VMS Software

VSI OpenVMS

VSI DECset for OpenVMS Guide to the Code Management System

Operating System and Version: VSI OpenVMS x86-64 Version 9.2-2 or higher
VSI OpenVMS IA-64 Version 8.4-1H1 or higher
VSI OpenVMS Alpha Version 8.4-2L1 or higher

Software Version: DECset Version 12.7

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

VSI DECset for OpenVMS Guide to the Code Management System

VMS Software

Table of Contents

Preface

The Code Management System for OpenVMS (CMS) is an online library system that helps track software development and maintenance. This guide provides reference and conceptual information on how to use CMS on OpenVMS systems.

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. Intended Audience

This guide is intended for all users of CMS, including managers, project programmers, writers, and others who might be responsible for maintaining CMS libraries.

This guide can be used by both experienced and novice users of CMS. You do not need a detailed understanding of the OpenVMS operating system. However, some familiarity with the conventions of the Digital Command Language (DCL) is helpful.

3. Document Structure

This guide is task-oriented and provides information on how to use CMS. It is divided into the following chapters and appendices:

- *Chapter 1, "Introduction to CMS"* describes the basic concepts of CMS and presents a tutorial example to help you get started.
- *Chapter 2, "Using CMS with DECwindows Motif"* describes how to use the CMS DECwindows Motif user interface.
- *Chapter 3, "Libraries"* describes how to set up a CMS library and how to use library search lists.
- *Chapter 4, "Elements and Generations"* explains the concepts of files in a CMS library.
- *Chapter 5, "Groups and Classes"* explains how to organize files into groups and classes.
- *Chapter 6, "Variants and Merging"* describes lines of descent, how to create variant lines of descent, and how to merge files.
- *Chapter 7, "Security Features"* describes the protection mechanisms that you can use in CMS.
- *Chapter 8, "Event Handling and Notification"* describes how CMS handles events and the concept of notification when these events occur.
- *Chapter 9, "Library Maintenance"* describes how to maintain the validity and integrity of your CMS library.
- *Chapter 10, "Command Syntax"* gives detailed information on CMS syntax and how to specify commands.
- *Appendix A, "Summary of CMS Interface Functional Mappings"* provides a table displaying how each of the CMS interfaces are functionally mapped to each other.

- *Appendix B, "CMS Library Storage Method"* contains information on how libraries are stored.
- *Appendix C, "System Management Considerations"* contains information about running CMS on the OpenVMS operating system.

4. Related Documents

The following documents might also be helpful when using CMS:

- The *VSI DECset for OpenVMS Installation Guide* contains instructions for installing CMS.
- The *Code Management System for OpenVMS Release Notes* contain added information on the use and maintenance of CMS.
- The *CMS Client User's Guide* describes the installation and use of the CMS Client software in a Microsoft Windows environment.
- The *VSI DECset for OpenVMS Code Management System Reference Manual* describes all the commands available for CMS.
- The *VSI DECset for OpenVMS Code Management System Callable Routines Reference Manual* describes the set of CMS callable routines.
- The *Using VSI DECset for OpenVMS Systems* contains information on using the other components of DECset.

5. References to Other Products

Some older products that DECset components worked with previously may no longer be available or supported by VSI. References in this manual to such products serve as examples only and do not imply that DECset has conducted recent interoperability testing.

See the Software Product Description for a current list of supported products that are warranted to interact with DECset components.

6. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

7. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: `<docinfo@vmssoftware.com>`. Users who have VSI OpenVMS support contracts through VSI can contact `<support@vmssoftware.com>` for help with this product.

8. Conventions

VMScluster systems are now referred to as OpenVMS Cluster systems. Unless otherwise specified, references to OpenVMS Cluster systems or clusters in this document are synonymous with VMScluster systems.

The contents of the display examples for some utility commands described in this manual may differ slightly from the actual output provided by these commands on your system. However, when the behavior of a command differs significantly between OpenVMS Alpha and Integrity servers, that behavior is described in text and rendered, as appropriate, in separate examples.

In this manual, every use of DECwindows and DECwindows Motif refers to DECwindows Motif for OpenVMS software.

The following conventions are also used in this manual:

Convention	Meaning
Ctrl/ *x*	A sequence such as **Ctrl/** *x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.
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 command format descriptions, parentheses indicate that you must enclose the options in parentheses if you choose more than one.
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.
[|]	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are options; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold text	This typeface represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER= *name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.

