VDE database using SQL, and industry-standard query language for relational databases. The library's *directory tree* consists of all disk directories that hold the CMS libraries, source files, build output files, and other files that comprise your software system. The VDE database records the locations of all directories in the directory tree.

To create a VDE library, you must first create the root directory and the VDE database. Thereafter, VDE can create the rest of the directory tree for you, depending on what facilities and development streams you need for your software system. VDE can create the directory tree using its own default rules, but you can also explicitly specify how you want the directory tree created.

Note: Once you have successfully created a VDE library, you should tailor or create the procedures necessary to perform periodic database backup operations. See Section 7.4 for details.

6.1.1 The VDE$UPLOAD CMS Upload Utility

VDE includes a utility, VDE$UPLOAD, that can perform the entire creation of a VDE database, based on the specification of a directory containing one or more CMS libraries.

The contents of all of the CMS libraries under the target directory are uploaded into VDE, including converting the specified CMS classes into VDE streams.

For information on VDE$UPLOAD, see the VDE Reference Manual.

As an alternative to the VDE$UPLOAD procedure, the VDE database can be created via a series of manual commands. An example is included in the kit, the VDE$CREATE procedure. For further information on this, see Section 6.7.3.

6.1.2 The CREATE LIBRARY Command

To create the root directory and the VDE database, use the CREATE LIBRARY command. This command creates the root directory you specify, creates and initializes a VDE database in that directory, and sets up an initial facility and an initial stream in the library. In short, it creates the minimal usable VDE library. The following example illustrates the command syntax:

```
VDE> CREATE LIBRARY DEV$:[ROOTDIR] /NAME=LIBNAME /CREATOR="John Doe" -
_VDE>                              /REMARK="Library for gadget development"
```

This example creates a VDE library with root directory DEV$:[ROOTDIR]. After creating the root directory, VDE creates the library's VDE database in that directory and records a library name, the library creator, a remark string, and certain other attributes in the database. (The library name is LIBNAME and the library creator is John Doe in this example.) You should expect library creation to take at least two to ten minutes of elapsed time, depending on your system configuration and load. The bulk of this time is required to create and initialize the Rdb database.

It is strongly recommended that you use a system-wide concealed logical name for the disk name so that you can move the VDE library to another disk at a later time by simply moving the library's files and directories and redefining the logical name. The VDE database records the device names and full directory specifications of numerous subdirectories within the root directory. This makes it difficult to move the VDE library to another disk unless a concealed logical name is used to denote the disk. It should be a system-wide name so it is accessible to all users of the VDE library. A group-wide concealed logical name is acceptable if all users, present and future, will belong to the same UIC group.

The CREATE LIBRARY command establishes a default facility called CODE in your library. For a simple software system, you can use this facility for your source code and then set up other facilities for tools, specifications, and documentation as needed. For a more complex software system, you can delete the CODE facility and create other facilities that better suit your needs. In addition, the CREATE LIBRARY command creates a default development stream called MAIN. You can use this stream for your main development thread unless your software system requires a more complex stream structure. Stream MAIN is the ancestor of all other streams you create as the library evolves.

When you enter the CREATE LIBRARY command, you are required to specify the desired library root directory, the library name, the name of the library creator (your own first and last names), and a remark string. The root directory name is the command parameter while the library name, creator name, and remark string are specified with qualifiers. VDE stores the library name and remark string in the ''database header record'' in the Rdb database. Because you are the library creator, VDE records your full name (as specified with the /CREATOR qualifier) and your OpenVMS user name in a User Table entry in the database. A field in the database header record points to your User Table entry to indicate that you created the library.

If you omit any required item, VDE prompts you for its value. The following example illustrates how VDE prompts for information if all required items are omitted:

```
VDE> CREATE LIBRARY
Enter directory specification for the new library: DEV$:[ROOTDIR]
Enter the name you want for the new library: LIBNAME
Enter your full name (first and last names): John Doe
Enter the remark for new library: Library for gadget development
```

This example creates exactly the same library as the previous example.

By default, the CREATE LIBRARY command creates the initial directory tree for your library. VDE creates a *delta-file root directory* called VDE$CMS.DIR within the library root directory. The delta-file root directory will contain one subdirectory for each facility in your software system to hold the facility's CMS library. Each CMS library will hold CMS elements, also called ''delta files,'' to contain all generations of each source module in a condensed form. VDE accesses these CMS libraries using callable CMS.

VDE creates one *stream root directory* and associated directory tree for each development stream in your system. Because the CREATE LIBRARY command always creates stream MAIN, it also creates the root directory and directory tree for that stream by default. The stream MAIN root directory is named MAIN.DIR and is a subdirectory of the library root directory.

The CREATE LIBRARY command also creates the *staging area directories* for the library. The top-level of this directory tree is called VDE$STAGE.DIR, and is a subdirectory of the library root directory. VDE uses subdirectories of the staging area directories to store, or ''stage,'' source files for queued replacements. Queued Replacements describes queued replacements and how they use staging area directories.

By default, the CREATE LIBRARY command creates the directories just described for the new library. It also creates the CMS library for facility CODE. However, you can defer directory creation by specifying the /DEFER qualifier with the CREATE LIBRARY command. /DEFER causes VDE to not create any directories other than the library root directory and to not create the CMS library for facility CODE. You can then use the SET DIRECTORY command to specify where you want these deferred directories to be created (see Section 6.3). After entering all desired SET DIRECTORY commands, you can enter the CREATE DIRECTORY_TREE command to actually create the directory tree (or various components thereof) for your library. Using this method, you can specify that some directories be placed on other disks, for example, or that each facility get additional subdirectories. Section 6.3 describes how to use these capabilities.

The CREATE LIBRARY command also specifies a number of other library attributes. These attributes include the default and authorized VDE privileges that new users are assigned, information on how new users are added to the library, whether the RESERVE command displays the expected CMS generation expression for the corresponding replacement, whether the REPLACE command requests information files by default, and various other attributes. Section 6.2 gives more detail on these attributes. If you do not specify these optional attributes with the CREATE LIBRARY command, VDE provides default settings. You can modify all such settings later with the MODIFY LIBRARY command. You can also use the MODIFY LIBRARY command to change the library name and remark string. The only attribute you cannot change with MODIFY LIBRARY is the library root directory.

6.1.3 Examples

This is an example of a CREATE LIBRARY command and its output:

```
VDE> CREATE LIBRARY DEV$:[ROOTDIR] /NAME=LIBNAME /CREATOR="John Doe" -
_VDE>                              /REMARK="Library remark"
Library LIBNAME to be created in root directory DEV$:[ROOTDIR]
   Library root directory DEV$:[ROOTDIR] already exists.
   Logical name VDE$LIBRARY now points to the new library.
   Creating library database; please wait...
   Library database has been created.

Staging area root directory DEV$:[ROOTDIR.VDE$STAGE] has been created

Creating directory tree for delta files:
   Root directory DEV$:[ROOTDIR.VDE$CMS] has been created
   Facility CODE root directory DEV$:[ROOTDIR.VDE$CMS.CODE] has been
            created
      CMS library for facility CODE has been created
   All directories for delta files now exist

Creating directory tree for stream MAIN:
   Root directory DEV$:[ROOTDIR.MAIN] has been created
      Subdirectory DEV$:[ROOTDIR.MAIN.VDE$BLD] has been created
   Facility CODE root directory DEV$:[ROOTDIR.MAIN.CODE] has been
            created
      Subdirectory DEV$:[ROOTDIR.MAIN.CODE.OBJ] has been created
      Subdirectory DEV$:[ROOTDIR.MAIN.CODE.SRC] has been created
      Subdirectory DEV$:[ROOTDIR.MAIN.CODE.VDE$COM] has been created
      Subdirectory DEV$:[ROOTDIR.MAIN.CODE.VDE$LOG] has been created
```

```
   All directories for stream MAIN now exist

Library creation completed.

VDE>
```

In this example, VDE creates a new library in root directory DEV$:[ROOTDIR]. The library has name LIBNAME and the library creator is named John Doe. The VDE output shows that VDE has created the library root directory (which turned out to already exist in this case), the VDE database, the VDE staging area directories, the delta-file directory tree including the CMS library for facility CODE, and the directory tree for stream MAIN. The new VDE library thus has the initial facility CODE and the initial stream MAIN fully set up. There are no modules in the library, however, and only one allowed user (the creator John Doe) is recorded in the library database.

In the following example, the library is created the same way except that the library creator (Jane Smith) specifies the /DEFER qualifier to defer the creation of the library directory tree and the CMS library for facility CODE. In addition, all required parameters and qualifiers are omitted from the command line. As a result, VDE prompts for all required information:

```
VDE> CREATE LIBRARY /DEFER
Enter directory specification for the new library: DEV$:[ROOTDIR]
Enter the name you want for the new library: LIBNAME
Enter your full name (first and last names): Jane Smith
Enter the remark for new library: Library remark string
Library LIBNAME to be created in root directory DEV$:[ROOTDIR]
   Library root directory DEV$:[ROOTDIR] has been created.
   Logical name VDE$LIBRARY now points to the new library.
   Creating library database; please wait...
   Library database has been created.

Library creation completed.

VDE>
```

As the output shows, VDE creates the library root directory and the library database, but does not create the rest of the directory tree. The user thus has the opportunity to modify the directory structure and its exact locations with SET DIRECTORY commands before using CREATE DIRECTORY_TREE commands to create the directory tree.

The following example shows the VDE library creation, as well as a number of other ancillary steps involved.

```
$!
$! Locate and delete previous incarnations of the test facility files,
$! then unpack the saveset containing the test files...
$!
$ if f$search("sys$scratch:A.REQ") .nes. "" Then Delete sys$scratch:A.REQ;*/log
$ if f$search("sys$scratch:B.REQ") .nes. "" Then Delete sys$scratch:B.REQ;*/log
$ if f$search("sys$scratch:C.B32") .nes. "" Then Delete sys$scratch:C.B32;*/log
$ if f$search("sys$scratch:D.B32") .nes. "" Then Delete sys$scratch:D.B32;*/log
$ if f$search("sys$scratch:E.B32") .nes. "" Then Delete sys$scratch:E.B32;*/log
$ if f$search("sys$scratch:F.B32") .nes. "" Then Delete sys$scratch:F.B32;*/log
$!
$ Backup VDE$CREATE.BCK/SAVE Sys$Scratch:
$!
$! Create the core directory -- this must be present...
$!
$ Create/Directory STAR$DATA:[QTV.WWW.VDE$DATABASE]
$ VDE
!
! create the WWW library in STAR$DATA:[QTV.WWW...], using the
! mnemonic WWW, placing all Rdb-related library components in
! STAR$DATA:[QTV.WWW.VDE$DATABASE].
!
create librar  STAR$DATA:[QTV.WWW.VDE$DATABASE] /DEFER -
 /NAME=WWW -
 /REMARK="QTV WWW Source Library" -
 /NOAUTO_CONVERT /AUTO_ADD_USER=VMS_SOURCE /AUTO_MAIL -
 /ASK_INFO_FILE /ASK_REPL_CAUSE /SHOW_GEN_EXPR /NOSTATISTI -
 /NOHISTORY_NOTES /NOALLOW_DELETE /NOWILD_INS_GEN -
 /NOCMS_ELEM_HIST /CREATOR="Stephen Hoffman" -
 /DEF_PRIVILEGES=(CREMOD,RESREP,CREFAC) -
 /AUTH_PRIVILEGES=(CREMOD,RESREP,CREFAC) -
 /data=STAR$DATA:[QTV.WWW.VDE$DATABASE] -
 /journ=STAR$DATA:[QTV.WWW.VDE$DATABASE] -
 /snap=STAR$DATA:[QTV.WWW.VDE$DATABASE]
!
SHOW DEFAULTS
SHOW LIBRARY/FULL
MODIFY CONTEXT/PUBLIC DEFAULT/NOSTREAM/NOFACILITY
!
CREATE USER SYSTEM /NAME="System Manager"/DEF_P=ALL/AUTH_P=ALL
System Manager
!
! Create the CMS ("delta") library directory structure.
! Configure the CMS libraries to use the directory structure:
!   STAR$DATA:[QTV.WWW.CMS$DATABASE.'facility'.CMS].
! Configure the "marker" files to use the directory structure:
!   STAR$DATA:[QTV.WWW.CMS$DATABASE.'facility'.SRC]...
!
SET DIRECTORY/DELTA_FILES STAR$DATA:[QTV.WWW.CMS$DATABASE]
SET DIRECTORY/DELTA_FILES/SUBDIRECTORY=VDE$CMS [.CMS]
SET DIRECTORY/DELTA_FILES/SUBDIRECTORY=VDE$MARKER [.SRC]
CREATE DIRECTORY_TREE/DELTA_FILES
SHOW DIRECTORY/DELTA_FILES
!
! Create the staging area...
!
SET DIRECTORY/STAGING STAR$DATA:[QTV.WWW.VDE$STAGING]
CREATE DIRECTORY_TREE/STAGING
!
! Create the stream/build directories
!
SET DIRECTORY/STAGING STAR$DATA:[QTV.WWW.VDE$STREAM]
CREATE DIRECTORY_TREE/STREAM
```

```
!
! create and load the VDE test facility.
!
CREATE FACILITY AAA_VDE_TEST -
    /REMARK="Test facility for VDE developers"
SET FACILITY AAA_VDE_TEST"
CREATE MODULE/NODELETE/REMARK="VDE self-test module" -
  A.REQ /INPUT=SYS$SCRATCH:
CREATE MODULE/NODELETE/REMARK="VDE self-test module" -
  B.REQ /INPUT=SYS$SCRATCH:
CREATE MODULE/NODELETE/REMARK="VDE self-test module" -
  C.B32 /INPUT=SYS$SCRATCH:
CREATE MODULE/NODELETE/REMARK="VDE self-test module" -
  D.B32 /INPUT=SYS$SCRATCH:
CREATE MODULE/NODELETE/REMARK="VDE self-test module" -
  E.B32 /INPUT=SYS$SCRATCH:
CREATE MODULE/NODELETE/REMARK="VDE self-test module" -
  F.B32 /INPUT=SYS$SCRATCH:
!
EXIT
$
$ Exit
```

6.2 Optional Library Attributes and Settings

There are a number of library attributes that you can optionally set with the CREATE LIBRARY command. These attributes all have default values (which is why their specification is optional) and most of them can be altered later with the MODIFY LIBRARY command. However, before you specify the CREATE LIBRARY command, you should consider whether you need to set these optional attributes.

Note: When dealing with critical libraries, it is strongly recommended that the staging area directories, the CMS libraries (the delta files) and the VDE database each be kept on a unique disk spindle or unique disk shadowset. Further, Rdb should be run with multiple files, and these component files should also be spread across unique disk spindles or shadowsets.

6.2.1 Database Placement

One attribute you must consider before creating the VDE library is the placement of the VDE database. The VDE database consists of three files: a database root file, a storage area file, and a snapshot file file. (These three files use the file extention types RDB, RDA, and SNP, respectively.)

The database root file (VDE$DATABASE.RDB) must reside in the VDE library root directory; the library root directory is by definition the directory that contains the database root file. This file is small, however; it only contains information that Rdb needs to manage the database.

The database storage area file (VDE$DATABASE.RDA) contains the actual data that VDE keeps about your software system. This file can be quite large, its actual size depending on the size of your software system.

The snapshot file (VDE$DATABASE.SNP) retains the information necessary to support concurrent access to the database. This file can also be relatively large.

By default, VDE places all three files in your VDE library's root directory. However, if your software system is sufficiently large or sufficiently active, you should place the storage area (RDA) and snapshot (SNP) files on seperate disk spindles or disk seperate shadow sets.

To relocate the storage and snapshot areas, specify the desired directory for each on the CREATE LIBRARY /DATABASE and /SNAPSHOT qualifiers. If /SNAPSHOT is *not* specified, the snapshot file will be coresident in the /DATABASE database directory. If neither qualifier is specified, all three database files will be coresident.

The following example illustrates how you can use these qualifiers to control the placement of the various database files:

```
VDE> CREATE LIBRARY DEV1$:[ROOTDIR] /DATABASE=DEV2$:[PROJDIR.DB] -
_VDE>                               /SNAPSHOT=DEV1$:[SNAPDIR]
```

This command causes VDE to place the database root file (VDE$DATABASE.RDB) in the library root directory on disk DEV1$, to place the database data file (VDE$DATABASE.RDA) in directory [PROJDIR.DB] on disk DEV2$, and to place the database snapshot file in directory [SNAPDIR] on disk DEV1$. Spreading the database files over two disks, as done here, can improve the performance of the database.

The Rdb database may also use a fourth file, an after-image journal file (with file type AIJ). When an Rdb database has after-image journalling enabled, all changes to the database are recorded in that journal file. If the database is later lost or damaged, perhaps due to hardware problems or disk corruption, you can restore the database from a backup file and then use the journal file to recover all updates since the last backup. Journalling is optional but strongly recommended. To enable journalling, you must specify the location of the journal file using the /JOURNAL qualifier to the CREATE LIBRARY or MODIFY LIBRARY command. The journal file should always be placed on a different physical disk from the other database files. Section 7.4 describes how to back up and recover your VDE database and gives more information on journal files.

6.2.2 Standard User Privileges

When you create your VDE library, you can specify the default and authorized VDE privileges you want assigned to new users as they are added to your VDE library. VDE privileges determine whether a user is allowed to reserve and replace modules or not, whether a user is allowed to perform queued replacements or not, whether a user is allowed to create modules, streams, and facilities or not, and so on. The description of the SET PRIVILEGES command contains the full list of all VDE privileges.

Each user has a set of default privileges and a set of authorized privileges. The default privileges are those privileges that are enabled by default each time the user starts a new VDE session. The authorized privileges are those privileges that the user is allowed to enable with the SET PRIVILEGES command. The default privileges are always a subset of (or the same as) the authorized privileges.

The VDE database stores a ‘‘standard’’ set of default and authorized privileges that are assigned to new users as they are added to the database. You can override these standard privileges for any given user by explicitly specifying different privileges with the CREATE USER or MODIFY USER command. However, use of the standard privileges relieves you of the necessity of specifying VDE privileges for each user individually. This is especially important if you allow users to be added to the database automatically (see Section 6.2.3).

The /DEF_PRIVILEGES qualifier to the CREATE LIBRARY command allows you to specify the default privileges you want new users to have. The /AUTH_PRIVILEGES qualifier allows you to specify the authorized privileges you want new users to have. You can later use these qualifiers with the MODIFY LIBRARY command to change the library's standard privileges. If you do not specify standard privileges for your VDE library, VDE gives new users the CREMOD and RESREP privileges by default.

The following example shows how you can specify the standard default and authorized privileges for a VDE library being created:

```
VDE> CREATE LIBRARY /DEF_PRIVILEGES=(CREMOD,RESREP) -
_VDE>               /AUTH_PRIVILEGES=(CREMOD,RESREP,CREGRP)
```

The example omits other qualifiers and parameters for clarity.

6.2.3 Automatic Addition of Users to Database

Before a user can use the new VDE library you are creating, that user must be added to the library's VDE database. This means that VDE must create an entry for the user in the database User Table. This entry records the user's OpenVMS username, full name (first and last names), default and authorized privileges, and other information. A user who has not been added to the database cannot use VDE to access information in that database or to perform VDE operations on the VDE library.

The CREATE LIBRARY command automatically adds the creator of a new VDE library to the library's database. Additional users must be added in one or two ways. First, they can be added manually using the CREATE USER command. This command allows you to specify the attributes of each new user explicitly. This way of adding users is most practical for VDE libraries with a small number of users, especially if you want careful control over who accesses the library.

Second, you can set a library attribute that allows new users to be added automatically the first time they access the VDE library. This way of adding users is more practical for VDE libraries with a large number of potential users. To enable automatic addition of new users, you must use the /AUTO_ADD_USER qualifier to the CREATE LIBRARY or MODIFY LIBRARY command. If this qualifier is used without a parameter, any user who attempts to access the VDE library through VDE is automatically added to the VDE database with the standard default and authorized VDE privileges for that library.

The qualifier can also be specified with a rights identifier parameter, as in /AUTO_ADD_USER=*rights-id*. In this case, VDE will only add those users to the database who hold the OpenVMS rights identifier given by the parameter. This way of adding users is useful when you already use a OpenVMS rights identifier to protect the files and directories of your VDE library. In this case, the OpenVMS security mechanism for the library also serves as the mechanism that restricts access to the VDE library through the VDE utility.

Section 6.4 gives a more complete description of how you can add users to your VDE library.

If you do not specify the /AUTO_ADD_USER qualifier, you must add users with explicit CREATE USER commands. However, you can change the library attribute that controls automatic addition of users at any time by using the /AUTO_ADD_USER or /NOAUTO_ADD_USER qualifier with the MODIFY LIBRARY command.

The owner information specified in the OpenVMS SYSUAF database will be used as the owner of any new VDE username created. VDE will ignore any leading numerics, leading hyphens, leading space characters, and leading tilde characters, listed in the SYSUAF owner field.

6.2.4 Conversion of Library Format

VDE maintains data about your VDE library in a relational database. As new capabilities are added to VDE, the format of this database may change from one version of VDE to the next. A new feature may, for example, require the addition of new database tables or of new columns to existing tables in order to store the data required to implement the feature. New features may also require other changes to your VDE library, such as the addition of a subdirectory or a new file. To accommodate such changes to the library format, each new version of VDE that requires a format change contains code that converts your existing VDE library to the new format. VDE also maintains a database format version number in your database so that it always knows the current format of your library and its database.

When a new version of VDE is installed on your system, that version may require that your existing VDE library be convered to a new format. If so, there are two ways to effect this conversion. One is to allow VDE to automatically convert the library format the first time any user accesses the library through VDE. The second way is to explicitly convert the library using the CONVERT LIBRARY command.

By default, VDE will *not* convert your VDE library automatically the first time any user starts a new VDE session that attaches to that library[1] Automatic conversion is useful for small libraries, for libraries that are accessed only by the local copy of VDE, and for libraries that are easily restored from backup. In order to avoid unexpected conversions and the

[1] Automatic conversion was the default for all libraries created prior to VDE V1.3. If you suspect your library was created on a version of VDE earlier than V1.3, use the MODIFY LIBRARY/NOAUTO_CONVERT command to disable automatic conversion.

resulting potential for library access outages, automatic conversion is, by default, disabled when the library is created.

If you specify the /AUTO_CONVERT qualifier with the CREATE LIBRARY or MODIFY LIBRARY command, VDE will automatically convert your library when new versions of VDE are installed on your system. If automatic conversion is disabled, a user must enter an explicit CONVERT LIBRARY command to convert the library. To use this command, you must have the VDE privilege MODLIB. This approach gives you more control over when conversions occur so that your VDE library cannot be accidentally converted to the format for a new version of VDE that you do not yet want to use.

Furthermore, when the library format changes, you should in general delete all processes connected to the VDE library before converting the library so that all user processes use the new version of VDE after the conversion. This is especially important if most users use kept VDE subprocesses to access the library. To delete such processes, use the RMU/CLOSE command with the /ABORT=DELPRC qualifier. For example:

```
$ RMU/CLOSE/CLUSTER/ABORT=DELPRC DEV$:[ROOTDIR]VDE$DATABASE
```

After entering this command, use the VDE CONVERT LIBRARY command. This procedure ensures that all users of your library will access it with the new version of VDE and that your library has the format that this version requires.

To use the RMU/CLOSE command, you must have the OpenVMS privilege SYSPRV.

You can modify the library attribute that controls automatic library conversion using the /AUTO_CONVERT or /NOAUTO_CONVERT qualifier with the MODIFY LIBRARY command.

6.2.5 Sending Mail Notifications

The /AUTO_MAIL qualifier specifies that VDE should send a mail notification message to the user who has queued a replacement when that replacement is later performed (see Queued Replacements). The /NOAUTO_MAIL qualifier suppresses such notification messages. These qualifiers thus control the volume of mail messages VDE sends. If you specify neither qualifier, /AUTO_MAIL is assumed by default.

The /NOSEND_MAIL qualifier suppresses all VDE mail notification messages for the library. This qualifier is useful if you are setting up a VDE library for testing purposes or simply to play with. For a production VDE library you should always enable mail notifications, which can you do by specifying the /SEND_MAIL qualifier or by simply omitting the /NOSEND_MAIL qualifier.

6.2.6 Allowing Deletion of Library Entities

VDE allows you to delete library entities such as modules, facilities, and streams with various DELETE commands. To use such commands, you must in general have the corresponding delete privileges. These delete privileges protect against deletion of library entities by unauthorized users. They also require authorized users to set those privileges before deleting entities, thus providing some protection against accidental deletion.

As an additional protection against accidental deletion of library entities, VDE allows you to specify that certain entities, such as specific modules, facilities, and streams, are not allowed to be deleted even when the user has the corresponding delete privilege. A given stream, for example, can have its delete attribute set to no-delete using the /NODELETE qualifier to the MODIFY STREAM command. VDE will then not allow the DELETE STREAM command to delete that stream. The stream can still be deleted, but only if the attribute is first changed with another MODIFY STREAM command to allow the stream to be deleted. The no-delete attribute thus prevents accidental deletion of the stream by requiring an extra command to explicitly allow deletion before the DELETE STREAM command can be used. The delete-stream privilege is required as well.

When you create your VDE library, you can specify the default setting of the deletion attribute for new database entities. The /ALLOW_DELETE qualifier to CREATE LIBRARY specifies that new database entities should be allowed to be deleted by default. The /NOALLOW_DELETE qualifier specifies that database entities should not be allowed to be deleted by default. If you specify neither qualifier, VDE by default marks new database entities as being allowed to be deleted. You can of course always override this default by explicity specifying the delete attribute when creating new entities. In general, allowing deletion is more convenient for the library maintainer. Not allowing deletion is more appropriate when numerous individuals have access to the library and some of them may be accident-prone when entering commands.

6.2.7 CMS History and Notes Defaults

Two sets of qualifiers to the CREATE LIBRARY command controls VDE's treatment of CMS history and notes information in the VDE library. The /NOCMS_ELEM_HIST qualifier causes VDE to create CMS elements (that is, delta files) with no default history and notes strings. This prevents CMS from producing files with history or notes information when modules are fetched with the CMS FETCH command. This library attribute has no effect on modules fetched with the VDE FETCH command, so it is only relevant if DCL command procedures access the CMS elements in the VDE library using CMS directly.

The /NOHISTORY_NOTES qualifier causes the VDE RESERVE command to never produce CMS history or notes information in its output files and to not accept the /HISTORY, /NOTES, and /POSITION qualifiers. Disabling history and notes information in reserved files makes certain

mistakes less likely where such information gets replaced into the VDE library as part of the module text.

Both of these qualifiers prevent certain problems where history or notes information gets included in output files where this information must not be present. Such problems only occur in special circumstances and do not affect most VDE libraries. As a result, these qualifiers are optional and the default settings are given by the positive forms of those qualifiers, /CMS_ELEM_HIST and /HISTORY_NOTES.

6.2.8 Configuring Architectures

This section is under construction...

After the creation of the various architectures using one or more CREATE ARCHITECTURE commands, one normally enables the display of differnt architectures with the MODIFY LIBRARY/SHOW_ARCHITECTURE command.

See Chapter 9 for further information.

6.2.9 Miscellaneous Attributes

Other qualifiers to the CREATE LIBRARY command allow you to set various miscellaneous library attributes. The /ASK_INFO_FILE qualifier determines whether the REPLACE command will by default ask the user whether to edit a replacement information file (see Section 2.7). The /SHOW_GEN_EXPR qualifier determines whether the RESERVE command displays the expected generation expression of the new module generation to be created when the module is later replaced. These and other qualifiers are described in more detail in the description of the CREATE LIBRARY command. Whether they are applicable to you depends on the development process you think is appropriate for your project.

If you have no strong feelings on these options, use the defaults. If you later change your mind, you can always change these setting with the MODIFY LIBRARY command.

6.3 Library Directory Structure

In addition to its root directory and database, your VDE library contains a number of additional disk directories and subdirectories. These additional directories contain the source modules for your software system, all associated derived modules (such as object modules and executable images), and various command files and log files that VDE uses to perform its functions. This section describes the structure and functions of these disk directories.

6.3.1 Delta File Directories

VDE always stores all source modules that are part of your software system in ‘‘delta files.’’ A *delta file* stores all generations of a source module in a condensed form. The delta file stores every line that has ever been part of the source module, along with information specifying which lines belong to which generations of the module. Because a line that is the same from one generation to the next is only stored once in the delta file, the file saves space by essentially only storing differences between generations. (Such files are called ‘‘delta files’’ because the Greek letter delta is often used to denote differences between quantities in mathematical notation.)

VDE uses callable CMS (VAX/VMS Code Management System) to manage its delta files. VDE sets up one CMS library for each facility in your software system. VDE also creates one CMS element (which is a delta file) for each source module in your software system.

By default, VDE creates a *delta-file root directory* called VDE$CMS.DIR within the library root directory when you create the library. Each time you create a new facility, VDE creates a subdirectory for that facility within the delta-file root directory. The CMS library for the facility is created in the facility’s subdirectory. CMS uses additional subdirectories within the CMS library to store the actual delta files (CMS elements), but VDE does not keep track of the internal organization of the CMS library. Table 6–1 illustrates the disk directories that VDE establishes for delta files.

Table 6–1 Delta File Disk Directories

Library root directory:	DEV$:[ROOTDIR]
Delta-file root directory:	DEV$:[ROOTDIR.VDE$CMS]
Directory for facility FOO:	DEV$:[ROOTDIR.VDE$CMS.FOO]
Directory for facility BAR:	DEV$:[ROOTDIR.VDE$CMS.BAR]

VDE records the directory specification for the delta-file root directory and each facility’s delta-file directory in its database. The SHOW LOCATION /DELTA_FILE command displays this information on your terminal.

Although VDE uses the default locations just described, you can specify different locations for the delta-file directories. To specify a different delta-file root directory (VDE$CMS by default), you must take the following actions:

- Specify the /DEFER qualifier with the CREATE LIBRARY command. This qualifier prevents VDE from initializing the delta-file directory tree using its default rules.
- Enter the SET DIRECTORY command with the /DELTA_FILE qualifier to specify the desired delta-file root directory. For example, to specify that the delta-file root directory should be DEV$:[DELTAROOT], use the following command:

```
VDE> SET DIRECTORY /DELTA_FILES DEV$:[DELTAROOT]
```

This command causes VDE to use directory DEV$:[DELTAROOT] as the delta-file root directory instead of DEV$:[ROOTDIR.VDE$CMS].

- Enter the SET DIRECTORY command with the /DELTA_FILE and /FACILITY qualifiers to specify the desired delta-file directory for a specific facility. For example, to specify that the delta-file directory for facility FOO should be DSK2:[PQR.STU], use the following command:

  ```
  VDE> SET DIRECTORY /DELTA_FILES /FACILITY=FOO DSK2:[PQR.STU]
  ```

  Note that you can use this command to place the delta-file directory for a given facility on a different disk than the rest of the VDE library.

- After having entered all desired SET DIRECTORY commands, enter the following command to actually create the specified delta-file root directory and the rest of the delta-file directory tree:

  ```
  VDE> CREATE DIRECTORY_TREE /DELTA_FILES
  ```

  This command creates the CMS library for each facility in addition to creating the actual directories.

  To create the delta-file directory and CMS library for a specific facility FOO, use the following command:

  ```
  VDE> CREATE DIRECTORY_TREE /DELTA_FILES /FACILITY=FOO
  ```

Note that the SET DIRECTORY command specifies where a directory should be located, but does not actually create the directory. In effect, it creates a template for some part of the VDE directory tree. To actually create directories from that template, you must enter the CREATE DIRECTORY_TREE command. This arrangement allows you to specify templates for many parts of the directory tree before you actually use the templates to create directories.

6.3.2 Staging Area Directories

When you use VDE for source control, you can replace modules in two ways. One is to replace modules immediately, in which case the modules being replaced are inserted into the VDE library right away. The other way is to queue modules for replacement, in which case the modules are not inserted into the library until an authorized user ‘‘performs’’ the queued replacement. Queued replacements are useful because they give project managers more control over the timing of replacements and they allow the replaced modules to be reviewed before being inserted into the library.

When a user queues modules for replacement, VDE copies the files being replaced into the library’s staging area. The *staging area* is a disk directory tree that temporarily holds files being replaced until the corresponding replacements are performed by an authorized user. The staging area starts with the staging area root directory. By default, this root directory is subdirectory VDE$STAGE within the library root directory.

Within the staging area root directory, VDE creates two more levels of subdirectories. The staging area root directory contains subdirectories with names of the form VDE$STG_*n*. Each such directory holds a group of 100 replacements. Directory VDE$STG_12, for example, contains replacements with Replacement Id values in the range 1200 to 1299. Within each such directory, VDE creates second-level directories that each hold the files for a single replacement. Within VDE$STG_12, for example, there are subdirectories with names such as VDE$REP_1261, representing replacement 1261. Table 6–2 illustrates this structure.

Within the replacement subdirectory, VDE creates one subdirectory for each facility involved in the replacement. VDE stores the individual files being replaced in those facility subdirectories. Comment and information files for the replacement are stored in the replacement subdirectory. This is also shown in Table 6–2.

Table 6–2 Staging Area Disk Directories

Library root directory:	
	DEV$:[ROOTDIR]
Staging area root directory:	
	DEV$:[ROOTDIR.VDE$STAGE]
Directory for 100 replacements:	
	DEV$:[ROOTDIR.VDE$STAGE.VDE$STG_12]
Directory for one replacement:	
	DEV$:[ROOTDIR.VDE$STAGE.VDE$STG_12.VDE$REP_1261]
Comment file for replacement ID 1261:	
	DEV$:[ROOTDIR.VDE$STAGE.VDE$STG_12.VDE$REP_1261]DOE.VDE$COMMENT
Directory for facility FACNAM:	
	DEV$:[ROOTDIR.VDE$STAGE.VDE$STG_12.VDE$REP_1261.FACNAM]
File for module being replaced:	
	DEV$:[ROOTDIR.VDE$STAGE.VDE$STG_12.VDE$REP_1261.FACNAM]MOD.TYP

As mentioned above, VDE's default location for the staging area root directory is subdirectory VDE$STAGE within the library root directory. To change this default, you must take the following actions:

- Specify the /DEFER qualifier with the CREATE LIBRARY command. This qualifier prevents VDE from initializing the staging area root directory using its default rules.
- Enter the SET DIRECTORY command with the /STAGING qualifier to specify the desired staging area directory. For example, to specify that the staging area directory should be DEV2$:[PROJ.STAGE], use the following command:

  ```
  VDE> SET DIRECTORY /STAGING DEV2$:[PROJ.STAGE]
  ```

  This command causes VDE to use directory DEV2$:[PROJ.STAGE] as the staging area root directory instead of DEV$:[ROOTDIR.VDE$STAGE]. It is preferable that the staging area be on a different disk than the rest of the library.

- Enter the following command to actually create the specified staging area root directory:

```
VDE> CREATE DIRECTORY_TREE /STAGING
```

The SET DIRECTORY and CREATE DIRECTORY_TREE commands can be used at any time after library creation to redirect the staging area to a different disk location.

After a given replacement has been performed, VDE no longer needs the corresponding staging area. VDE does not itself delete those files, but you should do so periodically to reclaim the disk space. Staging area files can be useful for some time, however, because they contain copies of the files inserted into the VDE library. They can thus be used to recover lost information if your delta files (CMS libraries) are lost or damaged. In case of such loss, you have to restore the CMS libraries from backups. That should make them close to being consistent with the information in the VDE database. After that, you can use the VDE VERIFY GENERATION command with the /RECOVER qualifier to restore the CMS libraries to the state given by the VDE database. This recovery operation requires access to the old staging area directories—to retrieve copies of any lost module generations for insertion into the CMS libraries. For this reason, you should always have the staging areas on a different disk than the delta files or the VDE database and you should keep the staging areas for old replacements around for some time after the replacements are performed.

For information on typical staging area maintenance operations, see Section 7.5.

6.3.3 Development Stream Directories

If you intend to use VDE to build your software system, you must set up a directory tree for each development stream that you want to build. That directory tree will contain clear copies (as opposed to delta-file copies) of your source files along with all output files produced by the builds. If you intend to use VDE only for source control and not for building, you do not need directory trees for streams unless you want VDE to maintain clear copies of your modules for debugging or for general reference. This section explains both how to set up stream directory trees if needed and how to suppress them if not needed.

By default, VDE creates a directory tree for each new development stream that you create. The root directory for this directory tree has the same name as the stream and is a subdirectory of the library root directory. Any periods (.) in the stream name are changed to underscores (_) in the directory name to maintain valid OpenVMS file names. Within the stream root directory, there is one subdirectory for each facility in your software system. Each such subdirectory has the same name as the facility. Table 6–3 shows this structure.

Within each facility subdirectory, VDE creates several additional subdirectories. By default, there are four such subdirectories called SRC, OBJ, VDE$COM, and VDE$LOG. The SRC subdirectory is meant to hold source files in clear-copy form. The OBJ directory is meant to hold object files, executable images, and all other derived modules that can be

produced when your software system is built. The VDE$COM subdirectory will hold command files that VDE creates to perform build steps and the VDE$LOG subdirectory will hold the corresponding log files. The contents of the VDE$COM and VDE$LOG subdirectories are thus under VDE control while the contents of the SRC and OBJ directories are under your control when you set up the VDE library. These subdirectories are further illustrated in Table 6–3.

Table 6–3 Stream Disk Directories

Library root directory:	DEV$:[ROOTDIR]
Stream V1.5 root directory:	DEV$:[ROOTDIR.V1_5]
Facility FACNAM root directory:	DEV$:[ROOTDIR.V1_5.FACNAM]
Source-file subdirectory:	DEV$:[ROOTDIR.V1_5.FACNAM.SRC]
Object-file subdirectory:	DEV$:[ROOTDIR.V1_5.FACNAM.OBJ]
VDE build subdirectory:	DEV$:[ROOTDIR.V1_5.FACNAM.VDE$COM]
VDE build log-file subdirectory:	DEV$:[ROOTDIR.V1_5.FACNAM.VDE$LOG]
Source file MUMBLE.C:	DEV$:[ROOTDIR.V1_5.FACNAM.SRC]MUMBLE.C
Object file MUMBLE.OBJ:	DEV$:[ROOTDIR.V1_5.FACNAM.SRC]MUMBLE.OBJ

You can direct VDE to create additional subdirectories within the facility root directory and you can explicitly specify your own locations for the stream and facility root directories. This topic is covered in more detail later in this section.

Suppressing Stream Directories

If you want to use VDE for source control only and not for building, you may want to suppress the creation of stream directory trees because you are not going to use them. (However, if you want VDE to maintain clear copies of source files on replacement then you do need stream directories.) To suppress creation of stream directory trees, take the following steps:

- Specify the /DEFER qualifier with the CREATE LIBRARY command. This qualifier prevents VDE from creating the directory tree for the default stream MAIN when creating the library.
- Enter CREATE DIRECTORY_TREE commands with the /DELTA_FILES and /STAGING qualifiers in order to create the delta-file directory tree and the staging area root directory. See Section 6.3.1 and Section 6.3.2.
- Enter the SET DIRECTORY command with the /STREAM and /DEFER qualifiers for stream MAIN:

```
VDE> SET STREAM MAIN
VDE> SET DIRECTORY /STREAM /DEFER
```

  /DEFER specifies that when new streams are created from the current stream, directory-tree creation will be deferred for those streams. This means that the descendants of stream MAIN will not get

directory trees unless you enter explicit CREATE DIRECTORY_TREE commands for them.

After these steps are done, you can create new streams with the CREATE STREAM command as needed. Because of the SET DIRECTORY command, each subsequent CREATE STREAM command has an implicit /DEFER qualifier which suppresses creation of the stream's directory tree.

Defining Your Own Stream Directories

If you need stream directory trees but you do not like the default directory structure or placement, then you can define your own stream directories. One way to do so is to explicitly specify the root directory for new streams when you create the streams. Another way is to use the SET DIRECTORY command with the /STREAM qualifier to specify the directory structure and placement you want for a given stream. You can enter multiple SET DIRECTORY commands to specify different aspects of the desired directory structure; together these commands define a template for the directory structure and placement that you want. Then use the CREATE DIRECTORY_TREE command with the /STREAM qualifier to actually create the directory tree for that stream. In addition, any future child streams that are created from a given stream will inherit the directory structure of the parent. The actual directory trees for such descendant streams are created when the streams are created.

The following are the options you have in specifying the directory structure and placement for streams:

- To specify the placement of a new stream's root directory, create the stream using the /DIRECTORY qualifier with the CREATE STREAM command:

  ```
  VDE> CREATE STREAM BIRCH /DIRECTORY=DEV2$:[MY_BIRCH]
  ```

  This example creates stream BIRCH and makes DEV2$:[MY_BIRCH] its root directory. This is generally the best way to specify the root directory for a new development stream. Specifying stream root directories this way allows you to spread your VDE library over multiple disks for better performance and disk utilization.

  You can also specify that all future descendants of the current stream must have their root directories specified when those streams are created. To do so, enter a SET DIRECTORY command with the /PROMPT qualifier for the current stream. For example:

  ```
  VDE> SET STREAM BIRCH
  VDE> SET DIRECTORY /STREAM /PROMPT
  ```

  These commands specify that a stream root directory must be explicitly specified for each future descendant of the existing stream BIRCH. When each such descendant is created with the CREATE STREAM command, VDE prompts the user for the name of the stream root directory unless the user specifies it with the /DIRECTORY qualifier. This feature is useful when you want each future stream's directory tree to go on a separate device to be selected when the stream is created.

Note that the SET DIRECTORY /STREAM command acts on the current default stream. For it to act on a given stream, you must thus set that stream using the SET STREAM command, as shown in the example.

- To specify the placement of the root directory of an already existing stream's directory tree, use the SET DIRECTORY command with the /STREAM qualifier. Then use the CREATE DIRECTORY_TREE command to actually create the stream directory tree. For example:

```
VDE> SET STREAM BIRCH
VDE> SET DIRECTORY /STREAM DEV2$:[MYDIR]
VDE> CREATE DIRECTORY_TREE /STREAM
VDE> CANCEL DIRECTORY /STREAM
```

  The first three commands make directory DEV2$:[MYDIR] the root directory for the existing stream BIRCH. If these commands are not specified, VDE uses subdirectory BIRCH in the library root directory by default. After you have created the directory tree, you should use the CANCEL DIRECTORY command to prevent future child streams from inheriting the DEV2$:[MYDIR] root directory. This ensures that subsequent streams created from stream BIRCH do not attempt to use the same root directory; instead they get a subdirectory within the library root directory by default.

- To specify the placement of the root directory for a given facility for the current stream, use the /FACILITY qualifier as well:

```
VDE> SET STREAM BIRCH
VDE> SET DIRECTORY /STREAM /FACILITY=FOO DEV2$:[BIRCHFOO]
VDE> CREATE DIRECTORY_TREE /STREAM/ /FACILITY=FOO
VDE> CANCEL DIRECTORY /STREAM /FACILITY=FOO
```

  These commands make directory DEV2$:[BIRCHFOO] the root directory for facility FOO within stream BIRCH. If this command is not specified, VDE uses subdirectory FOO within the stream root directory by default. This form of the SET DIRECTORY command is seldom used because there is usually no need to place the directories for an individual facility on a different disk than other facilities. However, if you do use it, use the indicated CANCEL DIRECTORY command after creating the directory so that subsequent child streams will not use the same directory.