Convention	**Meaning**
`Monospace type`	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

Chapter 1. Introduction to CMS

The Code Management System (CMS) for OpenVMS is a library system for software development and maintenance. CMS stores files called elements in an online library, keeps track of changes made to these files, and monitors user access to the files.

This chapter provides the following information:

- An overview of CMS and information on how to get started
- An introduction to CMS concepts
- A sample CMS session
- A summary of CMS commands

1.1. Overview

During software development, programmers continually make changes to project files. CMS stores and monitors these files.

CMS enables you to store project files in a central library where they are available to all project members. Some of the tasks you can perform on these files are as follows:

- Store files (called elements) in a library.
- Fetch elements, modify them, and test them in your own directory.
- Control concurrent modifications to the same element.
- Merge concurrent modifications to an element.
- Create successive versions (called generations) of elements.
- Compare two generations of an element within a library.
- Organize related library elements into groups.
- Define a set of generations of elements as a class to make up a base level or release version of a project.
- Track which users are working on which elements from the library.
- Maintain a historical account of element and library transactions.

1.2. CMS Concepts

This section introduces basic CMS concepts.

1.2.1. Libraries, Elements, and Generations

CMS stores all the information it needs in a library. A CMS **library** is an OpenVMS directory containing specially formatted files. It serves as a container or repository for various CMS entities (called **objects**).

An **element** is the basic structural unit in a CMS library; it consists of one file and all its versions. An element **generation** represents a specific version of that element. When you create an element and place it in a CMS library for the first time, CMS creates generation 1 of that element. Each time you reserve and then replace a generation of an element in the library, CMS creates a new generation of that element.

For information on libraries, see *Chapter 3, "Libraries"*. For information on elements and generations, see *Chapter 4, "Elements and Generations"*.

1.2.2. Groups and Classes

A **group** is a set of elements (or other groups) that you can combine and manipulate as a unit. For example, you might create a group containing all the elements that process error messages.

A **class** is a set of particular generations of elements. You typically combine generations of elements into classes to represent progressive stages, or base levels, in the development of an entire system.

For information on groups and classes, see *Chapter 5, "Groups and Classes"*.

1.2.3. Reservations and Replacements

As changes are made to a file in the OpenVMS file system, new versions of that file are created. Similarly, as an element is developed in CMS, new generations of that element are created. In addition to storing the element and its generations, CMS manages the development process by using reservations and replacements.

A **reservation** exists in the CMS library when you retrieve an element generation with the intent to modify it. The reservation ends and a replacement occurs when you return the modified contents to the library.

For information on reservations and replacements, see *Chapter 4, "Elements and Generations"*.

1.2.4. Review

You can mark an element generation for **review** to indicate that its contents should be reviewed by other users. After the review process is complete, the element generation can be marked as having been accepted or rejected.

For information on marking an element generation for review and the review process, see *Section 4.5.4, "The Review Attribute"*.

1.2.5. History and Remarks

All CMS commands that modify a library or its contents are recorded in the library **history**. You can display any part of the history by using the SHOW HISTORY command. All commands that are recorded allow you to enter a remark, which is recorded in the history along with the command. Remarks are useful in explaining library and element modifications.

For information on library history, see *Chapter 4, "Elements and Generations"*. For information on remarks, see *Section 10.2.3, "Remarks"*.

1.2.6. Reference Copies

For easy reference, you can direct CMS to automatically store copies of the latest main-line generation of selected library elements in a separately designated directory, called a **reference copy directory**.

For information on reference copies, see Sections *Section 3.1.4, "Creating a Reference Copy Directory"* and *Section 4.5.3, "The Reference Copy Attribute"*.

1.2.7. Lines of Descent and Variant Generations

The first generation of a newly created element is generation 1. Every time you reserve and replace a generation of that element, CMS numbers a new generation by adding 1 to the number of the reserved generation. This new generation is a descendant of the generation from which it was created. The **main line of descent** consists of generation 1 and its direct descendants.

A generation can have only one direct ancestor and one direct descendant, but it can also have a number of variant descendants. Those generations that are not on the direct line of descent of a generation are called **variant generations**. You specify variant generations by adding a letter, called the **variant letter**, and the number 1 to the parent generation. For example, generation 2E1 is a variant descendant of generation 2.

A variant generation and its direct descendants (for example, generations 2E1, 2E2, 2E3) form a variant line of descent. A variant generation can have variant descendants; for instance, generation 2E1W1 is a variant descendant of generation 2E1.

For information on lines of descent and variant generations, see *Chapter 6, "Variants and Merging"*.