- To specify that a given stream's directory tree should have additional subdirectories within each facility's root directory, use the /SUBDIRECTORY qualifier to the SET DIRECTORY command. To specify that each facility should have a subdirectory for analysis (.ANA) files, for example, you would use the following commands:

```
VDE> SET STREAM BIRCH
VDE> SET DIRECTORY /STREAM /SUBDIRECTORY=ANASYM [.ANA]
```

  This SET DIRECTORY command specifies that each facility in stream BIRCH and its descendant streams should have a subdirectory ANA. The form of the directory specification ''[.ANA]'' indicates that it is relative to each facility's root directory. The ANASYM parameter is a ''directory symbol'' which is used later in VDE scripts to reference the new subdirectory. (Usually the directory symbol is the same as the directory name but here they differ for clarity of exposition.)

VDE gives you four facility subdirectories by default: SCR, OBJ, VDE$COM, and VDE$LOG. The /SUBDIRECTORY qualifier to SET DIRECTORY allows you to specify additional such subdirectories. As always, the SET DIRECTORY command only creates a template for the desired directory tree; the actual directories are not created until you use the CREATE DIRECTORY_TREE command or create new streams that inherit the template from their parent.

These various ways of specifying directory placement and structure give you considerable control over VDE's stream directory trees. You can spread your VDE library over multiple disks using these capabilities. You can control how the various subdirectories for each facility are used by the way that you write the VDE scripts that determine how build steps and certain other operations are performed.

6.4 Adding Users to the Library

Before users other than yourself (the library creator) can use the new VDE library you have created, those users must be added to the library's database. This was described briefly in Section 6.2.3. When a user is added to the database, VDE creates a User Table entry for the user in the VDE database. When that user subsequently tries to access the VDE library through the VDE utility, VDE checks that the user is recorded in the database. Provided this is the case, VDE retrieves the user's VDE privileges and other attributes from the User Table entry and uses those attributes to control the rest of the VDE session. VDE denies access to the library to users who are not recorded in the database.

There are three ways of adding users to your library's database:

- VDE automatically adds the library creator to the database when the library is created. This user is authorized to set all VDE privileges and has all administrative privileges set by default.
- You can use the CREATE USER command to add users. This command lets you manually specify the users to add to the database, including their privileges and other attributes.
- You can enable a library attribute that causes VDE to automatically add new users to the library database when they first access the library through the VDE utility. Each such user gets the standard default and authorized privileges for the library.

The addition of the library creator has been described previously in this chapter. The rest of this section describes the other two methods in more detail.

6.4.1 The CREATE USER Command

The CREATE USER command lets you specify the OpenVMS username, the full name (first and last names), the VDE default and authorized privileges, and certain other information about each user to be added to the database. This is an example of a CREATE USER command that adds user Jane Smith to the database:

```
VDE> CREATE USER SMITH /NAME="Jane Smith" /REMARK="Project leader" -
_VDE>         /AUTH_PRIVILEGES=(CREFAC,CREMOD,CRESTRM,RESREP,PERFREP) -
_VDE>         /DEF_PRIVILEGES=(CREMOD,RESREP,PERFREP)
```

After this command has completed, Jane Smith can start VDE sessions using the VDE library. By default, she has the privileges necessary to create new source modules, to reserve and replace modules, and to perform queued replacements for herself and other developers. In addition, she is authorized to set the privileges necessary to create new facilities and development streams.

Using the CREATE USER command is the most practical way to add users to a VDE library if the library has a small number of users or if you want to exercise careful control over who accesses the library.

6.4.2 Automatic Addition of Users

If you specify the /AUTO_ADD_USER qualifier with the CREATE LIBRARY or MODIFY LIBRARY command, VDE will automatically add new users to the library database the first time each such user uses VDE to access the library. If this qualifier is used without a parameter, any user who attempts to access the VDE library through VDE is automatically added to the database. Each such user gets the standard default and authorized VDE privileges for that library. The OpenVMS username is retrieved from the operating system. The full name of the user is taken from the owner field of the user's record in the system User Authorization File (UAF). Other attributes, such as the remark string, are not set.

The /AUTO_ADD_USER qualifier can also be specified with a OpenVMS rights identifier parameter, as in /AUTO_ADD_USER=*rights-id*. In this case, VDE stores the given rights identifier in the database. Subsequently, VDE adds only those users to the database who hold the stored rights identifier. The users are added when they first access the library through VDE, just as for the plain /AUTO_ADD_USER qualifier. This way of adding users is useful when you use a OpenVMS rights identifier to protect the files and directories of your project. Because the intended users of your VDE library must hold the rights identifier anyway, you can use the rights identifier as your mechanism to control access to the library.

If you use automatic addition of users to your VDE library, you should periodically run the program VDE/READ_UAF. This program reads the system user authorization file (SYSUAF) of your OpenVMS system and updates the User Table in your VDE database to be consistent with the SYSUAF. It updates each user's full name in the VDE database to match the owner field in the SYSUAF. VDE/READ_UAF also marks users as deleted in the VDE database if they no longer have accounts in the UAF. VDE/READ_UAF thus allows you to maintain your database User Table by having it automatically track the SYSUAF, which your system manager must maintain anyway. And with /WRITE_UAF, the command can update the process quotas of OpenVMS users in the System Authorization Database (SYSUAF). The VDE/READ_UAF command should be included in a command procedure that runs regularly, perhaps every night or every weekend.

To run VDE/READ_UAF, you must specify the target VDE via the /LIBRARY qualifier or via the VDE$LIBRARY logical name. You must also have read access to the OpenVMS system authorization database, or read and write access when the /WRITE_UAF qualifier is specified. *Installation of the VDE/READ_UAF image with enhanced privileges is not recommended.* You can execute the program as follows:

```
$ SET PROCESS/PRIVILEGE=BYPASS
$ VDE/READ_UAF/WRITE_UAF/LIBRARY=lib-name
```

Automatic addition of users is the most practical approach for VDE libraries with a large number of potential users. Even if you automatically add users, you can of course still use the CREATE USER command to add users. This makes sense if you want to add a user who does not hold the rights identifier required to be added automatically or who should be added with a different set of privileges than the standard privileges for the library. In addition, you can always use the MODIFY LIBRARY command to change the privileges or other attributes of any user already recorded in the database.

The VDE/READ_UAF command, with the specification of the /WRITE_UAF qualifier and with read-write access to the System Authorization Database, can update process quotas of VDE users in the System Authorization Database (SYSUAF). This command is particularly useful when users are reporting quota-related messages when invoking VDE.

6.5 Creating Streams

Section under construction...

To create a stream, use the CREATE STREAM command. You normally need to select a parent stream for the operation, and you need to consider the stream propogation order.

```
VDE> CREATE STREAM V2.0/PARENT=V1.0
```

In addition to choosing the parent stream, you need to choose whether or not the new stream is a direct descendent of the parent stream (/MAINLINE), or if it is a variant and will diverge (/NOMAINLINE). (Mainline streams will keep the same generation identification values as the parent, while variant streams select longer generation identification values. The main line of descent is often an integer generation number, while variants tend to include increasing numbers of letters in the identification value. There are no other differences. Once the stream has been created, this setting cannot be easily changed.)

Other qualifiers of interest during stream creation include /CONFERENCE, /[NO]INIT_CMS_CLASS, /NOTIFY, /[NO]GENERATIONS, and /REVIEW.

Once created, streams can be repopulated via a POPULATE STREAM command. Once a stream is populated, either via the initial CREATE STREAM command or via the POPULATE STREAM command, a MODIFY STREAM/NOPOPULATE command should be issued, to prevent accidental repopulation of the stream.

6.5.1 Stream Directory Tree

Section under construction...

Streams typically have an associated directory tree.

6.5.2 Scripts to Populate New Streams

Section under construction...

To cancel a module reservation without replacing the module, use the ...

6.5.3 Stream Successor Relationships

Section under construction...

To cancel a module reservation without replacing the module, use the

6.6 Creating Facilities

Section under construction...

To cancel a module reservation without replacing the module, use the UNRESERVE command. The following example illustrates its use:

```
VDE> CREATE FACILITY FACNAM -
_VDE> /REMARK="The fumble utility"
```

This command cancels the reservation of module MOD.TYP in facility FACIL. The /STREAM qualifier specifies the stream for which the reservation was made. If you omit this qualifier, your default stream is assumed. If you have more than one reservation of the module for that stream, you must specify its reservation identifier with the /IDENTIFICATION qualifier. The command can cancel both queued reservation requests and real reservations.

6.7 Populating a Facility with Modules

Section under construction...

There are various ways to load modules into a VDE database. One can upload modules into a facility from an existing CMS library, an operation that normally only occurs during library creation, and one can also upload a directory containing a number of modules into a facility in a VDE database using the CREATE MODULE/FROM_DIRECTORY command. And one can upload individual modules into a facility using the CREATE MODULE command.

A new VDE database can be created and loaded using the procedure described in Section 6.7.3.

6.7.1 Getting Source Modules from a Directory

One can use the CREATE MODULE/FROM_DIRECTORY command for this task. This command will create one module in the facility for every file found in the specified directory. When /FROM_DIRECTORY is specified, no module name parameter is specified on the CREATE MODULE command.

```
VDE> CREATE MODULE/FROM_DIRECTORY=DEV$:[NEWFAC]
```

6.7.2 Getting Source Modules from a CMS Library

This section includes a code example that shows the steps involved in the conversion from a major CMS-based project into a project based on a VDE database. The project involves the importation of multiple CMS libraries for the various software units—what VDE calls facilities—into VDE. Each facility is organized in a CMS library located under the project-specific root directory DISK$EVMSCMS:[VMS].

Note: During the execution of the attached example code, *no* changes to the format of the CMS libraries are made.

Note: Before performing the CREATE FACILITY/FROM_CMS_LIBRARY command, one must create all VDE streams—those streams with the same name as the CMS classes to be uploaded from the CMS library into the VDE database, and one must also specify the /MAIN or /NOMAIN, /PARENT and /SUCCESSOR values correctly. If these operations are not correctly created and configured *before* issuing the CREATE FACILITY/FROM_CMS_LIBRARY command, CMS generation selection, and the variant and mainline streams will not be correctly configured.

```
$ set noon
$
$! These logicals will need to be (re)defined in the
$! site-specific system startup command procedure.
$
$ Define/System/Executive/Nolog VMS$ DISK$EVMSCMS:[VMS.]/Trans=Conceal
$ Define/System/Executive/Nolog VSC$DATADISK $10$DKB400:
$ Define/System/Executive/Nolog VSC$ROOTDISK $10$DKB400:
$ Define/System/Executive/Nolog VSC$SNAPDISK $10$DKB400:
$ Define/System/Executive/Nolog VSC$JNLDISK $10$DKB400:
$
$ Define/System/Executive/Nolog VSC$DBDATA VSC$DATADISK:[evms.dbdata]
$ Define/System/Executive/Nolog VSC$DBROOT VSC$ROOTDISK:[evms.dbroot]
$ Define/System/Executive/Nolog VSC$DBSNAP VSC$SNAPDISK:[evms.dbsnap]
$ Define/System/Executive/Nolog VSC$DBJNL  VSC$JNLDISK:[evms.dbjnl]
$
$ Create/Directory VSC$DBDATA
$ Create/Directory VSC$DBROOT
$ Create/Directory VSC$DBSNAP
$ Create/Directory VSC$DBJNL
$
$ VDE
create librar VSC$DBROOT /DEFER -
 /NAME=EVMS /REMARK="EVMS Source Library" -
        /NOAUTO_CONVERT /AUTO_ADD_USER=VMS_SOURCE /AUTO_MAIL -
        /ASK_INFO_FILE /ASK_REPL_CAUSE /SHOW_GEN_EXPR -
        /NOHISTORY_NOTES /NOALLOW_DELETE /NOWILD_INS_GEN -
        /NOCMS_ELEM_HIST /CREATOR="Stephen Hoffman" -
        /DEF_PRIVILEGES=(CREMOD,RESREP) -
```

```
        /AUTH_PRIVILEGES=(CREMOD,RESREP) -
 /data=VSC$DBDATA/journ=VSC$DBJNL/snap=VSC$DBSNAP -
!
SHOW DEFAULTS
SHOW LIBRARY/FULL
MODIFY CONTEXT/PUBLIC DEFAULT/NOSTREAM/NOFACILITY
!
SET DIRECTORY/DELTA_FILES VMS$:[000000]
SET DIRECTORY/DELTA_FILES/SUBDIRECTORY=VDE$CMS [.CMS]
SET DIRECTORY/DELTA_FILES/SUBDIRECTORY=VDE$MARKER [.SRC]
MODIFY FACILITY CODE/NAME=AAA_VDE_TEST/REMARK="Test facility for VDE developers"
CREATE DIRECTORY_TREE/DELTA_FILES
!
SHOW DIRECTORY/DELTA_FILES
!
! Set to the AAA_VDE_TEST facility for use by database administion,
! and populate it with a test module or two.
!
SET FACILITY AAA_VDE_TEST
CREATE MODULE/NODELETE/REMARK="VDE test module" LOGIN.COM /INPUT=SYS$LOGIN:
CREATE MODULE/NODELETE/REMARK="VDE test module" SYSTARTUP_VMS.COM /INPUT=SYS$STARTU
!
SET DIRECTORY/STAGING disk$common:[vms.stage]
CREATE DIRECTORY_TREE/STAGING
!
! Rename stream MAIN to be MAINLINE.  We will use this stream for a few
! selected modules that should always stay on the main development thread
! outside the normal development streams.
!
MODIFY STREAM MAIN/NAME=MAINLINE/REMARK="Mainline thread for selected modules"
SHOW STREAM/FULL MAINLINE
SET DIRECTORY/STREAM disk$common:[VMSMAINLINE]
CREATE DIRECTORY_TREE/STREAM
!
! Create the V6.1 parent stream to all others streams.  Specification
! of /MAIN or /NOMAIN, and /PARENT and /SUCCESSOR are key factors in
! the CMS generation selection and the propogation processing.  All
! streams must be created prior to specification of CREATE FACILITY
! /FROM_CMS_LIBRARY.  See the reference manual for details.
!
SET STREAM MAINLINE
CREATE STREAM/CLOSED/MAIN V6.1/PARENT=MAINLINE/REMARK="create release class"
CREATE STREAM/CLOSED/MAIN V6.2/PARENT=V6.1/REMARK="create release class"
CREATE STREAM/CLOSED/MAIN V6.3/PARENT=V6.2/REMARK="create release class"
!
SHOW DIRECTORY/DELTA_FILES
!
Exit
$ SHOW TIME
$ VDE
create facility ACAS/FROM_CMS/REMARK="ACAS"
create facility ACC/FROM_CMS/REMARK="Accounting"
create facility ACLEDT/FROM_CMS/REMARK="ACL Editor"
....
create facility ZSYSGEN/FROM_CMS/REMARK="ZSYSGEN"
create facility ZSYSVER/FROM_CMS/REMARK="ZSYSVER"
Exit
$ SHOW TIME
```

6.7.3 Example CMS Upload Procedures

VDE includes example command procedures and associated files in VDE$EXAMPLES:, the core example procedure used for uploading an existing CMS library into VDE is VDE$CREATE.COM. Files associated with VDE$CREATE.COM will need to be tailored to local requirements and to the local CMS library directory structure.

Note: To avoid losing any changes made during any subsequent VDE upgrades, relocate the example command procedures into a non-VDE directory *before* making any edits.

Also see Section 6.1.1 for information on VDE$UPLOAD, a tool that can directly perform the task of library creation based on a description of the desired VDE database contained in a data file.

6.7.4 Adding Individual Source Modules

Adding modules into a VDE library is a simple operation.

```
VDE> CREATE MODULE/NODELETE/REMARK="user login module" -
_VDE>     LOGIN.COM /INPUT=SYS$LOGIN:
```

For further information on the above command and for the available command options, please refer to the *VDE Reference Manual*.

7 Library Maintenance Operations

This chapter explains how how to perform various maintenance operations. These operations include clearing deadlocks, performing the source code database backups, creating and maintaining the associated backup procedures, restoring the database after a failure, and recovering lost information in the CMS libraries on a VDE library. Also included are the procedures involved in periodic cleanups of the staging area directories, and one approach toward storage recovery within VDE libraries. Some of these commands and operations will require you to have the OpenVMS SETPRV or SYSPRV privileges.

Operations described in this chapter are normally performed only by the maintainers of the OpenVMS VDE libraries, or by those users maintaining their own VDE libraries.

The command procedures described in this chapter are located in directory MASD$JNL:[VMS.TOOL] on the journal disk for the OpenVMS VDE libraries. Because these procedures may be needed in case of journal disk failures, they are also stored in the directory VMS$:[TOOLS].

7.1 Creating SCT Notes Conferences

For the automatic submission of the SCT note to operate properly, the SCT notes conference must have the correct protections. The following shows the commands necessary to enable the security necessary for the typical SCT notes conference:

```
$ set security/acl=-
_$ ((IDENTIFIER=VMS_SOURCE_READ,ACCESS=R+W+E),-
_$  (IDENTIFIER=VMS_SOURCE,ACCESS=R+W+E)) -
_$ NOTES$LIBRARY:SCT-*.NOTE
$ set security/acl=-
_$ ((IDENTIFIER=[DECNET,NOTES$SERVER],ACCESS=R+W),-
_$  (IDENTIFIER=NETWORK,ACCESS=NONE)) -
_$ /PROTECTION=(S:RWED,O:RWED,G:RE,W) -
_$ NOTES$LIBRARY:SCT-*.NOTE
```

The above security attributes and resource identifiers can and should be adjusted for site-specific requirements.

Failure to correctly set up the security attributes can prevent the creation of SCT entries by VDE.

7.2 Taking VDE Offline

Depending on requirements, one can use the MODIFY LIBRARY/LOCK command to mark a particular VDE library as read-only to all but privileged VDE users, and one can define the logical name VDE$OFFLINE to take the VDE environment completely offline.

```
$ Define/System/Executive VDE$OFFLINE -
_$ "Down for maintenance until 23:00"
```

Users with privileges within VDE can override the library lockout; a message indicating that a library lock is being overridden is displayed in this situation. For further information on this particular command, see the *VDE Reference Manual* manual.

Short of deassigning the VDE$OFFLINE logical name, there is no way to override the VDE$OFFLINE lockout—the VDE$_OFFLINE message and the translation of the logical name will be displayed to all users. Once this privileged logical name has been defined, users cannot invoke the VDE images, and users already executing VDE will be unable to invoke any new VDE commands.

Note that it can be difficult to debug certain VDE problems when the VDE$OFFLINE lockout is in place, as even the VDE maintainers cannot invoke the VDE images. This lockout is best suited for the protection of Rdb and CMS maintenance operations.

7.3 Resolving Database Problems

Various things have been seen to go wrong within VDE, or within Rdb. Failures include deadlocks, and various Rdb problems that result in process-level and system-level bugcheck dumps.

VDE has occasionally been known to deadlock at the Rdb level. This deadlock condition is usually heralded by a series of ‘‘VDE is wedged’’ complaints from various vociferous VDE users.

Note: Before attempting to clear a ‘‘stuck’’ VDE database, first determine if there is a valid reason for sluggish database access. VDE or Rdb commands executed by another database user. Project leaders use some commands and perform some operations that can potentially be quite time-consuming, while Rdb database backups are notorious for clobbering performance. Any of these can lead to intervals where the VDE database processing is extremely sluggish or various commands are locked out.

7.3.1 User Process Deadlocks

Process-level deadlocks occur infrequently, and they often involve high-traffic applications such as database backups or they involve a VDE command that was paused via CTRL/S XOFF command, and they can usually be easily and safely cleared using the following command:

```
$ RMU/CLOSE/CLUSTER/ABORT=DELPRC/WAIT-
_$ VDE$:[ROOTDIR]VDE$DATABASE
```

The above RMU command deletes all processes currently connected to the specified database. As most VDE users utilize kept subprocesses, this process deletion is often entirely transparent to users.

To use the RMU/CLOSE command, you must possess the OpenVMS SYSPRV or BYPASS privilege, or you must have access via the Rdb database access control list. If the RMU/CLOSE command hangs for an extended interval (it should hang for a while pending cluster-wide closure), see if the ‘‘freeze’’ deadlock mentioned in Section 7.3.2 has occured.

7.3.2 Freeze in Rdb

It is possible for Rdb processes to deadlock waiting for the Rdb ‘‘freeze’’ resource.

When a ‘‘freeze’’ deadlock occurs, one normally finds any attempt to SET LIBRARY to the VDE library or to run various RMU commands—including the RMU/CLOSE command mentioned in Section 7.3.1 – against the library will hang indefinitely. This deadlock can survive the shutdown of all Rdb database monitors, though a shutdown of the node holding the deadlocked recovery process can relieve the deadlock.

The following example shows the particular process holding the ‘‘freeze’’ lock:

```
$ RMU/SHOW LOCK/MODE=BLOCKING
================================================================================
SHOW LOCKS/BLOCKING Information
================================================================================

--------------------------------------------------------------------------------
Resource: freeze

           ProcessID Process Name     Lock ID   System ID Requested Granted
           --------- ---------------  --------- --------- --------- -------
Waiting:   224018E0  SHOFFMAN_00E0..  3F00433E  00010002  CW        NL
Blocker:   20402DC8  RDM_RB60_1.....  5D00A098  00010002  PR        PR
```

One can use ANALYZE/SYSTEM or similar to attempt to locate the reason the blocking process—RDM_RB60_1 in the above example—in an attempt to resolve the problem.