1.2.8. Concurrent Reservations

You can create variant generations at any time, but default usage creates successive generations along the same line of descent. You must create a variant generation when the direct successor of a reserved generation already exists and you replace a concurrent reservation.

A **concurrent** reservation exists when an element generation has been reserved more than once by one or more users. In this case, only one of these reservations can be replaced on the direct line of descent; the rest of the reservations must be replaced as variant generations.

For information on concurrency, see *Section 4.3, "Concurrency"*.

1.2.9. Merging and Conflicts

You can use variant generations to maintain separate but related development of an element, or you might have generations that have undergone concurrent development.

If concurrent changes have been made to a generation, you can **merge** the changes from one line of descent and some variant line of descent into a single generation.

CMS resolves changes from two generations by comparing them to their common ancestor generation. If both generations change a region of their common ancestor in different ways, this region is known as a **conflict**. Where the changes do not conflict, CMS includes the appropriate change; where the changes conflict, CMS includes the changes from both generations and flags the conflicting region. In either case, you should verify the resulting merged output for correctness; for example, program source code should be compiled and executed to ensure that it is syntactically and logically correct. After verifying and making any necessary modifications, you can replace the merged reservation.

For information on merging and conflicts, see *Chapter 6, "Variants and Merging"*.

1.2.10. Security

The OpenVMS operating system provides a security mechanism based on user identification codes (UIC) and access control lists (ACLs) to control access to files within the file system. Similarly, you can use **CMS ACLs** for controlling access to CMS objects through CMS operations. For you to successfully access an object in the CMS library, both the file system and the CMS internal security mechanism must allow you to do so.

For information on the OpenVMS and CMS security mechanisms, see *Chapter 7, "Security Features"*.

1.2.11. Events and Notification

You can use CMS ACLs to specify that a CMS object being accessed constitutes an event, and that some action should be taken when an event occurs. You can specify lists of people to be notified when certain events occur on objects in the CMS library. The default action performed is **notification** through the OpenVMS Mail Utility (MAIL) to one or more users. CMS provides a default notification event handler; in addition, you can write event handlers of your own for CMS to use.

For information on events and notification, see *Chapter 8, "Event Handling and Notification"*.

1.3. Invoking CMS

You can invoke CMS in the following ways:

- From the DCL command level
- From the CMS subsystem command level
- From a program that calls CMS routines directly
- From the DECwindows Motif user interface
- From the CMS Client user interface (for accessing OpenVMS CMS libraries from a PC environment)

Enter CMS commands at the DCL command level prompt ($) by preceding them with the word CMS. After each command executes, control is returned to DCL level. For example:

```
$ CMS SHOW RESERVATIONS
      .
      .
      .
$
```

You can invoke CMS as a subsystem in the command-line interface in the following ways:

- Enter the CMS command at the DCL prompt.
- Enter the CMS command with the /INTERFACE qualifier at the DCL prompt.
- Enter the CMS command with the /INTERFACE=CHARACTER_CELL qualifier and keyword at the DCL prompt.

For example, you can enter any one of the following commands to enter CMS command-line subsystem mode:

```
$ CMS
CMS> SHOW RESERVATIONS
    .
    .
    .

$ CMS/INTERFACE
CMS> SHOW RESERVATIONS
    .
    .
    .

$ CMS/INTERFACE=CHARACTER_CELL
CMS> SHOW RESERVATIONS
    .
    .
    .
```

For information on entering the CMS DECwindows Motif interface, see *Section 2.1, "Invoking CMS"*.

You should enter the CMS subsystem when you plan on entering a series of CMS commands. This avoids the overhead involved with invoking CMS multiple times.

To terminate the CMS session and return to DCL level, type EXIT or press Ctrl/Z.

Full CMS functionality is available at the command-line level, and most of this guide describes how to use CMS in that manner. However, *Chapter 2, "Using CMS with DECwindows Motif"* provides information on accessing CMS commands and options through the DECwindows pull-down menus and dialog boxes. In addition, callable interface routines are described in the VSI DECset for OpenVMS Code Management System Callable Routines Reference Manual. *Appendix A, "Summary of CMS Interface Functional Mappings"* contains a table that shows how each of these CMS interfaces map to each other.

1.4. Getting Help

You can get information about CMS either at DCL level or at CMS subsystem level. At DCL level, the DCL command HELP CMS provides online help on CMS commands, qualifiers, and other topics. For example:

```
$ HELP CMS
```

To get help on a specific CMS command, such as the CREATE ELEMENT command, type the command after HELP CMS, as follows:

```
$ HELP CMS CREATE ELEMENT
```

You can get help at the CMS subsystem level by typing either HELP, or HELP and the specific command. For example:

```
CMS> HELP CREATE ELEMENT
```

To get help from the DECwindows Motif interface, see *Section 2.2, "CMS Menus"*.

1.5. Sample Session

This section contains a tutorial example showing how to use basic CMS features. The numbers in the example match the explanations at the end of the example.

```
Username: JONES  ❶
Password:
$ SET DEFAULT [JONES.PROJECT]  ❷
$ DIRECTORY  ❸
Directory DISKX:[JONES.PROJECT]CMDRMVGRO.BLI;1        CMDRMVGRO.SDML;1
     DIFF_DESIGN.MEM;2               INSTALL-VERSION.TXT;4
 INTERNAL_CUST_SITES.COM;6LOGIN.COM;71           MAIL_FIL_KEY.COM;6NOTES
$NOTEBOOK.NOTE;1                     V010-29_INSTALL.TXT;1  V050-
CALLABLE.LOG;2Total of 10 files.
$ CMS  ❹
CMS> CREATE LIBRARY [JONES.CMSLIB]  ❺
_Remark: creating new library for my project
%CMS-S-CREATED, CMS Library DISKX:[JONES.CMSLIB] created
%CMS-I-LIBIS, library is DISKX:[JONES.CMSLIB]
%CMS-S-LIBSET, library set
CMS> CREATE ELEMENT/KEEP *.*  ❻
_Remark: creating elements from default directory to new CMS lib
%CMS-S-CREATED, element DISKX:[JONES.CMSLIB]CMDRMVGRO.BLI created
%CMS-S-CREATED, element DISKX:[JONES.CMSLIB]CMDRMVGRO.SDML created
%CMS-S-CREATED, element DISKX:[JONES.CMSLIB]DIFF_DESIGN.MEM created
%CMS-S-CREATED, element DISKX:[JONES.CMSLIB]INSTALL-VERSION.TXT created
    .
    .
    .
%CMS-S-CREATED, element DISKX:[JONES.CMSLIB]V010-29_INSTALL.TXT created
%CMS-S-CREATED, element DISKX:[JONES.CMSLIB]V050-CALLABLE.LOG created
CMS> EXIT  ❼
$ LOGOUT
    .
    .
    .
$ CMS  ❽
CMS> SET LIBRARY [.CMSLIB]  ❾
%CMS-I-LIBIS, library is DISKX:[JONES.CMSLIB]
%CMS-S-LIBSET, library set
-CMS-I-SUPERSEDE, library list superseded
CMS> SHOW ELEMENT  ❿
Elements in CMS Library DISKX:[JONES.CMSLIB]
CMDRMVGRO.BLI        "creating elements from default directory to new CMS
 lib"
CMDRMVGRO.SDML       "creating elements from default directory to new CMS
 lib"
DIFF_DESIGN.MEM      "creating elements from default directory to new CMS
 lib"
INSTALL-VERSION.TXT  "creating elements from default directory to new CMS
 lib"
    .
    .
    .
V010-29_INSTALL.TXT  "creating elements from default directory to new CMS
 lib"
```

```
V050-CALLABLE.LOG    "creating elements from default directory to new CMS
 lib"
CMS> CREATE GROUP  ⓫
_Group name: USER_MANUAL
_Remark: creating group for the project user's manual
%CMS-S-CREATED, group DISKX:[JONES.CMSLIB]USER_MANUAL created
CMS> INSERT ELEMENT CMDRMVGRO.BLI,CMDRMVGRO.SDML USER_MANUAL  ⓬
_Remark: inserting the command routine files into group USER_MANUAL
%CMS-I-INSERTED, element DISKX:[JONES.CMSLIB]CMDRMVGRO.BLI inserted into
DISKX:[JONES.CMSLIB]group USER_MANUAL
%CMS-I-INSERTED, DISKX:[JONES.CMSLIB]element CMDRMVGRO.SDML inserted into
DISKX:[JONES.CMSLIB]group USER_MANUAL
%CMS-I-INSERTIONS, 2 insertions completed
CMS> CREATE ELEMENT CMS$$GSR.TXT/INPUT=DISK$$XXX:[PROJECT.PUBLIC]  ⓭