7.3.3 Rdb process dumps

Rdb generates a dump file when it encounters an unexpected condition. Under typical circumstances, Rdb will display a message indicating a dump is being generated.

```
VDE> set lib evms
Generating bugcheck dump file WORK10:[user]RDSBUGCHK.DMP
```

One potential cause is the inability to write to the run-unit journal area, see Section 7.3.4 for information for details.

7.3.4 Rdb run-unit journal errors

The run-unit journal area is a key component of the Rdb processing, and allows Rdb to rollback incomplete or erroneous database operations. And when an unexpected condition occurs with the run-unit journal, Rdb tends to generate a dump file.

Errors involving the run-unit journal have a variety of footprints, including symptoms such as process-level Rdb bugcheck dumps and user process deletions.

7.3.4.1 Missing run-unit journal area

If the run-unit journal area is entirely messing, one can potentially see an Rdb bugcheck dump generated. By default, the run-unit journal is written to SYS$LOGIN_DEVICE:[RDM$RUJ], but this can be altered via the Rdb RDMS$RUJ logical name.

```
VDE> set lib evms
Generating bugcheck dump file WORK10:[user]RDSBUGCHK.DMP
```

When the run-unit journal area is entirely messing or or is incorrectly protected, Rdb will generate a process-level bugcheck dump file, and VDE will not operate correctly. (In addition to the above message, it is also possible one may see VDE stack-dumps.

For any user seeing these errors, make certain that SYS$LOGIN_ DEVICE:[RDM$RUJ] exists and is owned by [SYSTEM]. If it does not exist, create it. (Note that the directory is actually based on the device from SYS$LOGIN:, not from the SYS$LOGIN_DEVICE: specification.)

```
$ create/directory work10:[rdm$ruj]/owner=[system]
$ set security/owner=[system] -
_$ work10:[000000]rdm$ruj.dir,[rdm$ruj]*.*.*
$ run sys$system:sysman
SYSMAN> DISKQUTA ADD [SYSTEM] -
_SYSMAN> /DEVICE=WORK10/PERM=100000/OVER=1000
SYSMAN> CTRL/Z
```

If [SYSTEM] already has a quota entry for WORK10:, use the SYSMAN command DISKQUTA MODIFY in place of DISKQUOTA ADD.

Based on Rdb documentation, one can also redirect the location of the SYS$LOGIN_DEVICE:[RDM$RUJ] directory via the Rdb RDMS$RUJ logical name. (Note that Rdb does not actually use SYS$LOGIN_DEVICE:, it appears to parse SYS$LOGIN:.)

This particular error is more common on volumes that were not initialized with [SYSTEM] ownership:

```
$ INITIALIZE work10: work10/SYSTEM
```

Also see Section 7.3.4.2 for a related problem.

7.3.4.2 Rdb run-unit journal diskquota errors

Rdb requires the creation of a run-unit journal file. By default, this is in the user's login device, in the directory [RDM$RUJ], and this location can be altered using the logical name RDMS$RUJ. If this creation operation fails or due to insufficient quota, one can see the following errors:

```
$ vde/library=evms
%RDB-F-SYS_REQUEST, error from system services request
-RDMS-F-FILACCERR, error creating run-unit journal file
WORK1:[RDM$RUJ]VDE$DATABASE$0099C91EC6BCC945.RUJ;1
-RMS-E-CRE, ACP file create failed
-SYSTEM-F-EXDISKQUOTA, disk quota exceeded
```

To resolve this particular error for the quotas present on device WORK1:, issue the following commands:

```
$ set security/owner=[system] -
_$ work1:[000000]rdm$ruj.dir,[rdm$ruj]*.*.*
$ run sys$system:sysman
SYSMAN> DISKQUTA ADD [SYSTEM] -
_SYSMAN> /DEVICE=WORK1/PERM=100000/OVER=1000
SYSMAN> CTRL/Z
```

If [SYSTEM] already has a quota entry for WORK1:, use the SYSMAN command DISKQUTA MODIFY in place of DISKQUOTA ADD.

Also see Section 7.3.3 for a related situation.

7.3.4.3 Rdb recovery process dumps

Rdb generates a dump file when the recovery process encounters an unexpected condition. (See Section 7.3.4.4 for one example of an unexpected condition.) There may or may not be any indication this dump is being written—in one particular case, the process that was triggering the failing recovery process was being logged off the system.

Under typical circumstances, the recovery process dump file is written to SYS$SYSTEM:RDMDBRBUG.DMP. The following is an example of a recovery process dump file:

```
================================================================================
          Bugcheck dump SYS$SYSROOT:[SYSEXE]RDMDBRBUG.DMP;17
================================================================================

This file was generated by DEC Rdb V6.0-1 upon detection of a
fatal, unexpected, error.  Please return this file, the query or
program that produced the bugcheck, the database, monitor log,
and any other pertinent information with an SPR.

Current time is 21-JAN-1997 15:20:21.10

 3-FEB-1997 16:23:35.84: Linked DBR (RDBVMS) RDMS$ALPHA_V0601:[KODABLD]
25-JAN-1997 08:06:24.32: Compiled KOD$DBR (KODA) KOD$ALPHA_V0601R:[DBR]
25-JAN-1997 08:06:15.13: Compiled KOD$LIBRARY (KODA) KOD$ALPHA_V0601R:[CODE]
================================================================================
          Stack Dump
================================================================================

Saved PC = 001383C0 : STD$DUMP_ALPHA_STACK + 00000090
Saved PC = 001212E4 : KOD$BUGCHECK_DUMP + 00000D74
Saved PC = 000A1214 : KOD$BUGCHECK_AND_EXIT_HNDLR + 00000094
Saved PC = 80895D44 : S0 address
Saved PC = 962A62BC : S0 address
***** Exception at 0006B930 : DBR$RECOVER + 000009F0
%RDMS-F-FILACCERR, error opening run-unit journal file
WORK1:[RDM$RUJ]VDE$DATABASE$0099C91EC6BCC945.RUJ;1
-SYSTEM-W-ACCONFLICT, file access conflict
Saved PC = 0006AB50 : DBR + 00000B90
Saved PC = 00057F0C : BLI$CALLG + 000000BC
Saved PC = 0005011C : DBR$MAIN + 0000011C
Saved PC = 962D3C44 : S0 address
[...rest of dump file has been omitted...]
```

The above example clearly points to an access conflict on a run-unit journal file. The above recovery process failure was cleared by deleting the processes that were holding open run-unit journal, and restarting the recovery process.

```
$ run sys$system:sysman
SYSMAN> set environment/cluster
SYSMAN> set profile/privilege=all
SYSMAN> do show device/file/nosystem work1:
SYSMAN> CTRL/Z
```

7.3.4.4 Missing run-unit journal files

A recovery-process dump similar to Section 7.3.4.3 can also be seen when the RUJ file is missing. This can be a result in user process deletions and failed RMU backups, as well as other nasty behaviour.

One sees the following in an RMU/DUMP:

```
Active user with process ID 38810DD3
    Stream ID is 1
    Monitor ID is 2
    Transaction ID is 16
    Recovery journal filename is "WORK300:[RDM$RUJ]VDE$DATABASE$009A486CCA4B098E.RU
    Read/write transaction in progress
    Transaction sequence number is 183093
```

and one sees the following in SYS$SYSTEM:RDMDBRBUG.DMP:

```
***** Exception at 000253ED : DBR$RECOVER + 000005CB
%RDMS-F-FILACCERR, error opening run-unit journal file WORK300:[RDM$RUJ]VDE$DATABAS
-RMS-E-FNF, file not found
```

There's no good solution to this problem if the file cannot be located, short of locating the missing RUJ, or recovering the database from Rdb BACKUP.

The following steps will clear the missing RUJ and will leave the\ database accessable, *but the database is now likely corrupt.* Recovery from RMU/BACKUP is preferable.

```
$ RMU/ALTER databasename❶
RdbAlter> DISPLAY ROOT USER❷
RdbAlter> DEPOSIT ROOT USER # RUJ=""❸
RdbAlter> COMMIT❹
RdbAlter> EXIT
$ RMU/UNLOAD...❺
$ RMU/VERIFY/ALL❻
```

❶ Invoke the RdbAlter utility from a suitably privileged username, specifying the name of the target database file.

❷ Display all ''users'', numbered from 0 to n, and display any RUJ files that Rdb believes are needed for recovery.

❸ Specify the user number with the missing RUJ file in the # position. No spaces or characters should be specified between the double-quotes.

❹ COMMIT the transaction. *The database is now accessable, but is likely corrupt.* Recovery from Rdb BACKUP is preferable.

❺ Unload the contents of the database, verify the contents, and re-import.

❻ Use a minimum of RMU/VERIFY/ALL to verify the data structures of the reloaded database. Even with proper verification of the structures, the information in the database may be corrupt.

7.4 Database Backup and Recovery Procedures

Section under construction...

After establishing your VDE library, you should set up the backup and recovery procedures that allow you to recover the contents of your VDE library in case the library is destroyed or damaged. A disk failure, for example, could destroy your VDE database or any of the associated delta files (CMS libraries). To be able to recover from such failures, you must establish several separate backup procedures. This section describes those procedures.

7.4.1 Overview of Backup Procedures

To establish the necessary backup procedures for your VDE library, you need to take the following actions:

- Place the directories and files associated with your VDE library on two or more disks. In order to recover from the failure of any one disk, you must have VDE backup and journal files on a different disk than the disk or disks that hold the rest of your VDE library.
- Enable after-image journalling for your VDE database. After-image journalling causes Rdb to record all changes to your VDE database in a journal file. If you later must restore your database from a backup, you can use the journal file to ''roll'' the restored database forward until it contains all changes that your lost database had just before the failure.
- Edit the database backup and restore procedures that come with the VDE kit to make them refer to your VDE library. Place these procedures in a backup directory associated with your VDE library. Configure the site-specific backup procedure to execute periodically.
- Make sure that someone performs regular backups of all files in your VDE library using the OpenVMS BACKUP utility. These backup are normally written to tape and backup tapes should periodically be moved to fire storage. If your system manager or operations staff performs this function for you, you need only ensure that the appropriate staff knows to back up the disks that hold your VDE library. Otherwise, you may have to perform such backups yourself.

  See Section 7.4.4 for information specific to the synchronization problems that are likely when attempting to use OpenVMS BACKUP utility to perform backups of Rdb database files such as those used by VDE.
- For additional protection against disk or other hardware failures, you can consider using OpenVMS shadow sets for the disks that hold your VDE library and its backup files. Disk shadowing is appropriate when your VDE library requires high availability and when the cost of an unavailable library is high. Using disk shadowing doubles the number of disks you need.

The sections that follow describe how you perform these steps. They also describe how you restore your VDE database and other files in your VDE library in case of a failure. However, this manual does not further describe how to use the OpenVMS BACKUP utility to back up files and directory trees. Neither does it describe how to set up disk shadowing. For a full description of these topics, see the appropriate OpenVMS documentation or see your system manager.

7.4.2 Placement of Backup Files and Directories

Section under construction...

7.4.3 After-Image Journalling

Section under construction...

7.4.4 Creating the Database Backup Procedure

Section under destruction...

Note: Due to Rdb database transaction synchronization issues, disk or VDE database backups performed with the OpenVMS BACKUP utility can not be relied upon for the backup and the restoration of VDE databases.

The database for your VDE library is an Oracle Rdb7 database. (Rdb is a relational database package.) An Rdb database must be backed up periodically using the RMU utility described in the *Oracle Rdb7 RMU Reference Manual* and the *Oracle Rdb7 Guide to Database Maintenance*.

The OpenVMS BACKUP command can potentially take a snapshot of your database as it looks in the middle of an update transaction. This results in an invalid backup file because Rdb must preserve transaction semantics, meaning that an update transaction must complete fully or not at all.

The RMU utility, a component of the Rdb product, ensures that the backed-up database is internally consistent and observes transaction semantics, which the OpenVMS BACKUP utility cannot do. Unlike OpenVMS BACKUP, the RMU utility takes a consistent snapshot of the database as it is synchronized with any database transactions that might potentially be active. The RMU utility is also be used to back up and maintain the Rdb after-image journal files. These journal files allow Rdb to recover database transactions performed after the last backup operation.

The VDE kit contains a command procedure that you can modify to back up the Rdb database for your VDE library. This command procedure performs the appropriate RMU operations. To create the database backup procedure for your VDE library, perform the following step:

- *Section under construction...*
- *Section under construction...*

Section under construction...

7.4.5 Creating the Database Restore Procedure

Section under construction...

7.4.6 Backing up the VDE Database

Section under destruction...

To back up your VDE database, you must first set up a database backup procedure as described in Section 7.4.4. In the present section, we will assume that this procedure is called VDE_DATABASE_BACKUP.COM and is found in directory JNLDEV$:[PROJ.TOOL] as suggested in Section 7.4.2. We will also assume that disk JNLDEV$ is your journal and backup device and that you have set up subdirectories [PROJ.BACKUP], [PROJ.BACKUP_OLD], and [PROJ.AIJ] as outlined in Section 7.4.2.

Once your database backup procedure is in place, you can run it to perform full backups with database verification, full backups without database verification, or incremental database backups. A full backup saves all database information in a backup file. An incremental backup saves all database information that has changed since the last full backup in an incremental backup file. Database verification causes the RMU utility to check the internal structure of the Rdb database for consistency and validity.

To perform a full database backup without verification, you invoke your edited backup command procedure as follows:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_BACKUP FULL_NOVERIFY
```

This invocation of the procedure performs a full backup of the source control database. It does not attempt to verify the database.

The backup procedure performs the following steps for a full backup. It deletes all files in the JNLDEV$:[PROJ.OLD_BACKUP] directory and then renames all files in the JNLDEV$:[PROJ.BACKUP] directory to the OLD_BACKUP directory. The previous backup files are thus preserved in the OLD_BACKUP directory so they cannot be lost in case the current backup operation fails. Next, the RMU/BACKUP command performs a full database backup, producing the Rdb backup file VDE$DATABASE_FULL.RBF in the BACKUP directory. An ‘‘online backup’’ is used, meaning that other users can use the VDE database while it is being backed up. Finally, the active after-image journal file is backed up to the BACKUP directory under the name VDE$DATABASE_FULL.AIJ, after which the active journal file is set to be empty. After all this is done, the BACKUP directory, along with the active journal file in directory JNLDEV$:[PROJ.AIJ], contains all files needed to fully recover the VDE database.

If you want to perform a full backup and also verify the database (using the RMU/VERIFY command), invoke the backup procedure as follows:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_BACKUP FULL
```

For a large database, verification can be very time-consuming. You should nevertheless verify the database periodically, such as weekly.

If you omit the FULL parameter on the command line, you get a FULL database backup by default.

An incremental backup saves all changes to your database since the last full backup in an incremental backup file. To perform an incremental backup of the database, enter the following command:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_BACKUP INCREMENTAL
```

For an incremental backup, the backup procedure invokes RMU to perform an online incremental backup of the database. The output file is file VDE$DATABASE_INCR.RBF in the BACKUP directory. This file contains all changes since the last full backup. An incremental backup normally takes substantially less time than a full backup. How much less depends on how much your database has been changed since the last full backup.

By default, the VDE_DATABASE_BACKUP procedure displays the inputs and outputs of the backup operation and asks you to confirm that those are the right inputs and outputs. You must then answer YES if you want the backup operation to proceed. This is an example:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_BACKUP FULL_NOVERIFY
    The backup operation will be performed using the following
    device and directory specifications:

    The location of the database to back up is LIBDEV$:[PROJROOT]
    The database will be backed up to JNLDEV$:[PROJ.BACKUP]
    Existing backup files on JNLDEV$:[PROJ.BACKUP] will be renamed
        to JNLDEV$:[PROJ.BACKUP_OLD] before the backup starts
    The definition of VDE$JOURNAL is JNLDEV$:[PROJ.AIJ]

Do you want to continue with the database backup? (Y or N) : Y
```

To suppress the request for confirmation, you can add the keyword NOQUERY to the end of the command line invoking the procedure. For example:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_BACKUP FULL NOQUERY
```

You always need the NOQUERY keyword when you run the backup procedure in a batch job.

Because database backups should be performed regularly, the normal way to perform them is to submit a command procedure that runs every night to perform a full backup once a week and incremental backups the other nights. To start this command procedure, enter the following DCL command:

```
$ SUBMIT /LOG=JNLDEV$:[PROJ.LOG]NIGHTLY_VDE_BACKUP.LOG -
_$       /NOPRINT/KEEP/RESTART/AFTER="TOMORROW+1:00" -
_$       JNLDEV$:[PROJ.TOOL]NIGHTLY_VDE_BACKUP.COM
```

This command procedure resubmits itself each time it runs. You should therefore not need to restart it manually unless batch queues have been lost due to a system failure. In this example, the SUBMIT command causes the procedure to run at 1:00 am the following night. That is also the time that the procedure restarts itself on subsequent nights.

The database backup operations must produce backup and journal files on a disk (JNLDEV$: in these examples) different from the disk or disks containing the VDE library and the VDE database. Because the backup files are on disk, they are immediately available for restoring the VDE database after a failure. However, the backup disk must itself be backed up to tape (using the OpenVMS BACKUP utility) on a regular basis.

The following example illustrates the output that you will get when you back up your database:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_BACKUP FULL_NOVERIFY
    The backup operation will be performed using the following
    device and directory specifications:

    The location of the database to back up is LIBDEV$:[PROJROOT]
    The database will be backed up to JNLDEV$:[PROJ.BACKUP]
    Existing backup files on JNLDEV$:[PROJ.BACKUP] will be renamed
        to JNLDEV$:[PROJ.BACKUP_OLD] before the backup starts
    The definition of VDE$JOURNAL is JNLDEV$:[PROJ.AIJ]

Do you want to continue with the database backup? (Y or N) : Y

    Full backup of database without verification being performed.

    Renaming database files from the backup area JNLDEV$:[PROJ.BACKUP]
           to JNLDEV$:[PROJ.BACKUP_OLD]

    Full database backup starting for LIBDEV$:[PROJROOT]VDE$DATABASE.RDB
        Backing database up to JNLDEV$:[PROJ.BACKUP]VDE$DATABASE_FULL.RBF

%RMU-I-QUIETPT, waiting for database quiet point
%RMU-I-BCKTXT_01, Thread 1 uses devices LIBDEV$:
%RMU-I-BCKTXT_08, Thread 1 was assigned file LIBDEV$:[PROJROOT]VDE$DATABASE.RDA;1
%RMU-I-BCKTXT_00, Backed up root file LIBDEV$:[PROJROOT]VDE$DATABASE.RDB;1
%RMU-I-BCKTXT_02, Full backup of storage area LIBDEV$:[PROJROOT]VDE$DATABASE.RDA;1
%RMU-I-BCKTXT_02, Full backup of storage area LIBDEV$:[PROJROOT]VDE$DATABASE.RDA;1
%RMU-I-BCKTXT_04,     ignored 2 space management pages
%RMU-I-BCKTXT_05,     backed up 10 inventory pages
%RMU-I-BCKTXT_06,     backed up 520 logical area bitmap pages
%RMU-I-BCKTXT_07,     backed up 1270 data pages

    Full database backup completed

    Backup of journal file starting for LIBDEV$:[PROJROOT]VDE$DATABASE.RDB
        Backing journal file up to JNLDEV$:[PROJ.BACKUP]VDE$DATABASE_FULL.AIJ

%RMU-I-LOGCREBCK, created backup file
                        JNLDEV$:[PROJ.BACKUP]VDE$DATABASE_FULL.AIJ;1

%RMU-I-LOGBCKAIJ, backing up AIJ file
                        JNLDEV$:[PROJ.AIJ]VDE$DATABASE.AIJ;1
%RMU-I-LOGAIJBCK, backed up 2 committed transactions at 11:14:14.93

    Backup of journal file completed
```

7.4.7 Restoring the VDE Database

If a failure has corrupted or damaged your VDE database, you must restore the database before your VDE library becomes usable again. As described in Section 7.4.2, the backup files needed to restore the database are found in the JNLDEV$:[PROJ.BACKUP] directory and the after-image journal file that records all transactions since the last backup is found in the JNLDEV$:[PROJ.AIJ] directory. The present section also

assumes that you have set up the command procedure VDE_DATABASE_RESTORE.COM in directory JNLDEV$:[PROJ.TOOL] as described in Section 7.4.5. Given that, you can restore the VDE database and recover all journalled transactions by entering the following DCL command:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_RESTORE
```

This command procedure runs the RMU utility to first restore the VDE database and to then recover all journalled transactions. After this procedure has run, you should have a fully functional and completely up-to-date VDE database. You must have the OpenVMS SYSPRV privilege to perform these steps.

If this procedure cannot restore the database, you should determine why. If the reason is that the BACKUP directory is not correctly set up, you may have to restore the BACKUP directory from the OLD_BACKUP directory or from a backup tape. If you do so, you must be sure that you do not lose any after-image journal files. You need all journal files used to log transactions since the backup files you are using were created and these journal files must be applied to the database in order by date to avoid loss of transactions. After setting up the BACKUP directory, you should be able to rerun the command procedure. If all else fails, you may have to restore the database using the RMU utility directly. See the *VAX Rdb/VMS RDO and RMU Reference Manual* for detailed descriptions of the RMU/RESTORE and RMU/RECOVER commands.

By default, the VDE_DATABASE_RESTORE procedure displays the inputs and outputs of the restore operation and asks you to confirm that those are the right inputs and outputs. You must then answer YES if you want the restore operation to proceed. This example illustrates this behavior:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_RESTORE
    The restore operation will be performed using the following
    device and directory specifications:

    The location of the backup area to restore from is JNLDEV$:[PROJ.BACKUP]
    The database root file will be restored to LIBDEV$:[PROJROOT]
    The database storage area file will be restored to LIBDEV$:[PROJROOT]
    The database snapshot file will be restored to LIBDEV$:[PROJROOT]
    The definition of VDE$JOURNAL is JNLDEV$:[PROJ.AIJ]
    The current version of the database is undefined.
        An after image journal file will not be copied
        before the database restore begins.

Do you want to continue with the database restore? (Y or N) : Y
```

To suppress the request for confirmation, you can add the keyword NOQUERY to the end of the command line invoking the procedure. For example:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_RESTORE NOQUERY
```

You always need the NOQUERY keyword when you run the restore procedure in a batch job.

The following example illustrates the output that you will get when you restore your database:

```
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_RESTORE
    The restore operation will be performed using the following
    device and directory specifications:

    The location of the backup area to restore from is JNLDEV$:[PROJ.BACKUP]
    The database root file will be restored to LIBDEV$:[PROJROOT]
    The database storage area file will be restored to LIBDEV$:[PROJROOT]
    The database snapshot file will be restored to LIBDEV$:[PROJROOT]
    The definition of VDE$JOURNAL is JNLDEV$:[PROJ.AIJ]
    The current version of the database is undefined.
        An after image journal file will not be copied
        before the database restore begins.