_Remark: also need the shareable image
%CMS-S-CREATED, element DISKX:[JONES.CMSLIB]CMS$$GSR.TXT created
CMS> INSERT ELEMENT CMS$$GSR.TXT USER_MANUAL  ⓮
_Remark: inserting the shareable image into group USER_MANUAL
%CMS-I-INSERTED, element DISKX:[JONES.CMSLIB]CMS$$GSR.TXT inserted into
DISKX:[JONES.CMSLIB]group USER_MANUAL
%CMS-I-INSERTIONS, 1 insertion completed
CMS> CREATE CLASS BASELEVEL1  ⓯
_Remark: creating class to contain files needed for base level 1
%CMS-S-CREATED, class DISKX:[JONES.CMSLIB]BASELEVEL1 created
CMS> RESERVE DIFF_DESIGN.MEM,USER_MANUAL "must add topics to these files"
  ⓰
%CMS-I-RESERVED, generation 1 of element DISKX:[JONES.CMSLIB]CMDRMVGRO.BLI
 reserved
%CMS-I-RESERVED, generation 1 of element DISKX:[JONES.CMSLIB]CMDRMVGRO.SDML
 reserved
%CMS-I-RESERVED, generation 1 of element DISKX:[JONES.CMSLIB]CMS$$GSR.TXT
 reserved
%CMS-I-RESERVED, generation 1 of element DISKX:[JONES.CMSLIB]DIFF_DESIGN
 reserved
%CMS-I-RESERVATIONS, 4 elements reserved
CMS> REPLACE CMS$$GSR.TXT "made two changes to table"  ⓱
%CMS-S-GENCREATED, generation 2 of element DISKX:[JONES.CMSLIB]CMS$$GSR.TXT
 created
CMS> SHOW GENERATION  ⓲
Element generations in CMS Library DISKX:[JONES.CMSLIB]
CMDRMVGRO.BLI     1   23-JAN-2005 17:45:46 JONES "creating elements from
           default directory to new CMS lib"
CMDRMVGRO.SDML    1   23-JAN-2005 17:46:47 JONES "creating elements from
           default directory to new CMS lib"
CMS$$GSR.TXT      2   23-JAN-2005 18:12:18 JONES "made two changes to
           table
     .
     .
     .
V050-CALLABLE.LOG 1   23-JAN-2005 17:49:47 JONES "creating elements from
           default directory to new CMS lib"CMS>
INSERT GENERATION CMS$$GSR.TXT/GEN=1,MAIL_FIL_KEY.COM BASELEVEL1  ⓳
_Remark: these generations needed in class to build baselevel1
%CMS-S-GENINSERTED, generation 1 of element DISKX:[JONES.CMSLIB]CMS$
$GSR.TXT
inserted into class DISKX:[JONES.CMSLIB]BASELEVEL1
%CMS-S-GENINSERTED, generation 2 of element DISKX:
[JONES.CMSLIB]MAIL_FIL_KEY.COM
```

```
inserted into class DISKX:[JONES.CMSLIB]BASELEVEL1

CMS> SHOW RESERVATIONS  ⑳
Reservations in CMS Library DISKX:[JONES.CMSLIB]
CMDRMVGRO.BLI     (1)  JONES   1   24-JAN-2005 17:45:46 "need to add
 merging to these files"
CMDRMVGRO.SDML     (1)  JONES   1   24-JAN-2005 17:46:47 "need to add
 merging to these files"
DIFF_DESIGN    (1)  JONES   1   24-JAN-2005 17:48:29 "need to add merging
 to these files"
CMS> SHOW GROUP USER_MANUAL/CONTENTS  ㉑
Groups in CMS Library DISKX:[JONES.CMSLIB]
USER_MANUAL      "creating group for the project user's manual"
    CMDRMVGRO.BLI
    CMDRMVGRO.SDML
    CMS$$GSR.TXT
CMS> EXIT  ㉒
$
```