Do you want to continue with the database restore? (Y or N) : Y

    Deleting database header files in LIBDEV$:[PROJROOT]

    Deleting database storage area files in LIBDEV$:[PROJROOT]

    Deleting database snapshot files in LIBDEV$:[PROJROOT]

    Deleting database after image journal files in JNLDEV$:[PROJ.AIJ]

    Full restore starting for JNLDEV$:[PROJ.BACKUP]VDE$DATABASE_FULL.RBF
        root file LIBDEV$:[PROJROOT]VDE$DATABASE.RDB
        storage file LIBDEV$:[PROJROOT]VDE$DATABASE.RDA
        snapshot file LIBDEV$:[PROJROOT]VDE$DATABASE.SNP

%RMU-I-RESTXT_04, Thread 1 uses devices LIBDEV$:
%RMU-I-RESTXT_00, Restored root file LIBDEV$:[PROJROOT]VDE$DATABASE.RDB;1
%RMU-I-LOGRESSST, restored storage area LIBDEV$:[PROJROOT]VDE$DATABASE.RDA;1
%RMU-I-LOGRESSST, restored storage area LIBDEV$:[PROJROOT]VDE$DATABASE.RDA;1
%RMU-I-RESTXT_05,     rebuilt 2 space management pages
%RMU-I-RESTXT_06,     restored 10 inventory pages
%RMU-I-RESTXT_07,     rebuilt 520 logical area bitmap pages
%RMU-I-RESTXT_08,     restored 1270 data pages
%RMU-I-RESTXT_01, Initialized snapshot file LIBDEV$:[PROJROOT]VDE$DATABASE.SNP;1
%RMU-I-LOGINIFIL,     contains 1000 pages, each page is 2 blocks long

    Full restore completed

   Recover starting for LIBDEV$:[PROJROOT]VDE$DATABASE.RDB
        using JNLDEV$:[PROJ.BACKUP]VDE$DATABASE_FULL.AIJ


%RMU-I-LOGRECDB, recovering database file LIBDEV$:[PROJROOT]VDE$DATABASE.RDB;1
%RMU-W-NOTRANAPP, no transactions in this journal were applied

    Recover completed
```

7.4.8 Recovering from a Lost Journal File

If the after-image journal file for your VDE database becomes unavailable due to a disk failure or other cause, nobody will be able to use the database and VDE operations will fail. To recover from this condition, you must close, alter, back up, and restore the database. The following sequence of commands will perform the necessary operations:

```
$ RMU/CLOSE/ABORT=DELPRC LIBDEV$:[PROJ]VDE$DATABASE
$ RMU/OPEN  LIBDEV$:[PROJ]VDE$DATABASE
$ RMU/CLOSE LIBDEV$:[PROJ]VDE$DATABASE
$ ! RMU/ALTER for versions of Rdb before V6.0...
$ RMU/ALTER LIBDEV$:[PROJ]VDE$DATABASE
RdbALTER> DEPOSIT ROOT AIJ_FILENAME = ""
RdbALTER> COMMIT
RdbALTER> EXIT
$ ! RMU/SET for Rdb version V6.0 and later...
$ RMU/SET AFTER_JOURNAL LIBDEV$:[PROJ]VDE$DATABASE/DISABLE
$ !
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_BACKUP FULL_NOVERIFY
$ @JNLDEV$:[PROJ.TOOL]VDE_DATABASE_RESTORE
```

The first RMU/CLOSE command closes the database; it deletes all user processes attached to the database and denies access to new users. Any transactions the deleted processes had in progress are lost. The RMU/OPEN command rolls back any such transactions, thus making the database consistent. The second RMU/CLOSE command closes the database so that new users cannot access it. The RMU/ALTER command prepares the database to be altered. The DEPOSIT ROOT AIJ_FILENAME = "" command sets the database root file to have after-image journaling turned off. This will permit the database to be backed up. The COMMIT command causes the DEPOSIT command to be committed. The EXIT command then exits from the RMU/ALTER utility.

The next command backs up the database (without verifying it since verification can be very time-consuming) and the last command restores the database from the backup file. Restoring the database creates a new after-image journal file, so it should only be done when the failed disk is available again or a new disk is substituted. After this is done, the database should be fully functional.

You must have the SYSPRV privilege to perform this recovery procedure.

If you want to place the after-image journal file on a different disk than the original location, you need to specify the new journal file immediately after performing the above recovery procedure. To do so, use the VDE MODIFY LIBRARY command:

```
$ VDE MODIFY LIBRARY/JOURNAL=NEWDEV$:[PROJ.AIJ]
```

This operation should be done before any operations are done that update the VDE database.

Note that the after-image journal file becomes unavailable if the disk holding it runs out of space. To prevent this situation, the journal file should be placed on a disk with ample space. However, if the journal file still runs out of space, you should close, open, close, and alter the database as just described, and then delete the journal file. After that, you can back up and restore the database to make it functional again. The restore operation will create a new, initially empty journal file.

7.4.9 Associated Rdb commands

The following is the Rdb V6.0 command used to rename a particular journalling file. The version number is required.

```
$ ! Rdb V6.0 journaling  command...
$ RMU/SET AFTER_JOURNAL LIBDEV$:[PROJ]VDE$DATABASE -
_$   /ALTER=(name=aij,FILE=filename.aij;0)
```

Use the Rdb V6.0 ‘‘RMU/SHOW AFTER_JOURNAL databasename’’ command to display the current journal file(s) and the current state of journaling on the database.

If the journal is inaccessable, use the following sequence to create a new journal file:

```
$ ! Rdb V6.0 journaling  command...
$ RMU/SET AFTER_JOURNAL LIBDEV$:[PROJ]VDE$DATABASE -
_$   /DISABLE
$ RMU/SET AFTER_JOURNAL LIBDEV$:[PROJ]VDE$DATABASE -
_$   /DROP=(name=aij)
$ RMU/SET AFTER_JOURNAL LIBDEV$:[PROJ]VDE$DATABASE -
_$   /ADD=(name=aij,FILE=filename.aij;0)
$ RMU/SET AFTER_JOURNAL LIBDEV$:[PROJ]VDE$DATABASE -
_$   /ENABLE
```

The following commands are used to relocate various Rdb component files associated with the VDE Rdb database:

```
$  RMU/ALTER LIBDEV$:[PROJ]VDE$DATABASE
RdbALTER> deposit file RDB$SYSTEM SNAPSHOT -
_RdbALTER> SPECIFICATION="LIBDEV_SNAP$:[VDE$ROOT.DB_SNAPSHOT]VDE$DATABASE.SNP;1"
RdbALTER> deposit file RDB$SYSTEM -
_RdbALTER> SPECIFICATION="LIBDEV_AREA$:[VDE$ROOT.DB_STORAGE]VDE$DATABASE.RDA;1"
RdbALTER> deposit root specification="LIBDEV_ROOT$:[VDE$ROOT]VDE$DATABASE.RDB;1"
RdbALTER> commit
RdbALTER> ^Z
```

Note that RMU/ALTER does *not* currently support continuation characters such as the hyphen; the hyphen in the above example is used to strictly for readability.

7.4.10 Recovering Lost CMS Libraries

If the delta files (the CMS libraries) in your VDE library are damaged or lost due to a disk failure or other cause, you must recover as much of its lost contents as possible. The procedure for doing so involves initially restoring or recreating missing files or directories (consisting primarily of CMS libraries) from backup tapes, and then restoring the VDE database from the database backup and journal disk(s), and finally bringing the CMS libraries up-to-date using information from the database and the queued replacement staging area directories.

The first step is to determine how much of the VDE library is damaged. If the whole disk is unavailable (due to a head crash, for example), you must restore the whole disk from backup tapes created by the normal nightly backups. If only certain CMS libraries are damaged, you may only need to restore those CMS libraries. Either way, you should wind up with a VDE directory structure that is up-to-date as of the last backup, and thus at most a day out of date.

If the VDE database was affected by the failure, the next step is to restore and recover it as explained in Section 7.4.7. This operation restores the database from its most recent backup files and then applies the after-image journal files to the restored database, thus rolling the database forward until it is up-to-date as of the last transaction completed before the failure.

Because the VDE database is fully up-to-date, information in the database can be used to determine what module generations are missing from the restored CMS libraries (which may be up to a day out-of-date). To determine what generations are missing from the CMS libraries, you can use the following VDE command:

```
VDE> VERIFY GENERATION/SINCE=YESTERDAY [*]*.*
```

This command checks all module generations that according to the VDE database were created since midnight yesterday and determines whether each such generation is present in the corresponding CMS library. If not, VDE prints a message for the missing generation.

If only a few CMS libraries were restored, you can save time by verifying generations for only those facilities. For example:

```
VDE> VERIFY GENERATION/SINCE=YESTERDAY [FAC1]*,[FAC2]*,[FAC3]*
```

If the output of the VERIFY GENERATION command indicates that any module generations are missing from the restored CMS libraries, you can usually recover those generations from the VDE staging area directories. The VDE staging area directories contain the original source file for each generation created by a replacement. Missing generations can therefore be restored to a CMS library from those source files. To do so, use the VERIFY GENERATION command with the /RECOVER qualifier. For example:

```
VDE> VERIFY GENERATION/SINCE=YESTERDAY/RECOVER [*]*.*
```

This command prints a message for each missing generation, but it also locates the replacement staging area directory containing the file that created the generation. VDE uses that file to insert the missing generation into the CMS library. As a result, all CMS libraries should be fully up-to-date and consistent with the VDE database after this command completes. However, if some generation cannot be recovered, perhaps because the staging area directory has been deleted or because the generation was created without using staging area directories, then VDE prints a message to that effect and the generation is not recovered.

Keep in mind that not all module generations are stored in staging area directories. Only queued replacements use staging areas; immediate replacements do not. Also, generations created with the /INPUT or /DIRECTORY qualifier to the CREATE MODULE command are not stored in staging area directories. Because these qualifiers require VDE privileges and are normally used only during initial set-up of the VDE library, recovery of such generations is usually not an issue. However, it is generally wise to always use queued replacements if recovery of lost generations is a concern for your VDE library.

In those cases where a missing generation cannot be recovered from its staging area directory but is available from a file elsewhere, you can recover that generation with a separate VERIFY GENERATION command that gives an explicit file location for the generation:

```
VDE> VERIFY GENERATION [FAC1]FOO.C -
_VDE> /GENERATION=5A2/RECOVER=DSK$:[MYDIR]FOO.C
```

This command recovers generation 5A2 of module FOO.C in facility FAC1 from file DSK$:[MYDIR]FOO.C. If you use this form of the command to recover multiple generations for the same module, you must recover the generations in the same order as they were originally created.

If you are unable to recover a missing generation, the VDE database will be inconsistent with the affected CMS library. This is highly undesirable because it means that the missing generation (which is likely to be the latest generation of at least one stream) cannot be fetched or reserved. To correct the problem, you must do one of two things. One possibility is to get the creator of the missing generation to recreate it so you can recover it as just described. The other possibility is that you recover the missing generation using a dummy input file and then delete that generation using the VDE DELETE GENERATION command; in this case, the missing generation is permanently lost but the database becomes consistent with the CMS libriary.

7.4.11 Moving to Larger CMS Disks

This section describes how to move the CMS library to different—and usually larger—disks. The use of a shadow set for the CMS library volume, and the use of VDE in a cluster, is assumed.

- Send out a warning to the users to indicate that Rdb and the CMS libraries will be unavailable.
- Enable full privileges with the command SET PROCESS/PRIVILEGE=ALL
- Shut down all versions of Rdb running on the current system. This may involve procedures such as RMONSTOP, RMONSTOP41, RMONSTOP60, etc., as appropriate.
- If it is in use, modify MSCPMOUNT to remove shadow set mount including the specifications of the source disk(s), and remove any reference(s) to the target disk(s).
- Dismount all members of the CMS shadow set, and dismount all of the target disk volume(s).
- If MSCPMOUNT is in use, wait for fifteen minutes, or locate and remove any and all running MSCPMOUNT jobs that might be lurking anywhere on any of the cluster members.
- Set the hardware write-lock on the source disk(s).
- Mount the source and target disk privately.
- Perform a disk-to-disk BACKUP, with verification.
- Dismount the source and target disks.

- If MSCPMOUNT is in use, modify MSCPMOUNT to include the new disk volumes using the original shadow set name—do not change the name of the shadow set. Invoke copies of MSCPMOUNT on one or more nodes, as appropriate.
- Perform a cluster-wide MOUNT of the target disk—this disk is the first or root volume of the shadow set—and then MOUNT additional members cluster-wide as appropriate. When additional volumes are mounted, a shadow-copy operate will be initiated.
- Start Rdb on all cluster nodes using RMONSTART, RMONSTART60, RMONSTART41, etc., as approrpriate.
- Test the access to the CMS library from within VDE.

Note: If you need to remount the source volume of the above copy operation with /SYSTEM or /CLUSTER, you must release the hardware write-lock, mount the volume privately, use the SET VOLUME/LABEL command to alter the volume label, dismount the disk, re-enable the write-lock, and finally mount the volume /SYSTEM or /CLUSTER—duplicate volume labels are not allowed on volumes mounted /SYSTEM or /CLUSTER.

7.4.12 Moving to Larger Rdb Disks

This section describes how to relocate VDE Rdb database files to other disks, how to deal with device name changes, and how to reset the filenames of the database root, storage area, and snapshot area should the filename(s) stored in the database root file not match the filename(s) of the actual files.

```
$ RMU/CLOSE/CLUSTER/ABORT=DELPRC/WAIT -
_$   DISK$OLDROOTDISK:[TOOLS.DBROOT]VDE$DATABASE.RDB
$ RMU/MOVE_AREA/NOAREA/NOONLINE/LOG -
_$   /dir=DISK$NEWDATADISK:[VDE.TOOLS] -
_$   DISK$OLDROOTDISK:[TOOLS.DBROOT]VDE$DATABASE.RDB -
_$   /after=DISK$NEWAIJDISK:[VDE.TOOLS]VDE$DATABASE.AIJ;1 -
_$   /root=DISK$NEWROOTDISK:[VDE.TOOLS]VDE$DATABASE.RDB;1 -
_$   /snap=file=DISK$NEWSNPDISK:[VDE.TOOLS]VDE$DATABASE.SNP;1
```

This sequence performs a ‘‘controlled’’ relocation of all areas of the database (specification of /NOAREA indicates all areas are to be moved), the after-image journal, the database root, and the snapshot area, to the specified new area. The above commands will clean up the old area, as well.

Once the databases have been moved, be sure to update VDE$SYSTARTUP.COM, and any system device MOUNT procedures, to reflect the new location of the databases. Also perform a full database RMU/BACKUP.

If is also possible for this situation to arise unintentionally, should some disks get shuffled around. Should the databases be moved unintentionally, VDE can produce RDMS$_FILACCERR, RDMS$_BADROOTMATCH, and RDB$_DB_CORRUPT errors. As part of resolving this, you will want to determine why these files are no longer under the expected filenames—device name changes are one potential cause.

```
$ VDE
VDE> SET LIBRARY VMS
%RDB-F-SYS_REQUEST, error from system services request
-RDMS-F-FILACCERR, error opening storage area file $11$DUA1010:[VMS.DATABASE]VDE$DATABASE.SNP;1
-RMS-F-DEV, error in device name or inappropriate device type for operation
VDE> CTRL/Z
$

$ VDE
VDE> SET LIBRARY VMS
%RDB-F-DB_CORRUPT, database filename is corrupt
-RDMS-F-BADROOTMATCH, root file "$22$DUA1010:[VMS.DATABASE]VDE$DATABASE.RDB;1" no longer has its original
VDE> CTRL/Z
$

$
$ RMU/ALTER $22$DUA1010:[VMS.DATABASE]VDE$DATABASE.RDB
RdbALTER>  dep file RDB$SYSTEM snapshot speci="$22$DUA1010:[VMS.DATABASE]VDE$DATABASE.SNP;1"
RdbALTER>  dep file RDB$SYSTEM speci="$22$DUA1010:[VMS.DATABASE]VDE$DATABASE.RDA;1"
RdbALTER>  dep root specif="$22$DUA1010:[VMS.DATABASE]VDE$DATABASE.RDB;1"
RdbALTER>  commit
RdbALTER>  CTRL/Z
$
```

Once the database has been brought back online or has been relocated, make *certain* to perform a full database RMU/BACKUP of the database.

7.5 Purging the Staging Area Directories

The staging area directories will grow over time and must be manually cleaned. This operation is performed at site-specific periodic intervals—the intervals are determined by the size of the staging device and the frequency and size of replacements, and by the frequency of the database backups. This section describes the steps used to purge old replacements. For information on utilitizing the staging area directories during VDE database recovery operations, see Section 7.4.10.

The staging area directories can be located with the command:

```
VSC> SHOW LOCATION/STAGING
```

This command displays the staging (sub)directory containing all staging area directories.

Underneath the staging subdirectory exists three levels of subdirectories. The files associated with a replacement are located in the lowest two levels of the staging heirarchy—all files associated with a particular replacement are located below a unique subdirectory [.VDE$REP_*], in facility-specific subdirectories. Located one level above the [.VDE$REP_*] replacement subdirectory is a staging area subdirectory in the form [.VDE$STG_*]. The staging area subdirectory exists solely to reduce the number of files in any single directory. For further details on the stating area directory structure, see Section 6.3.2.

An example of the full specification of a staging directory follows:

```
VSC$STAGEDISK:[VMS.STAGE.VDE$STG_130.VDE$REP_13066.ERF]
```

The names of the [.VDE$REP_*] replacement subdirectories are built from a predefined prefix with the five decimal digit replacement number appended. Replacement numbers are allocated in ascending order, and are not reused unless the database is recreated.

The names of the [.VDE$STG_*] staging area subdirectories are comrpised of a predefined prefix with all but the last two decimal digits of the replacement number appended. This means each of staging area subdirectories holds one hundred replacement subdirectory files. Staging area subdirectories—once filled – are not reused until and unless the entire database is recreated.

To perform the cleanup, issue the command

```
VSC> SHOW REPLACEMENT/FULL
```

And look for the replacement number associated with the oldest replacement displayed. This is typically the first entry displayed.

Using the oldest replacement identifier, delete all [.VDE$STG_*] staging area subdirectories where the first three digits are less than the observed replacement number. For example, if the oldest outstanding replacement number is ‘‘13054’’, all subdirectories and all files located under staging area subdirectories [.VDE$STG_129...], [.VDE$STG_128...], [.VDE$STG_127...], etc., can be deleted. Do *not* delete staging area subdirectories more recent than the last VDE database backup operation. See Section 7.4.10.

Failure to insure adequate space in the staging area can produce errors similar to the following:

```
%COPY-E-OPENOUT, error opening VSC$STAGEDISK:[VMS.STAGE.VDE$STG_131.
VDE$REP_13163.DECW$DEVICE]DECWINDOWS000.E; as output
-RMS-F-FUL, device full (insufficient space for allocation)
%COPY-W-NOTCOPIED, dev:[dir]DECWINDOWS000.E;1 not copied
%VDE-E-SCRPMODFAIL, STAGING script for module
[DECW$DEVICE]DECWINDOWS000.E terminated with errors
```

The above error occured on the following command:

```
VSC> REPLACE DECWINDOWS000.E
```

7.6 Offloading Large Files

It is possible for a VDE database to exceed the capacity of a volume. The following section describes the steps required to offload the larger files from one of the VDE libraries.

The following command can be used to locate the larger files present in the VDE libraries. In the case of OpenVMS, these files are typically savesets from contributing projects, are typically large.

Note: The steps outlined in the following procedure apply *only* to files referenced via marker files. Further, the steps outlined below should only be performed with the assistance and under the review of individual(s) familiar with the status of the various streams present in the library.

Marker files are typically large binary, image, saveset, or library files, and as such are not (and should not be) stored directly in the CMS library, as this reduces the overhead involved when there are typically rather large changes between one module generation and the next. Use the SHOW MODULE/FULL command to determine if a particular module is a marker file. Use the SHOW DIRECTORY/DELTA command to determine the

current (default) location of the delta directory—specifically the location referenced by the applicable VDE$MARKER symbol.

Note: Recent versions of CMS have added support for the storage of binary files, but VDE does not currently utilize this support.

The following command can be used to locate the larger file(s) present in the local VDE database in the ‘‘VMS$’’ library:

```
$  DIRECTORY/SIZE/SELECT=SIZE=MINIMUM=500 VMS$:[000000...]*.*.*
```

Based on the files located by above command, a list of the larger files was generated, and the following command procedure was then created:

```
$ Set NoOn
$ dir/col=1/siz/out=sys$scratch:masd_listing1.tmp -
    vms$:[ddtm.src]ddtmkit.bck
$ dir/col=1/siz/out=sys$scratch:masd_listing2.tmp -
    vms$:[decw$buildtool.src]decw$buildtool.sav,decw$motif%%%.%
$ dir/col=1/siz/out=sys$scratch:masd_listing3.tmp -
    vms$:[decw$common.src]decwindows%%%.a
$ dir/col=1/siz/out=sys$scratch:masd_listing4.tmp -
    vms$:[decw$device.src]decwindows000.b,.c,.d,.e,.f
$ dir/col=1/siz/out=sys$scratch:masd_listing5.tmp -
    vms$:[decw$fonts.src]decwindows000.c,.d,.e
$ dir/col=1/siz/out=sys$scratch:masd_listing6.tmp -
    vms$:[decw$server.src]decwindows000.b,.c,.q
$ dir/col=1/siz/out=sys$scratch:masd_listing7.tmp -
    vms$:[ipc.src]ipckit.bck
$ dir/col=1/siz/out=sys$scratch:masd_listing8.tmp -
    vms$:[lm.src]lmkit.bck
$ dir/col=1/siz/out=sys$scratch:masd_listing9.tmp -
    vms$:[starlet.src]vic_starlet.obj
$ dir/col=1/siz/out=sys$scratch:masd_listinga.tmp -
    vms$:[syslib.src]*.*
$ dir/col=1/siz/out=sys$scratch:masd_listingb.tmp -
    vms$:[tpssda.src]tpssdakit.bck
$
$ Copy sys$scratch:masd_listing%.tmp sys$scratch:masd_listing.lis
```

Based on the above command procedure, the following command is issued for each module located:

```
VSC> SHOW GENERATION/TREE [DDTM]DDTMKIT.BCK
```

Based on the above generation listing and the directory listing, you must now translate each module generation to the associated file version number—this information allows you to remove the appropriate module generations. Each generation of the module must be checked against the directory listing and the generation tree. The following two commands must be issued—the first command is issued once per module, and the second is issued once for each module generation:

```
VSC> SHOW MODULE [DDTM]DDTMKIT.BCK/FULL
VSC> FETCH/OUT=TT:/GENERATION=n [DDTM]DDTMKIT.BCK
```

The above commands are used to determine the generation-to-version mapping, and to ensure that the module uses a marker file. Based on the directory listing, on the generation-to-version translation, and on the knowledge of which versions are in streams that are unlikely to be immediately required, a review of the files that can be offloaded can be conducted. Based on the results of the review, a command procedure similar to the following can be created:

```
$ SET NOON
$
$! Ignore any higher-version warnings..
$
$ READ/PROMPT="^Z TO ABORT, <RET> TO CONTINUE " -
    /ERROR=DONE/END_OF_FILE=DONE SYS$COMMAND YESNO
$
$ CREATE/DIRECTORY VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DDTM]
$ COPY VMS$:[DDTM.SRC]DDTMKIT.BCK;54 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DDTM]
$ COPY VMS$:[DDTM.SRC]DDTMKIT.BCK;53 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DDTM]
$ CREATE/DIRECT VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DECW$DEVICE]
$ COPY VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.B;18 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DECW$DEVICE]
$ COPY VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.C;17 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DECW$DEVICE]
$ COPY VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.D;18 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DECW$DEVICE]
$ COPY VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.E;13 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DECW$DEVICE]
$ COPY VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.E;11 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DECW$DEVICE]
$ COPY VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.F;11 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.DECW$DEVICE]
$ CREATE/DIRECT VDE$RESD1$:[941002_VMS_MASD_CLEANUP.IPC]
$ COPY VMS$:[IPC.SRC]IPCKIT.BCK;47 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.IPC]
$ CREATE/DIRECT VDE$RESD1$:[941002_VMS_MASD_CLEANUP.LM]
$ COPY VMS$:[LM.SRC]LMKIT.BCK;47 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.LM]
$ CREATE/DIRECT VDE$RESD1$:[941002_VMS_MASD_CLEANUP.STARLET]
$ COPY VMS$:[STARLET.SRC]VIC_STARLET.OBJ;32 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.STARLET]
$ COPY VMS$:[STARLET.SRC]VIC_STARLET.OBJ;29 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.STARLET]
$ CREATE/DIRECT VDE$RESD1$:[941002_VMS_MASD_CLEANUP.SYSLIB]
$ COPY VMS$:[SYSLIB.SRC]VAXCDEF.TLB;30 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.SYSLIB]
$ COPY VMS$:[SYSLIB.SRC]VAXCDEF.TLB;28 -
    VDE$RESD1$:[941002_VMS_MASD_CLEANUP.SYSLIB]
$
$ READ/PROMPT="^Z TO ABORT, <RET> TO CONTINUE " -
    /ERROR=DONE/END_OF_FILE=DONE SYS$COMMAND YESNO
$
$ DELETE VMS$:[DDTM.SRC]DDTMKIT.BCK;54
$ DELETE VMS$:[DDTM.SRC]DDTMKIT.BCK;53
$ DELETE VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.B;18
$ DELETE VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.C;17
$ DELETE VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.D;18
$ DELETE VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.E;13
$ DELETE VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.E;11
$ DELETE VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.F;11
$ DELETE VMS$:[IPC.SRC]IPCKIT.BCK;47
$ DELETE VMS$:[LM.SRC]LMKIT.BCK;47
$ DELETE VMS$:[STARLET.SRC]VIC_STARLET.OBJ;32
$ DELETE VMS$:[STARLET.SRC]VIC_STARLET.OBJ;29
$ DELETE VMS$:[SYSLIB.SRC]VAXCDEF.TLB;30
$ DELETE VMS$:[SYSLIB.SRC]VAXCDEF.TLB;28
$
$ READ/PROMPT="^Z TO ABORT, <RET> TO CONTINUE " -
    /ERROR=DONE/END_OF_FILE=DONE SYS$COMMAND YESNO
$
$ COPY SYS$SCRATCH:Q.Q VMS$:[DDTM.SRC]DDTMKIT.BCK;54
$ COPY SYS$SCRATCH:Q.Q VMS$:[DDTM.SRC]DDTMKIT.BCK;53
$ COPY SYS$SCRATCH:Q.Q VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.B;18
$ COPY SYS$SCRATCH:Q.Q VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.C;17
$ COPY SYS$SCRATCH:Q.Q VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.D;18
$ COPY SYS$SCRATCH:Q.Q VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.E;13
$ COPY SYS$SCRATCH:Q.Q VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.E;11
```

```
$ COPY SYS$SCRATCH:Q.Q VMS$:[DECW$DEVICE.SRC]DECWINDOWS000.F;11
$ COPY SYS$SCRATCH:Q.Q VMS$:[IPC.SRC]IPCKIT.BCK;47
$ COPY SYS$SCRATCH:Q.Q VMS$:[LM.SRC]LMKIT.BCK;47
$ COPY SYS$SCRATCH:Q.Q VMS$:[STARLET.SRC]VIC_STARLET.OBJ;32
$ COPY SYS$SCRATCH:Q.Q VMS$:[STARLET.SRC]VIC_STARLET.OBJ;29
$ COPY SYS$SCRATCH:Q.Q VMS$:[SYSLIB.SRC]VAXCDEF.TLB;30
$ COPY SYS$SCRATCH:Q.Q VMS$:[SYSLIB.SRC]VAXCDEF.TLB;28
$
$ EXIT
$
$DONE:
$ EXIT
```

The above command procedure is used to substitute a stub file—an example ‘‘Q.Q’’ stub file follows—for the original version of the file.

```
 ***  this file was archived as of Oct 2, 1994 ***
 ***  Contact [name], at [email-address]       ***
```

The end result of the above commands and command procedures is the relocation of the larger modules—those that are referenced via marker files—and out of the VDE library. Should the offloaded generation of the file be reserved or fetched for any reason, the ‘‘archived’’ notice will be returned. The archived file must be manually located and retrieved from online or offline storage. (It is typically good practice to indicate the particular location of this storage in the stub file.)

A future version of VDE may include the capability to ‘‘tie off’’ streams and offload modules referenced by marker files in an easier and cleaner fashion.

7.7 Defragmenting the CMS Library

This section contains a simple set of procedures used to defragment the CMS portion of the VDE database using the OpenVMS BACKUP utility, using the steps appropriate for a VDE database located on a shadow set.

The following example assumes the VDE database is at ddcu:[vde]VDE$DATABASE.RDB, and the CMS libraries are located on the 11DSA1: shadow set device, a shadow set comprised of 11DUA10: and 11DUA11:. 11DUA100: is assumed to be a scratch disk equivilent to 11DUA10: and 11DUA11:.

```
$ RMU/CLOSE/CLUSTER/ABORT=DELPRC ddcu:[vde]VDE$DATABASE.RDB
$ DISMOUNT/CLUSTER $11$DSA1:
$ MOUNT/OVER=(SHADOW,IDENT)/NOWRITE $11$DUA10:
$ MOUNT/OVER=(SHADOW,IDENT)/NOWRITE $11$DUA11:
$ MOUNT/FOREIGN $11$DUA100:
$ BACKUP/IMAGE $11$DUA10: $11$DUA100:/VERIFY/INITIALIZE
$ DISMOUNT/NOUNLOAD $11$DUA10:
$ MOUNT/FOREIGN $11$DUA10:
$ BACKUP/PHYSICAL $11$DUA100: $11$DUA10:
$ DISMOUNT/NOUNLOAD $11$DUA11:
$ MOUNT/FOREIGN $11$DUA11:
$ BACKUP/PHYSICAL $11$DUA100: $11$DUA11:
$ MOUNT $11$DSA1:/SHADOW=($11$DUA10:,$11$DUA11:) -
_$ VMSCMSMASTER DISK$VMSCMSMASTER
```

The above steps quickly bring the defragmented database shadow set volumes into synchronization. While the shadow set copy operation could also be used to resynchronize the defragmented volume, shadowing merge

performance is not tuned for this particular (infrequent) situation; it is far slower than the double BACKUP/PHYSICAL shown above.

7.8 Relocating Rdb Components

It is normally best to use the Rdb RMU/BACKUP and RMU/RESORE commands to relocate a database. As this is not always possible, the following commands are used to close and then to relocate various individual Rdb component files associated with the VDE Rdb database:

```
$ DEFINE/SYSTEM/EXECUTIVE VDE$OFFLINE "offline for move"
$ RMU/CLOSE/CLUSTER/ABORT=DELPRC/WAIT -
_$ VDE$:[ROOTDIR]VDE$DATABASE
$  ! BACKUP or COPY the database files to the new location
$  RMU/ALTER LIBDEV$:[PROJ]VDE$DATABASE.RDB
RdbALTER> deposit file RDB$SYSTEM SNAPSHOT -
_RdbALTER> SPECIFICATION="LIBDEV_SNAP$:[VDE$ROOT.DB_SNAPSHOT]VDE$DATABASE.SNP;1"
RdbALTER> deposit file RDB$SYSTEM -
_RdbALTER> SPECIFICATION="LIBDEV_AREA$:[VDE$ROOT.DB_STORAGE]VDE$DATABASE.RDA;1"
RdbALTER> deposit root specification="LIBDEV_ROOT$:[VDE$ROOT]VDE$DATABASE.RDB;1"
RdbALTER> commit
RdbALTER> ^Z
$  ! Now adjust any VDE$SYSTARTUP.COM definitions associated
$  ! with the database to reflect the new location, and also
$  ! adjust the live logical name definitions on all cluster
$  ! members as appropriate.
$ DEASSIGN/SYSTEM/EXECUTIVE VDE$OFFLINE
```

Note that RMU/ALTER does *not* currently support continuation characters such as the hyphen; the hyphen in the above example is used to strictly for readability.

7.9 Recreating Rdb Components

As part of the process described in Section 7.8, you may sometimes need to recreate the snapshot file and the after-image journal (AIJ) file, via the following commands:

```
$  RMU/REPAIR/AREA=RDB$SYSTEM/INITIALIZE=SNAPSHOT  -
_$  VDE$:[ROOTDIR]VDE$DATABASE.RDB
$  RMU/SET AFTER_JOURNAL VDE$:[ROOTDIR]VDE$DATABASE.RDB;1 -
_$  /DISABLE
$  RMU/SET AFTER_JOURNAL VDE$:[ROOTDIR]VDE$DATABASE.RDB;1 -
_$  /DROP=(NAME=RDB$JOURNAL) -
$  RMU/SET AFTER_JOURNAL VDE$:[ROOTDIR]VDE$DATABASE.RDB;1 -
_$  /ADD=(NAME=RDB$JOURNAL, -
_$  FILE=LIBDEV_SNAP$:[VDE$ROOT.DB_JOURNAL]VDE$DATABASE.AIJ)
$  RMU/SET AFTER_JOURNAL VDE$:[ROOTDIR]VDE$DATABASE.RDB;1 -
_$  /ENABLE
```

Also, the following command:

```
$  RMU/REPAIR/ABM VDE$:[ROOTDIR]VDE$DATABASE.RDB;1
```

can be of interest if the RMU/VERIFY command detects inconsistencies among the ABM and SPAM pages in the database.

The AIJTERMINATE error can arise when the journal file is inaccessable, or when it is corrupted. Typically, an AIJTERMINATE error is generated whenever any operation attempts to access a database after a journal file problem has been detected by Rdb.

```
$  VDE
$  SET LIBRARY libname
%RDMS-F-AIJTERMINATE, inaccessible AIJ file forced image exit to protect database
```

The following sequence of commands can be used to recreate the journal file:

```
$  RMUE/SHOW AFTER VDE$DATABASE.RDB
JOURNAL IS ENABLED -
    RESERVE 1 -
    ALLOCATION IS 512 -
    EXTENT IS 512 -
    OVERWRITE IS DISABLED -
    SHUTDOWN_TIMEOUT IS 60 -
    NOTIFY IS DISABLED -
    BACKUPS ARE MANUAL -
    CACHE IS DISABLED
ADD JOURNAL RDB$JOURNAL -
    FILE VDE$JOURNAL:VDE$DATABASE
$  RMUE/SET AFTER VDE$DATABASE.RDB/DISABLE
$  RMUE/SET AFTER VDE$DATABASE.RDB/DROP=NAME=RDB$JOURNAL
$  RMUE/SET AFTER VDE$DATABASE.RDB -
_$  /ADD=(NAME=RDB$JOURNAL,FILE=VDE$DATABASE.AIJ)
%RMU-W-DOENBLAIJ, after-image journaling must be enabled to ensure recovery
$   RMUE/SET AFTER VDE$DATABASE.RDB/ENABLE
%RMU-W-DOFULLBCK, full database backup should be done to ensure future recovery
$ ! backup both the journal file, and the database file, now...
```

The names and locations of the database and the journal file will typically need to be included in the above commands.

Also of interest may be the output from the command:

```
$  RMU/DUMP/HEADER/OPTIONS=DEBUG VDE$DATABASE.RDB
```

As the AIJ_STATUS value can indicate what happened with the journal file. (Seeing ''AIJ_STATUS = 1.'' indicates that the journal file could not be opened.) Also check for any RDSBUGCHK.DMP files.

8 Building your Software System

The software building capabilities described in this chapter have not been completely debugged.

This chapter contains preliminary information. Large sections of this chapter are currently under construction...

This chapter describes how to use the VDE builder utility to...

Note: The VDE utility and the VDE> are shown in this chapter. The VSC utility is intended to be used only with the OpenVMS VAX source code library.

8.1 Build Dependency Graphs

Section under construction...

Section under construction...

8.2 Central Build Algorithm

Section under construction...

8.3 Creating Build Jobs

Section under construction...

8.4 Displaying Build Jobs

Section under construction...

8.5 Running Build Jobs

Section under construction...

```
VDE> START BUILD_JOB V2-BL5"
```

Section under construction...

8.5.1 Starting Build Jobs

Section under construction...

8.5.2 Stopping Build Jobs

Section under construction...

8.5.3 Suspending Build Jobs

Section under construction...

8.5.4 Waiting for Build Jobs

Section under construction...

8.6 Scripts for Build Step Execution

Section under construction...

```
VDE> CREATE SCRIPT DEVD$:[DIR]FILENAME.VDESCRIPT
```

8.7 Performing Individual Build Steps

Section under construction...

8.7.1 Copy Steps

Section under construction... To cancel a module reservation without replacing the module, use the

8.7.2 Compile Steps

Section under construction... To cancel a module reservation without replacing the module, use the

8.7.3 Link Steps

Section under construction... To cancel a module reservation without replacing the module, use the

8.8 Initializing the Dependency Graph

Section under construction...

9 Architecture Support

This chapter provides an overview and a general description of how to use the multiple-architecture support present within VDE

9.1 Architectures in the Source Code Library Roots

VDE architecture support adds the *arch-variant*, the architecture variant. One processor-independent root is present—''VMS'' – and zero or more processor-dependent variants can be created. *Previous versions of VDE included implicit support for arch-variants, and all such versions use ''VMS'' as the name of the processor architecture independent arch-variant.*

As a specific example of adding arch-variants, one can support both OpenVMS Alpha and OpenVMS VAX development in parallel if the OpenVMS source code libraries are logically reorganized as follows:

- The VMS arch-variant directory contains the facility subdirectories for all architecture independent modules in OpenVMS. There is a CMS library within the VMS arch-variant for each facility that contains at least one architecture-independent module.
- Subdirectories within the VAX arch-variant directory contain CMS libraries for all facilities that contain at least one module specific to the VAX architecture. Likewise, subdirectories within the Alpha arch-variant directory contain CMS libraries for all facilities that contain at least one module specific to the Alpha architecture.
- If a module is architecture-independent, it is only represented in a CMS library within the VMS arch-variant. When a module becomes architecture-specific, it is turned into two modules so that a CMS element is created for it within both the VAX and the Alpha arch-variants. The CMS element within the VMS arch-variant remains since it contains the module's contents for all old development streams. In the VMS arch-variant, the stream (CMS class) in which the module was made architecture-specific is bound to a generation whose text says that the module is now architecture-specific. In the the VAX and Alpha arch-variants, this stream is bound to the initial generations of the new architecture-specific modules in those arch-variants. These new generations have the same contents as the latest generation in the VMS arch-variants before the module was made architecture-specific. Thereafter, the three modules (the VMS, VAX, and Alpha modules, all with the same module name) evolve separately as separate CMS elements.
- If a new module is specific to a single architecture, that module is only represented in a CMS library within the VAX or the Alpha arch-variant, whichever architecture applies. New modules specific to OpenVMS Alpha, for example, will only appear within the Alpha arch-variant.

The structure described here is not a proposed structure, but the actual structure as it is presently implemented.

9.2 Commands to Specify Multiple Architectures

To support multiple architectures within a source code library structure, the VDE source control system must represent the concept of an architecture in its module-name syntax and in its database. The module-name syntax we will use in this specification is ARCH:[FAC]MOD.TYP. Here ARCH is the architecture of the module: VMS for architecture-independent modules and VAX or Alpha for architecture-specific modules. The remaining fields ([FAC]MOD.TYP) have the normal meanings explained earlier in this document. The architecture field is optional; if it is omitted, a user-settable default architecture is supplied.

To set up architecture information in the source control database, the VDE utility supports the following command:

```
VDE> CREATE ARCHITECTURE VAX
```

This command creates an architecture entry in the database and assigns the architecture a name and a remark string. This command is only used when the database is initially set up (and whenever we invent additional architectures in the future). In addition, the VDE utility supports a MODIFY ARCHITECTURE command (to change the remark string), a DELETE ARCHITECTURE command (to correct mistakes in defining architectures), and a SHOW ARCHITECTURE command. The SHOW ARCHITECTURE command displays information about all defined architectures. The SHOW DEFAULTS command shows the current default architecture, along with other defaults.

A command that is of more interest to most developers is the command that establishes a default architecture for the current VSC or VDE session:

```
VSC> SET ARCHITECTURE Alpha
```

This example specifies that Alpha is the default architecture when the architecture name is omitted from module names. This command also resets the default facility to be the facility in the new architecture that has the same name as the current default facility. If there is no such facility, the default facility setting is cleared so that the default facility becomes undefined.

9.3 Creating Architecture-Specific Modules

The VSC and VDE utilities provide a qualifier to the MODIFY MODULE command that makes a module architecture-specific by creating one or more single-architecture modules from an existing architecture-independent module. This is the command syntax:

```
VDE> MODIFY MODULE/ARCHITECTURE_SPECIFIC VMS:[FACIL]FOO.BAR
```

This command makes module VMS:[FACIL]FOO.BAR architecture-specific by creating the two modules VAX:[FACIL]FOO.BAR and Alpha:[FACIL]FOO.BAR. The command fetches out the latest generation of the module for the default development stream and makes that the

initial generation for the two new modules. It also fetches out the latest generations of any successor development streams (if different) and inserts those into the new modules. The default stream and its successors are thus all defined for the two new architecture-specific modules. In addition, this command creates a new generation of the architecture-independent module in the VMS arch-variant. This generation is bound to the default stream and its successors and contains fixed text that indicates that the module is now architecture-specific. The database record for this generation contains a flag indicating that the module is obsolete because it was made architecture-specific; hence the source control system can determine that the module is obsolete without actually examining the text of the module.

The /ARCHITECTURE_SPECIFIC qualifier can optionally take an architecture name as a parameter. If this parameter is specified, the MODIFY MODULE command creates a new module for that one architecture only, and not for other architectures.

To make an architecture-specific module architecture-independent again, a developer must first insert the appropriate generations of the module into the architecture-independent module (such as VMS:[FACIL]FOO.BAR). This is done with a combination of FETCH, RESERVE, REPLACE, and INSERT GENERATION commands, and it must be done for all active streams. Merging two architecture-specific versions of a module into one architecture-independent version is thus largely a manual operation. Thereafter, the module must be marked as obsolete for each such stream in the two single-architecture modules (for example, VAX:[FACIL]FOO.BAR and Alpha:[FACIL]FOO.BAR).

To create a new module which is specific to a single architecture, a user enters the CREATE MODULE command, specifying the architecture:

```
VDE> CREATE MODULE Alpha:[FACIL]FOO.BAR
```

This example creates a module for the Alpha architecture only. If the module should be architecture-independent, the architecture must be specified as VMS, explicitly or by default.

9.4 Operation of Existing Commands

The presence of multiple architectures affects the look-up rules for all commands that accept module names as parameters. Commands that refer directly to modules behave a bit differently from commands that refer to module generations.

Commands that refer directly to modules, such as the CREATE, MODIFY, DELETE, and SHOW MODULE commands, only create or reference the module in the architecture the user specified. If the architecture is not specified explicitly on the module name, the command uses the default architecture established by the most recent SET ARCHITECTURE command. On any command where the module name may be wildcarded, the architecture part may also be wildcarded. The following command thus shows three modules if the module has forms that are both dependent and independent of architecture:

```
VDE> SHOW MODULE *:[FACIL]FOO.BAR
Module Alpha:[FACIL]FOO.BAR
Module VAX:[FACIL]FOO.BAR
Module VMS:[FACIL]FOO.BAR
```

Commands that refer to module generations need to look up a generation of a module in a given development stream. Usually the latest generation of the module in the given stream is requested. Whether such a generation exists depends on the module's architecture as well as the stream.

Given an architecture (specified explicitly or by default), a module, and a development stream, the FETCH or RESERVE command looks up the the module with the specified name in the specified architecture. If such a module is found, the command looks up the latest generation of the module in the requested development stream. If such a generation is found and is not marked as obsolete because the module was made architecture-specific, that is the generation that is fetched or reserved. Otherwise, the command prints a message saying that the module does not exist, is not defined in the given stream, or is obsolete for the given stream. If the module was made architecture-specific, the command prints the names of the architectures in which the module is defined. It is then up to the user to fetch or reserve the corresponding module for each desired architecture.

A developer who primarily works on OpenVMS Alpha code may thus set his or her default architecture to Alpha. He or she will get the instance of the module specific to OpenVMS Alpha, if there is one, else a a message indicating the module exists for the architecture VMS will be displayed. A developer whose work is not focused on a specific architecture could instead set his or her default architecture to VMS. When fetching or reserving a module, such a developer will either get the architecture-independent instance of the module or a message saying that the module is defined for architecture VAX or Alpha, whichever applies, or for both.

The REPLACE command does not look up generations in the same sense. It looks up a reservation for a specified module and development stream on the assumption that the module belongs to the architecture that the user specifies, either explicitly or by default. If there is no such reservation, the command prints an error message. The UNRESERVE command uses the same procedure.

The SHOW and MODIFY GENERATION commands only look up generations that belong to the module in the architecture the user specifies (explicitly or by default). Generations that have been made architecture-specific are marked as such in the SHOW GENERATION output. When the SHOW GENERATION command displays the list of generations that gave rise to the latest generation of the module in a specified stream, it also shows at what generation the module was converted from being architecture-independent to being architecture-specific. The INSERT GENERATION and DELETE GENERATION commands only work on the architecture that the user specifies.

9.5 Change Propagation and Fold Records

When a module is replaced into the library, the source control system may have to propagate the change from the specified development stream to its successors. Change propagation and fold records have already been explained for the one-architecture case. In the presence of multiple architectures, the following additional rules apply:

- If the stream successor chain does not cross from the architecture-independent to the architecture-specific instance of the module being replaced, change propagation and fold record generation work as described earlier in this document.
- If the development stream successor chain crosses an architecture boundary for module [FACIL]FOO.BAR (for example, if VMS:[FACIL]FOO.BAR contains the generation in stream AETNA and VAX:[FACIL]FOO.BAR and Alpha:[FACIL]FOO.BAR contain the generations in stream PHOENIX), then the source control system generates fold records saying that the change to VMS:[FACIL]FOO.BAR must be manually propagated to modules VAX:[FACIL]FOO.BAR and Alpha:[FACIL]FOO.BAR. The same is true if the change must be propagated from an architecture-independent module to a single-architecture module.
- Source changes will not be propagated automatically across an architecture boundary in Version 1 of the new source control system. In general, a module will not be made architectures-specific unless its Alpha form is different from its architecture-independent form. The VAX form is more likely to be the same as the original architecture-independent form, which would make change propagation across architectures useful, but the implementation of such change propagation is difficult and will therefore be deferred. This means that change propagation across architectures must be done manually, just as folds are. Fold records will remind developers when such change propagation is required.
- If a module is architecture-specific, then a change to the Alpha module can optionally generate a fold record that reminds the user to make the same change to the VAX module, and vice versa. In this situation, the REPLACE command will ask the user whether to create such a fold record.

In summary, while VDE does include support multiple source code architectures, there is no support automatic change propagation across architectures. VDE does support fold records for changes that span architectures.

10 The Graphical User Interfaces

VDE provides an X Windows Motif graphical user interface, as well as a World Wide Web (WWW) interface accessable via Mosaic. *This section is under construction...*

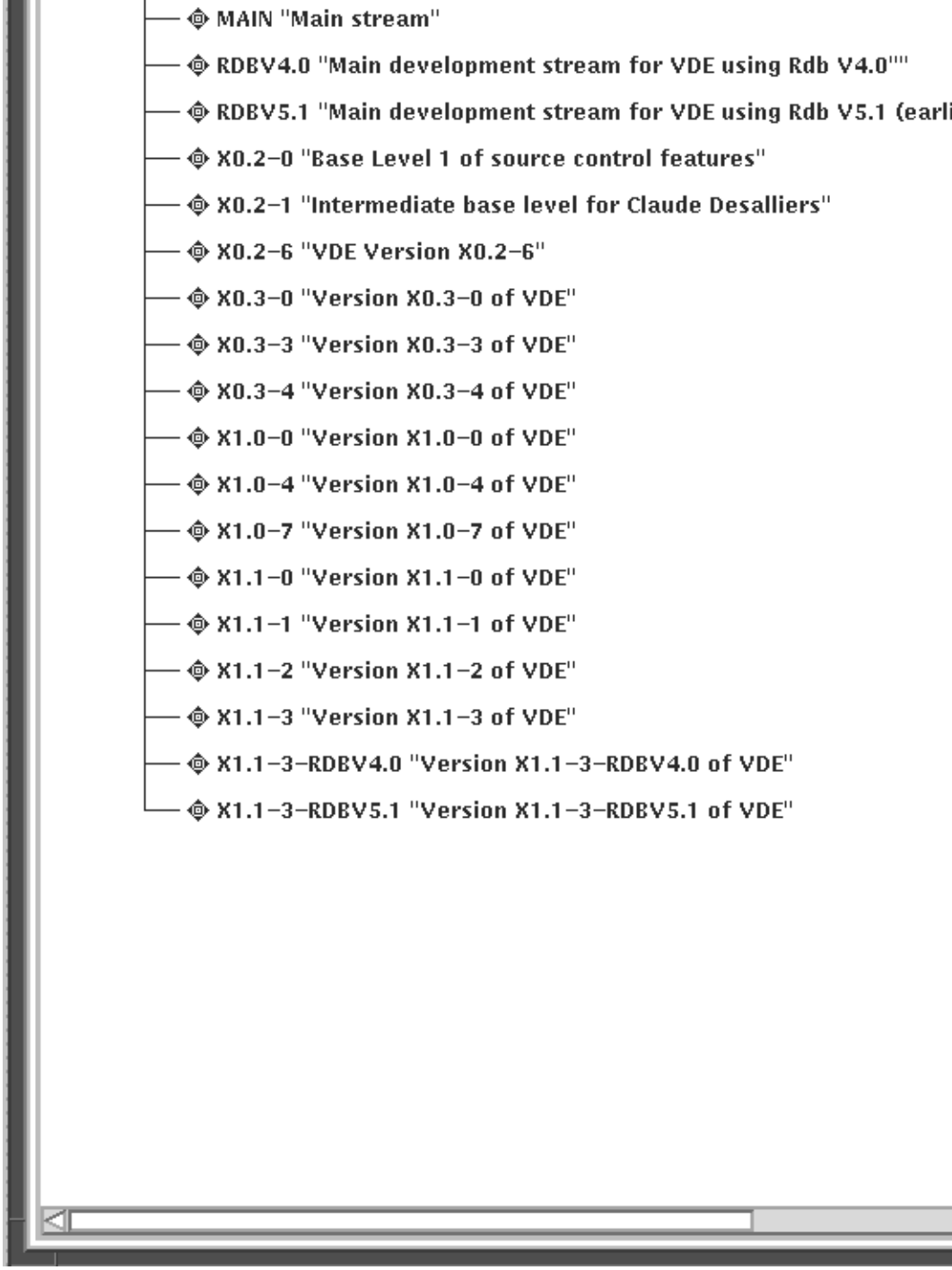
10.1 Motif Interface

This section is under construction...

10.1.1 Motif Version Information

The VDE is heirarchical; the display shows, in decending order, the VDE library, the streams within the library, the facilities within the stream, and the modules within each facility.

Figure 10–1 is an example of the facility-level display.



This is the VDE Motif interface for V1.5-0. This interface is Copyright 1997.

This section is under construction...

10.1.2 Motif Library Selection

This section is under construction...

Initial VDE library selection is based on the definition of the logical name VDE$LIBRARY. If the VDE$LIBRARY logical name is not defined, VDE searches for the file Motif data file DECW$USER_INCLUDE:VDE.DAT. An example of VDE.DAT follows:

```
#
# VDE.DAT
# VDE Motif configuration
# file created by: VDE for OpenVMS VAX Version V1.5-0
# file created at: Wed Jan  4 13:32:47 1997
# values:  DEFAULT SETTINGS
#
VDE.title:  VDE: OpenVMS Development Environment
VDE.x:   100
VDE.y:   100
VDE.remarksVisible: true
VDE.fontList:  *-*-Menu-Medium-R-Normal--*-120-*-*-P-*-ISO8859-1
VDE.libraryVDE:  VSC$ROOTDISK:[VMS.DBROOT]
VDE.libraryVDEDebug: VDED$:[VDE]
VDE.libraryVSC:  VSC$ROOTDISK:[VMS.DBROOT]
VDE.libraryVSCDebug: VDED$:[VDE]
VDE.libraryLabel:
VDE.libraryList:
VDE.sctNotesPrefix: SCT-
VDE.sctNotesNode: NODE::
VDE*borderColor: #545445450000
# end VDE.DAT
```

10.1.3 Motif Library Default Selection

This section is under construction...

VDE provides support for multiple default library selection lists. These selection lists allow a VDE user an easy way to avoid memorizing all the various ‘‘interesting’’ libraries that might be around, and they provide an easy way for a user and/or a system manager to change the default library list, and they provide a shorthand reference to individual libraries. Multiple lists of library defaults are supported and can be created for individual users, user groups, all system users, all cluster users, and for eclectic groups of users based on a shared logical name table. Note that each individual user has access to only a single list of defaults.

There are three parts to each default library specification—the library database device and directory specification, the display value shown to the user for each database, and the shorthand mnemonic for each library. These values are used to access the library on disk, for the text used to display the library, and as a nickname for a particular library, respectively.

VDE reads and uses values from logical name search lists, or from the VDE.DAT Motif resource file, as the default values for various library selection operations. The logical names used for this purpose are VDE$LIBRARY_DEFAULTS, VDE$LIBRARY_DEFAULTS_LABELS, VDE$LIBRARY_DEFAULTS_CMS, and VDE$LIBRARY_DEFAULTS_ MNEMONICS.

VDE translates the list of default libraries from the above logical names, or (if the logicals are absent) from the VDE.DAT Motif resource file.

Note: A null VDE$LIBRARY_DEFAULTS logical name definition, or a missing VDE$LIBRARY_DEFAULTS logical name definition will cause VDE to access the default Motif resource file looking for the list of defaults. The null definition logic allows a non-privileged user to override a system-wide or group-wide logical name definition (using a null logical name in a searched-before

logical name table) and use a set of default library definitions located in a user-specific or system-wide resource file.

An example of the definition of the VDE$LIBRARY_DEFAULTS, VDE$LIBRARY_DEFAULTS_LABELS, VDE$LIBRARY_DEFAULTS_CMS, and VDE$LIBRARY_DEFAULTS_MNEMONICS logical names follows:

```
$ DEFINE/JOB VDE$LIBRARY_DEFAULTS -
_$ SYS$SYSDEVICE:[LIB1], -
_$ XYZZY$:[LIB2], -
_$ PLUGH$:[MASTERLIB]
$ DEFINE/JOB VDE$LIBRARY_DEFAULTS_LABELS -
_$ "Library One", -
_$ "Library Two", -
_$ "Master Library"
$ DEFINE/JOB VDE$LIBRARY_DEFAULTS_MNEMONICS -
_$ "L1", -
_$ "L2", -
_$ "ML"
$ DEFINE/JOB VDE$LIBRARY_DEFAULTS_CMS -
_$ "L1CMS", -
_$ "L2CMS", -
_$ "MLCMS"
```

The mnemonic names for the libraries are also accepted by the command (non-graphical) interface SET LIBRARY command. The following two commands, based on the aforementioned logical names, are valid and equivalent:

```
VDE> SET LIBRARY ML
VDE> SET LIBRARY PLUGH$:[MASTERLIB]
```

10.1.4 Motif Resources

This section is under construction...

See the *Save Options* entry on the *Options* pulldown for information.

10.2 WWW/Mosaic Interface

This section is under construction...

A TroubleShooting Information

The following sections contain troubleshooting information. And in addition, see the VDE Reference Manual.

This section is under construction...

A.1 Periodic Maintenance

VDE has a few tasks that will need to be performed at periodic intervals.

- BACKUPs

  Periodic BACKUPs of the VDE library are required, both the CMS libraries and the a reference>(Rdb_acronym) database.

  See Section 7.4.

- Staging

  VDE stages all replacement files into the staging area, and the files in this area must be manually deleted at site-specific intervals. The interval these deletions must occur is dependent on the amount of freespace on the staging disk, and on the size and frequency of the replacements, and on the backup policies used for the staging disk and the CMS library disk(s), and on the likelyhood the staging files will be needed during any potential recovery of the CMS libraries.

  See Section 7.5.

A.2 CMS Troubleshooting

This section is under construction...

A.2.1 CMS Library Locked

The following example shows an CMS CMS$_TRYAGNLAT interlock error.

```
%VDE-I-GENDEL, generation [KITTING]STARTUP.UP3;38(38) deleted from stream DRAGON
%VDE-I-GENRETAINED, generation [KITTING]STARTUP.UP3;38(38) retained in library
%CMS-E-NOREF, error referencing VMS$:[000000.KITTING.CMS]
-CMS-E-TRYAGNLAT, library locked; try again later
-RMS-E-FLK, file currently locked by another user
%VDE-E-CMSINSGEN, CMS error inserting generation 37 of element STARTUP.UP3 into cla
-CMS-E-NOREF, error referencing !AS
%VDE-I-ROLLBACK, database transaction has been rolled back
```

Based on the particular CMS library that caused the error, the following example shows the commands necessary to detect the process holding the interlock among all member nodes present in the VMScluster, and how to (brute-force) free it.

```
$ Run Sys$System:SysMan
SYSMAN> DO SHOW DEVICE/FILE/NOSYSTEM VMS$
%SYSMAN-I-OUTPUT, command execution on node HELENA
Files accessed on device DSA69: on 21-MAR-1997 17:50:22.21
%SYSMAN-I-OUTPUT, command execution on node GALAXY
Files accessed on device DSA69: on 21-MAR-1997 17:50:24.10
%SYSMAN-I-OUTPUT, command execution on node DIMOND
Files accessed on device DSA69: on 21-MAR-1997 17:50:25.23
Process name      PID     File name
DRAGONBLD       6081487F  [VMS.KITTING.CMS]00CMS.CMS;1
%SYSMAN-I-OUTPUT, command execution on node EDISON
Files accessed on device DSA69: on 21-MAR-1997 17:50:27.65
%SYSMAN-I-OUTPUT, command execution on node BRYTT
Files accessed on device DSA69: on 21-MAR-1997 17:50:43.58
SYSMAN> ^Z
$ SET PROCESS/PRIVILEGE=WORLD
$ STOP/ID=6081487F
```

In the above examples, the VMS$ logical name translates to DSA69:.

A.2.2 CMS 3.8-2 and Indexed Files

Under certain circumstances, CMS 3.8-2 cannot correctly retrieve an indexed file stored in the CMS library. The file remains valid and remains uncorrupted within the CMS library.