Key to Example:

❶ User Jones logs in.

❷ Jones sets the default directory to the [.PROJECT] directory.

❸ Jones displays the default directory DISKX:[JONES.PROJECT].

❹ Jones invokes the CMS image and enters the CMS subsystem.

❺ Jones creates [.CMSLIB] with the CREATE LIBRARY command.

❻ Jones enters the CREATE ELEMENT command, and all files from the default directory [JONES.CMSLIB] are created as elements in the CMS library. The files are not deleted from Jones's default directory because the /KEEP qualifier was specified on the CREATE ELEMENT command.

❼ Jones exits from CMS and logs out.

❽ Jones later logs in and reenters CMS.

❾ Jones sets the library to [JONES.CMSLIB].

❿ Jones displays all elements with the SHOW ELEMENT command.

⓫ Jones creates a group named USER_MANUAL.

⓬ Jones then inserts the two elements CMDRMVGRO.BLI and CMDRMVGRO.SDML into the group USER_MANUAL.

⓭ Jones decides that an element from the project directory is needed, and specifies the /INPUT qualifier on the CREATE ELEMENT command to indicate that the element is located in a different directory from the default directory. Because Jones did not specify /KEEP, the file will be deleted from the project directory.

⓮ Jones then inserts the element into the group USER_MANUAL.

⓯ Jones creates the BASELEVEL1 class with the CREATE CLASS command.

⓰ Jones reserves the element DIFF_DESIGN.MEM and the group USER_MANUAL from the CMS library. CMS places the element DIFF_DESIGN.MEM and the contents (in this case, elements) of group USER_MANUAL in Jones's default directory. Jones can then modify these files as necessary.

17 Jones had previously reserved the element CMS$$GSR.TXT (which is part of the group USER_MANUAL), and made changes to that file. Jones replaces the element from the default directory [JONES] back into the CMS library [JONES.CMSLIB].

18 Jones enters the SHOW GENERATION command to display the last generation on the main line of descent for each element in the CMS library.

19 Jones then inserts generation 1 of the element CMS$$GSR.TXT and a generation of the element MAIL_FIL_KEY.COM into class BASELEVEL1. (If you do not specify the /GENERATION qualifier on an element, CMS uses the latest generation.)

20 Jones displays all current reservations.

21 Jones displays the contents of the group USER_MANUAL.

22 Jones exits from CMS.

1.6. Command Summary

Table 1.1, "CMS Command Summary" lists and briefly describes all CMS commands.

Table 1.1. CMS Command Summary

Command	Description
ACCEPT GENERATION	Changes the review status of one or more generations from pending to accepted and removes them from the review pending list.
ANNOTATE	Creates a listing file (element-name .ANN) that includes the element history and an annotated source listing.
CANCEL REVIEW	Changes the review status of one or more element generations from pending to none and removes them from the review pending list.
CONVERT LIBRARY	Converts libraries that were created with Version 2.*n* of CMS for use with Version 3.0 or higher.
COPY CLASS	Copies one or more existing classes (including generation history and file attributes) to form one or more new classes.
COPY ELEMENT	Copies one or more existing library elements (including history and file attributes) to form one or more new elements.
COPY GROUP	Copies one or more existing groups (including history and file attributes) to form one or more new groups.
CREATE CLASS	Establishes one or more classes. Once a class is established, any set of element generations can be placed in that class with the INSERT GENERATION command.

Command	**Description**
CREATE ELEMENT	Establishes one or more new elements in a CMS library by moving one or more files into the CMS library. By default, CMS deletes all copies of the input file after creating the element.
CREATE GROUP	Establishes one or more groups. Once a group is established, any set of elements or groups can be placed in that group with the INSERT ELEMENT or INSERT GROUP command.
CREATE LIBRARY	Creates one or more CMS libraries by loading one or more empty directories with CMS control structures.
DELETE CLASS	Deletes one or more classes from the library.
DELETE ELEMENT	Deletes one or more elements from the library.
DELETE GENERATION	Deletes one or more generations from one or more elements in the library.
DELETE GROUP	Deletes one or more groups from the library.
DELETE HISTORY	Deletes some or all of the library history.
DIFFERENCES	Compares the contents of two files and creates a listing file (filename.DIF) showing all the lines that differ. DIFFERENCES can also compare element generations in a CMS library, or a file to an element generation.
DIFFERENCES/CLASS	Compares the contents of two classes and creates a listing file (classname.DIF) showing all member generations that differ between classes.
FETCH	Retrieves a copy of one or more specified element generations.
HELP	Provides online CMS help.
INSERT ELEMENT	Places one or more elements in one or more groups.
INSERT GENERATION	Places one or more element generations in one or more classes.
INSERT GROUP	Places one or more groups in another group or groups.
MARK GENERATION	Changes the review status of one or more generations to pending and adds them to the review pending list.
MODIFY CLASS	Changes the attributes of a class from those established with the CREATE CLASS command, or with a previous MODIFY CLASS command.
MODIFY ELEMENT	Changes the attributes of one or more elements from those established with the CREATE ELEMENT command, or with a previous MODIFY ELEMENT command.