```
%CMS-E-NOFETCH, error fetching element [name]
-CMS-E-OPENOUT, error opening !AS for output
-CMS-E-OPENOUT, error opening [name] for output
-RMS-E-NPK, no primary key defined for indexed file
%CMS-F-BADBUG, there is an unrecoverable bug in CMS or something it calls
-CMS-F-RBKNOTOPEN, File passed to rollback is not open
```

This is a known problem in CMS 3.8-2 on OpenVMS Alpha. Do not store indexed files directly in CMS, store all binary files as a VDE ‘‘marker file’’.

A.2.3 Recovering Lost CMS Libraries

If the delta files (the CMS libraries) in your VDE library are damaged or lost due to a disk failure or other cause, you must recover as much of its lost contents as possible.

See Section 7.4.10 for details on how to perform this task.

A.2.4 Moving to Larger CMS Disks

This section describes how to move the CMS library to different—and usually larger—disks.

See Section 7.4.11 for one procedure used to get to larger disks.

A.3 VMS_MENU Troubleshooting

This section is under construction...

VMS_MENU displays generated VDE commands the logical name VDE$$DBG_SCRIPT is defined.

A.4 VDE Spawned Processing Troubleshooting

This section is under construction...

Like VMS_MENU, VDE uses logical names to enable the display of DCL commands executed in spawned subprocesses. If the logical name is VDE$$DBG_SENDMSG is defined, the DCL code generated by various VDE routines will be displayed during execution.

A.5 Rdb Troubleshooting

A.5.1 User Process Deadlocks

Process-level deadlocks occur infrequently, and they often involve high-traffic applications such as database backups or they involve a VDE command that was paused via CTRL/S XOFF command.

To resolve these deadlocks, see Section 7.3.1.

A.5.2 Freeze in Rdb

It is possible for Rdb processes to deadlock waiting for the Rdb ‘‘freeze’’ resource.

To resolve this deadlock, see Section 7.3.2.

A.5.3 Resolving Rdb process dumps

Rdb generates a dump file when it encounters an unexpected condition. Under typical circumstances, Rdb will display a message indicating a dump is being generated.

To determine the cause of the dump, see Section 7.3.3.

A.5.4 Rdb run-unit journal errors

The run-unit journal area is a key component of the Rdb processing, and allows Rdb to rollback incomplete or erroneous database operations. And when an unexpected condition occurs with the run-unit journal, Rdb tends to generate a dump file.

To resolve these errors, see Section 7.3.4.

A.5.5 Missing run-unit journal area

If the run-unit journal area is entirely messing, one can potentially see an Rdb bugcheck dump generated.

To resolve these errors, see Section A.5.5.

A.5.6 Rdb run-unit journal diskquota errors

Rdb requires the creation of a run-unit journal file. By default, this is in the user's login device, in the directory [RDM$RUJ], and this location can be altered using the logical name RDMS$RUJ.

To resolve these errors, see Section 7.3.4.2.

A.5.7 Rdb recovery process dumps

Rdb generates a dump file when the recovery process encounters an unexpected condition. Also see Section A.5.7.1.

To resolve these errors, see Section 7.3.4.3.

A.5.7.1 Missing run-unit journal files

A recovery-process dump similar to Section A.5.7 can also be seen when the RUJ file is missing.

To resolve these errors, see Section 7.3.4.4.

A.5.8 Recovering from a Lost Journal File

If the after-image journal file for your VDE database becomes unavailable due to a disk failure or other cause, nobody will be able to use the database and VDE operations will fail.

To recover from these errors, see Section 7.4.8.

A.5.9 Moving to Larger Rdb Disks

Sometimes, one just runs out of storage space on the disk(s) holding the Rdb database.

See Section A.5.9, and Section 7.8.

B Comparison of VDE, VSC, and VTSC

Certain VSC installations have three command interfaces: VDE, VSC, and VTSC.

As mentioned previously, the VDE, VSC and VTSC command interfaces all use the same command syntax, all use the same message prefix, and all perform the same basic functions. Table B–1 contains a high-level comparison of the differences among the command interfaces.

Table B–1 VDE, VSC and VTSC

VDE	VSC	VTSC	Operation
No	Yes	No	Defaults to referencing the OpenVMS VAX library
No	No	Yes	Defaults to referencing the OpenVMS testing library
Yes	No	No	Defaults to the library referenced by the VDE$LIBRARY logical name
Yes	No	No	Defaults to processing the command in the current process.
No	Yes	Yes	Defaults to spawning a kept subprocess and processing commands there.
Yes	Yes	Yes	Can operate correctly on the OpenVMS VAX library.
Yes	Yes	Yes	Can SET LIBRARY and can operate correctly on user-specified private source code libraries.

There are no other operational differences among the command interfaces.

While this document generally refers to the VDE utility and sometimes to VSC utility, the information here is also directly applicable to the VTSC utility. And in a few places—where information related to libraries other than the OpenVMS VAX source code control library is presented—the VDE utility is specifically used. For additional information on VSC and VDE, please see the *VDE Reference Manual*.

Why are there three interfaces? Simple: each was tailored to the needs of specific parts of the OpenVMS community. However, due to the creation of various additional libraries and the common use of these libraries for various purposes, the use of the ‘‘stock’’ VDE image has generally become preferable.

Index

A

ABM • 7–24
Accepting a queued replacement • 4–7
ACCEPT REPLACEMENT command • 4–7
  qualifiers • 4–8
ACCEPT subcommand • 4–4
ACCONFLICT • 7–5
Adding users to database • 6–9, 6–21
AFTER_JOURNAL • 7–24
AIJ • 7–14, 7–24
AIJTERMINATE • 7–24
ANALYZE/SYSTEM • 7–3
architecture • 9–1
arch-variant • 9–1
At-sign (@) command • 1–7
ATTACH command • 1–7
ATTACH subcommand • 4–4
Attributes
  of a library • 6–7
Audit trail identifiers • 2–2

B

Backing up the Rdb database • 7–9
BACK subcommand • 4–3
Backup • 7–9
BACKUP
  Lack of Database Synchronization • 7–8
Backup procedures for library database • 7–7
Backup Recovery • 7–12
BADROOTMATCH • 7–18
bugcheck dump • 7–4
Bugcheck Files
  See RDSBUGCHK
BUILD command • 8–1
Build jobs
  creating • 8–1
  displaying • 8–1
  running • 8–1
  starting • 8–1
  stopping • 8–2
  suspending • 8–2
  waiting for • 8–2
Build steps
  compile steps • 8–2
  copy steps • 8–2
  link steps • 8–2
  performing • 8–2
  scripts for • 8–2
BYPASS
  Privilege • 7–2

C

CANCEL FOLD Command • 2–24
Cancelling Replacements • 2–26
CANCEL LOG command • 1–7
CANCEL REPLACEMENT Command • 2–26
  Qualifiers • 2–27
CANCEL VERIFY command • 1–7
Change propagation • 2–23, 5–11, 5–12
CLD • 2–16
CMS • 6–2, 6–26
  history and notes defaults • 6–12
  importation • 6–25
  indexed file error • A–2
  library locked error • A–1
  Multiple Architectures • 9–1
  Uploading from • 6–2
CMS$_BADBUG • A–2
CMS$_NOREF • A–1
CMS$_RBKNOTOPEN • A–2
CMS$_TRYAGNLAT • A–1
CMS libraries
  Defragmentation • 7–23
  recovering lost information • 7–15
CMS library
  build stream classes • 5–9
  class names • 5–11
  development stream names • 5–11
  populating facilities from • 6–26
CMS overhead
  Delta • 2–10
Command
  command character • 1–6
  command recall • 1–2
  continuation character • 1–1, 1–6
  seperator character • 1–1

Command Procedure
NIGHTLY_VDE_BACKUP • 7–10
POPULATE_LIBRARIES • 5–9, 5–11
POP_REUSED_CLASS • 5–10, 5–11
VDE_DATABASE_BACKUP • 7–9, 7–10
VDE_DATABASE_RESTORE • 7–12
Comment Character • 1–6
COMPILE command • 8–2
Compile steps • 8–2
Compiling modules • 8–2
Concurrent reservations • 2–7
Contexts • 1–8
Continuation Character • 1–1, 1–6
Conversion
Automatic • 6–11
of library format • 6–10
Converting
From CMS • 6–2, 6–26
CONVERT LIBRARY command • 6–11
COPY command • 8–2
Copying modules • 8–2
Copy steps • 8–2
CRE • 7–4
CREATE ARCHITECTURE command • 6–13, 9–2
CREATE DIRECTORY_TREE command • 6–26
CREATE FACILITY/FROM_CMS_LIBRARY command • 5–10
CREATE FACILITY command • 6–24, 6–26
CREATE LIBRARY/DATABASE command • 6–8
CREATE LIBRARY/SNAPSHOT command • 6–8
CREATE LIBRARY command • 6–2, 6–3, 6–4, 6–11
examples • 6–4
CREATE MODULE command • 2–9, 6–25, 6–27, 9–3
CREATE MODULE Command
Examples • 2–10
Qualifiers • 2–9
CREATE SCRIPT command • 5–14, 8–2
CREATE STREAM command • 5–9, 5–10, 6–23
CREATE USER command • 6–9, 6–21, 6–23
examples • 6–21
Creating
a VDE library • 6–1
build jobs • 8–1
facilities • 6–24
Modules • 2–9
Streams • 5–8, 6–23
users • 6–21
CURRENT subcommand • 4–2

D

Database
backup procedures • 7–7
Device name changes • 7–18
Moving databases • 7–18
placement • 6–7
recovery procedures • 7–7
DB_CORRUPT • 7–18
Default Facility
switching • 1–8
Default Stream
switching • 1–8
DEFINE/COMMAND command • 1–7
DEFINE/KEY command • 1–7
Defragmentation • 7–23
DELETE ARCHITECTURE command • 9–2
DELETE GENERATION command • 5–6
qualifiers • 5–6
DELETE MODULE Command • 2–9
Deleting
database entities • 6–12
Deleting generations • 5–6
Delta
CMS overhead • 2–10
Delta files
directory structure • 6–14
Dependency graph
initializing • 8–2
Development Streams
Displaying • 3–11
Specification • 1–3
Differences
displaying • 3–9
DIFFERENCES command • 3–9
qualifiers • 3–10
Directory structure
delta files • 6–14
of a library • 6–13
staging area • 6–15
Directory Structure
Streams • 6–17
DIRECTORY subcommand • 4–2
Directory tree
for streams • 6–24
DISKQUOTA • 7–4, 7–5
Disk Quota errors • 7–4
Displaying
build jobs • 8–1
default settings • 1–3, 1–4

Displaying (cont'd)
Development • 3–11
differences between generations • 3–9
editor definitions • 1–7
facilities • 3–1
Generations • 3–4
modules • 3–2
replacements • 2–24
Reservations • 2–13
Streams • 3–11
users • 3–13
Dump Files
See RDSBUGCHK

E

Editing text • 1–7
Editor definition • 1–7
EDIT subcommand • 4–3
Errors
Generating bugcheck dump... • 7–4
RDB$_DB_CORRUPT • 7–18
RDB$_SYS_REQUEST • 7–5, 7–18
RDMS$_BADROOTMATCH • 7–18
RDMS$_FILACCERR • 7–5, 7–18
SS$_ACCONFLICT • 7–5
SS$_EXDISKQUOTA • 7–5
SS$_FILACCERR • 7–5
Examples
FETCH • 2–12
RESERVE Command • 2–6
SHOW REPLACEMENT Command • 2–26
SHOW RESERVATION • 2–14
Upload • 6–2, 6–26
EXDISKQUOTA • 7–4, 7–5
Executing build jobs • 8–1
EXIT/KEPT_PROCESS command • 1–2, 1–6
EXIT command • 1–7
EXIT subcommand • 4–2

F

Facilities • 1–3
adding new modules to • 6–27
creating • 6–24
displaying • 3–1
populating from CMS library • 6–26
populating from directory • 6–25
FETCH command • 2–11, 9–4
FETCH Command
Examples • 2–12
Qualifiers • 2–11
Fetching source modules • 2–11
FILACCERR • 7–5, 7–18
FIRST subcommand • 4–3
Fold Records • 1–5
Cancelling • 2–24
Cross-Architecture • 9–5
Definition Of • 2–23
Displaying • 2–23
Freeze deadlock • 7–3
FUL
staging device full • 7–20

G

Generating bugcheck dump... • 7–4
Generation
CMS • 2–1
expression • 2–1, 2–2
number • 1–5, 2–1
Generations • 1–4
deleting • 5–6
Displaying • 3–4
inserting • 5–3
recovering • 7–16, 7–17
removing • 5–4
verifying • 7–16

H

Hang on SET LIBRARY • 7–3
HELP command • 1–7
History
Displaying • 3–4
History and notes
default settings • 6–12

I

Identifiers
audit trail • 2–2

Immediate replacements • 2–18
Initialization file • 1–6
Initializing
  dependency graph • 8–2
INSERT GENERATION command • 5–3, 9–4
  qualifiers • 5–4
Inserting generation into a stream • 5–3

J

Journal file
  recovering from lost • 7–13
Journal File • 7–14, 7–24
  Reinitialization • 7–24

K

Kept Subprocess • 1–5, B–1

L

LAST subcommand • 4–3
Library
  See Library Mnemonics
  adding users to • 6–21
  Closing • 6–11, 7–2
  Corrupt • 7–6
  creating • 6–1
  database backup procedures • 7–7
  database recovery procedures • 7–7
  directory structure • 6–13
  Master Pack • 1–1, B–1
  Mnemonics • 10–4
  Private • B–1
  Privileges • 6–8
  RMU Backup • 7–9
  RMU Recovery • 7–12
  Testing Master Pack • B–1
  Test Suite Master Pack • 1–1
Library attributes • 6–7
  adding users • 6–9
  CMS history and notes • 6–12
  database placement • 6–7
  deletion of entities • 6–12
  directory structure • 6–13
  format conversion • 6–10
Library attributes (cont'd)
  Mail notifications • 6–11
  miscellaneous • 6–13
Library entities
  allowing deletion of • 6–12
LINK command • 8–2
Linking modules • 8–2
Link steps • 8–2
Logical name
  VDE$$DBG_SCRIPT • A–2
  VDE$$DBG_SENDMSG • A–3
Logical Name
  initialization files • 1–6
  RDMS$RUJ • 7–4
  SYS$SCRATCH • 3–10
  VDE$INIT • 1–6
  VDE$LIBRARY • 10–2, B–1
  VDE$LIBRARY_DEFAULTS • 10–3, 10–4
  VDE$LIBRARY_DEFAULTS_CMS • 10–3, 10–4
  VDE$LIBRARY_DEFAULTS_LABELS • 10–3, 10–4
  VDE$LIBRARY_DEFAULTS_MNEMONICS • 10–3, 10–4
  VDE$OFFLINE • 7–1
  VDE$SCRATCH • 3–10
  VSC$INIT • 1–6
  VTSC$INIT • 1–6

M

Mail notification • 6–11
Marker file • 7–20, 7–23
Marker File • 2–10
Mnemonics
  See Library Mnemonics
MODIFY ARCHITECTURE command • 9–2
MODIFY GENERATION command • 9–4
MODIFY LIBRARY/LOCK command • 7–1
MODIFY LIBRARY/NOAUTO_CONVERT command • 6–10
MODIFY LIBRARY command • 6–4, 6–11, 6–13, 6–23
MODIFY MODULE command • 5–7, 5–8, 9–3
MODIFY STREAM command • 5–7, 5–8, 5–11
MODIFY USER command • 6–9
Modules • 1–3
  adding to facilities • 6–27
  compiling • 8–2
  copying • 8–2
  Creating • 2–9
  displaying • 3–2

Modules (cont'd)
linking • 8–2
name syntax • 1–3
Mosaic • 1–1, 10–1
Motif • 1–1, 10–1

N

Name Space • 1–3
NEXT subcommand • 4–3
NIGHTLY_VDE_BACKUP command procedure • 7–10
Notification lists • 5–7

O

Obsolete
marking a file as • 5–7
Offline • 7–1
OpenVMS
Master Pack • 1–1

P

Performing
build steps • 8–2
Performing replacements • 5–1
PERFORM REPLACEMENT command • 5–1
examples • 5–3
qualifiers • 5–2
PERFORM REPLACEMENT Command • 2–18
POPULATE_LIBRARIES
Command Procedure • 5–9, 5–11
POPULATE_LIBRARIES command procedure • 5–9, 5–11
Populating
directories for new streams • 6–24
facility from CMS library • 6–26
facility from directory • 6–25
facility with new modules • 6–27
POP_REUSED_CLASS
Command procedure • 5–11
Command Procedure • 5–10
POP_REUSED_CLASS command procedure • 5–10, 5–11
Privileges
BYPASS • 7–2
CREMOD • 5–8, 6–9
CRESCRP • 5–14
CRESTRM • 5–8
Default • 6–9, 6–21
Inheritance • 1–6
Kept Subprocesses • 1–6
MODLIB • 6–11
PERFREP • 5–2, 5–3, 5–5, 5–6
READALL • 6–23
RESREP • 6–9
SETPRV • 7–1
Standard for library • 6–8
SYSPRV • 6–11, 7–1, 7–2, 7–12, 7–14
Process Deletion
Unexpected • 7–5, 7–6
Propogation • 1–5

Q

QAR • 2–16
Quality Assurance Report
QAR
Queued Replacements • 2–17, 6–3
Queued Reservations • 2–9
Quotas
Inheritance • 1–6
Kept Subprocesses • 1–6

R

Rdb
bugcheck dump • 7–4
RDB$JOURNAL • 7–24
RDB$_DB_CORRUPT • 7–18
RDB$_SYS_REQUEST • 7–5, 7–18
Rdb Bugcheck Files
See RDSBUGCHK
Rdb database
backing up • 7–9
Rdb Dump Files
See RDSBUGCHK

RDM$RUJ • 7–5
RDMDBRBUG.DMP • 7–5
RDMS$RUJ Logical Name • 7–4
RDMS$_BADROOTMATCH • 7–18
RDMS$_FILACCERR • 7–5, 7–18
RDSBUGCHK • 7–25
RDSBUGCHK.DMP • 7–4
Recovering
  from lost journal file • 7–13
  generations • 7–16, 7–17
  lost CMS information • 7–15
Recovery procedures for library database • 7–7
Recovery Process Dump • 7–5
Recreating
  Journal Files • 7–24
  Snapshot Files • 7–24
Rejecting a queued replacement • 4–7
REJECT REPLACEMENT command • 4–7
  qualifiers • 4–8
REJECT subcommand • 4–4
REMOVE GENERATION command • 5–4
  qualifiers • 5–5
Removing generations • 5–4
Repairing
  ABM • 7–24
  AIJ • 7–24
  AIJTERMINATE errors • 7–24
  SPAM • 7–24
REPLACE command • 7–20, 9–4, 9–5
  qualifiers • 2–19
REPLACE Command • 2–15
  Examples • 2–21
Replacements
  accepting queued • 4–7
  Cancelling • 2–26
  Creating • 2–15
  displaying • 2–24
  immediate • 2–18
  performing queued • 5–1
  Queued • 2–17, 6–3
  Rejecting Queued • 4–7
  reviewing queued • 4–1
Replacing Source Modules • 2–15
Reservations
  Canceling • 2–13
  creating • 2–2
  Displaying • 2–13
  queued • 2–9
RESERVE command • 2–2, 2–9, 9–4
RESERVE Command
  Examples • 2–6
  Qualifiers • 2–4
Reserving source modules • 2–2
Restoring the VDE database • 7–12
Reviewer lists • 5–7
Reviewing queued replacement • 4–1
REVIEW REPLACEMENT command • 4–1
  examples • 4–5
  qualifiers • 4–2
REVIEW REPLACEMENT Command
  Subsystem Commands • 4–2
Rightslist
  Inheritance • 1–6
  Kept Subprocesses • 1–6
RMONSTART • 7–18
RMONSTART41 • 7–18
RMONSTART60 • 7–18
RMONSTOP • 7–17
RMONSTOP41 • 7–17
RMONSTOP60 • 7–17
RMS$_CRE • 7–4
RMS$_FUL
  staging device full • 7–20
RMU • 6–11, 7–2, 7–6, 7–9, 7–12, 7–13, 7–14
RMU/ALTER • 7–14, 7–24
RMU/ALTER command • 7–6
RMU/BACKUP • 7–18, 7–19
RMU/BACKUP command • 7–6
RMU/CLOSE • 7–18, 7–24
RMU/DUMP/HEADER/OPTIONS=DEBUG command • 7–25
RMU/DUMP command • 7–25
RMU/MOVE_AREA • 7–18
RMU/REPAIR • 7–24
RMU/REPAIR/ABM • 7–24
RMU/SET • 7–24
RMU/SET AFTER_JOURNAL command • 7–25
RMU/SHOW LOCK/MODE=BLOCKING command • 7–3
RMU/UNLOAD command • 7–6
Root • 9–1
Running build jobs • 8–1

S

Script Commands • 5–14

Scripts • 1–5, 5–14
  for build steps • 8–2
  populating New Streams • 6–24
SCRPMODFAIL
  staging device full • 7–20
SCT Notes Conference • 7–1
Security • 7–1
SELECT subcommand • 4–3
Seperator Character • 1–1
Sessions • 1–7
SET ARCHITECTURE command • 9–2
SET DIRECTORY command • 6–13
SET EDITOR command • 1–7
SET FACILITY command • 1–3, 2–3, 2–7
SET KEY command • 1–7
SET LIBRARY
  Hang • 7–3
SET LIBRARY command • 10–4, B–1
SET LOG command • 1–7
SET PRIVILEGES command • 6–8
SETPRV
  Privilege • 7–1
SET SECURITY command • 7–1
SET STREAM command • 1–3
SET TERMINAL command • 1–7
SET VERIFY command • 1–7
SHOW ARCHITECTURE command • 9–2
SHOW BUILD_JOB command • 8–1
SHOW DEFAULTS command • 1–3, 1–4
SHOW DIRECTORY command • 2–10, 7–20
SHOW EDITOR command • 1–7
SHOW FACILITY command • 3–1
  examples • 3–2
  qualifiers • 3–1
SHOW FOLD command • 2–23
SHOW GENERATION command • 3–4, 9–4
  examples • 3–7
  qualifiers • 3–5
SHOW HISTORY command • 3–4
SHOW LOCATION/STAGING command • 7–19
SHOW MODULE command • 3–2, 7–20
  examples • 3–3
  qualifiers • 3–3
SHOW REPLACEMENT command • 2–24, 4–8
  qualifiers • 2–24
SHOW REPLACEMENT Command
  Examples • 2–26
SHOW RESERVATION command • 2–13
SHOW RESERVATION Command
  Examples • 2–14
  Qualifiers • 2–13
SHOW STREAM command • 3–11, 5–12
  examples • 3–12
  qualifiers • 3–11
SHOW subcommand • 4–3
SHOW USER command • 3–13
  examples • 3–13
  qualifiers • 3–13
Snapshot • 6–8
Snapshot File
  Reinitialization • 7–24
SPAM • 7–24
SPAWN command • 1–7
SPAWN subcommand • 4–4
SS$_ACCONFLICT • 7–5
SS$_EXDISKQUOTA • 7–4, 7–5
SS$_FILACCERR • 7–5
Staging area
  directory structure • 6–15
Staging Area
  periodic cleanup requirements • 7–19
  VDE$REP subdirectory • 7–19
  VDE$STG subdirectory • 7–19
Standard privileges • 6–8
START BUILD_JOB command • 8–1
Starting build jobs • 8–1
Steps
  compile steps • 8–2
  copy steps • 8–2
  link steps • 8–2
  performing • 8–2
  scripts for • 8–2
STOP BUILD_JOB command • 8–2
Stopping build jobs • 8–2
Streams
  Architecture-dependent • 9–5
  Architecture-independent • 9–5
  Creating • 5–8, 6–23
  Development • 1–3
  Directory Structure • 6–17
  Directory trees • 6–24
  Displaying • 3–11
  Fold Records • 1–5
  Scripts to Populate • 6–24
  Specification • 1–3
  Successors • 5–11, 5–12, 6–24
SUBMIT
  Nightly VDE Backup • 7–10
Successors for streams • 6–24
Successor streams • 5–11, 5–12

SUSPEND BUILD_JOB command • 8–2
Suspending build jobs • 8–2
SYS$SCRATCH
  Logical Name • 3–10
SYS$SYSTEM:RDMDBRBUG.DMP • 7–5
SYSMAN • 7–4, 7–5
SYSPRV
  Privilege • 7–1, 7–2
System Authorization Database • 6–23
SYSUAF • 6–10, 6–22, 6–23
SYS_REQUEST • 7–5, 7–18

T

Troubleshooting • A–1

U

UAF • 6–22
UNRESERVE command • 2–13
UNRESERVE Command
  Qualifiers • 2–13
Unreserving source modules • 2–13
Upload Example • 6–2, 6–26
Uploading CMS Libraries • 6–2, 6–26
Users
  adding to database • 6–9, 6–21
  creating • 6–21
  displaying • 3–13

V

VDE$$DBG_SCRIPT logical name • A–2
VDE$$DBG_SENDMSG logical name • A–3
VDE$BUILD • 1–5
VDE$CREATE.COM • 6–2, 6–26
VDE$EXAMPLES logical name • 6–26
VDE$INIT • 1–6
VDE$LIBRARY • 10–2, B–1
VDE$LIBRARY_DEFAULTS • 10–3, 10–4
VDE$LIBRARY_DEFAULTS_CMS • 10–3, 10–4
VDE$LIBRARY_DEFAULTS_LABELS • 10–3, 10–4
VDE$LIBRARY_DEFAULTS_MNEMONICS • 10–3, 10–4
VDE$MARKER symbol • 7–21
VDE$OFFLINE • 7–24
VDE$OFFLINE logical name • 7–1
VDE$REPLACE_DEFAULT logical name • 2–17
VDE$REP subdirectory • 7–19
VDE$SCRATCH
  Logical Name • 3–10
VDE$STG subdirectory • 7–19
VDE$SYSTARTUP.COM • 7–18, 7–24
VDE$UPLOAD • 6–2, 6–27
VDE$_OFFLINE • 7–2
VDE$_SCRPMODFAIL
  staging device full • 7–20
VDE/READ_UAF command • 6–22, 6–23
  examples • 6–23
VDE database • 6–1
  conversion of format • 6–10
  placement • 6–7
  recovering from lost journal file • 7–13
  restoring • 7–12
VDE library
  creating • 6–1
VDE utility • 1–1
VDE vs VSC • 1–1
VDE_DATABASE_BACKUP command procedure • 7–9, 7–10
VDE_DATABASE_RESTORE command procedure • 7–12
VERIFY GENERATION command • 7–16, 7–17
Verifying
  generations • 7–16
VSC$INIT • 1–6
VSC utility • 1–1
VSC vs VDE • 1–1
VTSC$INIT • 1–6
VTSC utility • 1–1

W

WAIT BUILD_JOB command • 8–2
Waiting for build jobs • 8–2
Wildcard characters • 2–3
WWW • 10–1

X

X Windows • 10–1