Command	Description
MODIFY GENERATION	Changes the attributes of one or more generations from those established with the CREATE ELEMENT or REPLACE command, or with a previous MODIFY GENERATION command.
MODIFY GROUP	Changes the attributes of one or more groups from those established with the CREATE GROUP command, or with a previous MODIFY GROUP command.
MODIFY LIBRARY	Changes the attributes of the library from those established with the CREATE LIBRARY command, or with a previous MODIFY LIBRARY command.
MODIFY RESERVATION	Changes the remark associated with a specific reservation of an element.
REJECT GENERATION	Changes the review status of one or more generations from pending to rejected and removes them from the review pending list.
REMARK	Enters a remark in the library history.
REMOVE ELEMENT	Removes one or more elements from one or more groups.
REMOVE GENERATION	Removes one or more generations from one or more classes.
REMOVE GROUP	Removes one or more groups from another group or groups.
REPLACE	Returns the most recent version of one or more reserved generations to the library, thus creating a new generation of each element. The reservation ends, and CMS deletes all versions of the input file.
RESERVE	Delivers a copy of one or more generations and marks them as reserved.
RETRIEVE ARCHIVE	Delivers a copy of one or more generations from one or more archive files created with the DELETE GENERATION/ARCHIVE command.
REVIEW GENERATION	Associates a review comment with one or more generations that are currently under review.
SET ACL	Manipulates access control lists on various objects in the CMS library.
SET LIBRARY	Identifies one or more existing CMS libraries so subsequent CMS commands refer to the specified library or libraries.
SET NOLIBRARY	Removes one or more libraries from the current library search list.
SHOW ACL	Displays the access control list associated with one or more specified objects.

Command	Description
SHOW ARCHIVE	Displays information about the contents of one or more archive files created with the DELETE GENERATION/ARCHIVE command.
SHOW CLASS	Displays one or more established classes.
SHOW ELEMENT	Displays information about one or more elements.
SHOW GENERATION	Displays a listing of one or more established generations.
SHOW GROUP	Displays a listing of one or more established groups.
SHOW HISTORY	Displays a chronological listing of all CMS transactions that have affected the library.
SHOW LIBRARY	Displays the current library directory specification or list of library directory specifications.
SHOW RESERVATIONS	Displays a listing of all current reservations and concurrent replacements.
SHOW REVIEWS_PENDING	Displays a listing of generations that currently have reviews pending, and any associated review remarks.
SHOW VERSION	Displays the version number of your CMS system.
UNRESERVE	Cancels an existing reservation.
VERIFY	Performs a series of consistency checks on your CMS library to confirm that all elements are present and stored properly.

Chapter 2. Using CMS with DECwindows Motif

This chapter describes how you use CMS with the DECwindows Motif interface. It describes how to invoke CMS in the DECwindows Motif environment, get help, and display information. The chapter also shows a sample session.

Before continuing with this chapter, you should be familiar with how to start a DECwindows Motif desktop session, use and manage windows, and run a DECwindows Motif application.

2.1. Invoking CMS

To invoke the CMS DECwindows Motif interface, enter the following command:

```
$ CMS/INTERFACE=DECWINDOWS
```

The following section describes the available menus and menu options.

2.2. CMS Menus

You use CMS menus to access buttons and dialog boxes that let you create and open libraries, manipulate elements and generations, and perform other CMS operations.

The main CMS window contains the following menus:

	File	Data
	Edit	Options
	View	Help
	Maintenance	

2.2.1. File Menu

Choose the File menu items to perform the following operations:

- New – Create either a new library or a new element.

  The New Library... and New Element... items in the submenu invoke dialog boxes that prompt you for information to create the new objects.

- Open... – Establish an existing CMS library as the current library. The library is automatically inserted into the library search list.

  When you choose Open..., a dialog box is displayed, enabling you to specify options for the library to be opened. You can specify multiple libraries to be opened by separating each library with a comma.

- Fetch... – Retrieve a copy of the specified object or objects from the CMS library. The object can be an element name, group name (CMS will fetch the most recent generations of elements in the group), wildcard expression, or a list of these separated by commas.

  CMS fetches the most recent generation on the main line of descent, unless you fill in the Generation field. The Fetch function delivers a copy of the specified generations to your current, default directory. The generation is not reserved, and CMS does not allow you to replace it. CMS allows you to fetch a generation that is already reserved, and notifies you of any current generation reservations for the element. If a version of a file with the same name as the element already exists in your current, default directory when you execute the fetch transaction, CMS notifies you. A new version is then created with the next higher version number.

- Reserve... – Retrieve a copy of the specified object from the CMS library and mark it as reserved. The object can be an element, group, wildcard expression, or a list of these separated by commas.

  The Reserve function places a copy of the object in your current default directory and marks the object as reserved. When you choose the Reserve menu item, a dialog box is displayed, enabling you to specify options for the reserve transaction.

- Replace... – Return an element reservation to the library and create a new generation of the element. The replace transaction transfers a file from your default directory to the current CMS library, thus creating a new generation.

- Unreserve... – Cancel one or more reservations of a generation of an element. You cannot unreserve a generation held by another user unless you hold BYPASS privilege, or unless you are granted BYPASS access to the element by an access control list (ACL).

  If you have more than one reservation of an element or if you are canceling another user's reservation, you must specify the exact reservation to be canceled. You do this by using either the Generation option or the Reservation Identification button in the Options dialog box (choose the Options button).

- Close... – Remove one or more libraries from the current library search list.

- Set Directory... – Specify a default device and directory to be automatically used for file input and output.

  CMS uses the default device and directory you specify for the duration of the current CMS session. Your initial default device and directory are restored when you exit from the CMS session.

- Close View... – Close the current view window when there are multiple CMS views open.

  The Close View menu item is located in both the File and View menus, and has the same function in both menus.

- Exit – End the CMS session.

2.2.2. Edit Menu

Choose the Edit menu items to perform the following clipboard operations:

- Copy – Move the selected text to the clipboard. The Copy function does not alter any information.

- Select All – Select the entire contents of the window, not just the data currently visible in the window.

2.2.3. View Menu

Choose the View menu items to display CMS library objects and information about those objects. The options are as follows:

- Element – Display all the elements in the current library. CMS lists the elements in alphabetical order.

Double click on a specific element to expand it to display the list of generations that belong to the element.

If you have opened multiple libraries, CMS displays the name of each library in the library list. You must separately expand each library into its elements.

- Group – Display all the groups in the current library. CMS lists the groups in alphabetical order.

  Double click on a specific group to expand it to display the list of elements and other groups contained in the group.

  If you have opened multiple libraries, CMS displays the name of each library in the library list. You must separately expand each library into its groups.

- Class – Display all the classes in the current library. CMS lists the classes in alphabetical order.

  Double click on a specific class to expand it to display the list of generations contained in the class.

  If you have opened multiple libraries, CMS displays the name of each library in the library list. You must separately expand each library into its classes.

- Reservation – Display elements and generations that are reserved in the current library.

  Double click on a reserved element to expand it into individual generation reservations. Double click on a reserved generation to expand it into the following information:

  - Reservation identification number
  - Name of the user who has it reserved
  - Generation number
  - Date, time, and remark associated with the reservation

  If you have opened multiple libraries, CMS displays only the name of each library in the library list. You must separately expand each library into its reserved elements and generations.

- History – Display a chronological list of the transactions performed in the library. Each history record contains the following information:

  - Date and time of the transaction
  - User name of the user who performed the action
  - Transaction that was performed
  - Name of the element and generation number
  - Remark associated with the transaction

  If you have opened multiple libraries, CMS displays the name of each library in the library list. You must separately expand each library into its history records.

- Review – Display a list of all elements and generations that currently have reviews pending in the library.

  Double click on a specific element under review to expand it into its individual generations under review. Double click on a generation to expand it into a list of review comments, if any. CMS displays the following information:

  - Generation number of the element
  - Name of the user who placed the element under review
  - Date, time, and remark associated with the element or generation under review

  If you have opened multiple libraries, CMS displays the name of each library in the library list. You must separately expand each library into its review elements.

- Command – Display a list of every CMS command on which an ACL can be placed.

  Double click on a specific command to expand it to display the ACL assigned to the command, if any.

  If you have opened multiple libraries, CMS displays the name of each library in the library list. You must separately expand each library into its commands.

You can also use the View menu items to do the following:

- Expand and collapse – Expand and collapse an object's children, attributes, ACL, group or class membership list, or all of the above options.
- New – Display a list of all available types of views. You can choose one of the following types:

  - Element
  - Group
  - Class
  - Reservation
  - History
  - Review
  - Command

  CMS displays a new view window that contains the type of view you chose.

  To close a window when you have multiple view windows open, pull down the View menu, then choose Close View. The Close View menu item is located in both the File and View menus, and has the same function in both menus.

- Restrict... – Control the contents of the current view. The Restrict View options are equivalent to options available when using command-line interface SHOW commands.

  When you choose Restrict..., a dialog box is displayed, enabling you to specify an object (or objects) to be displayed with the options you specify.

To view an object other than the object type in your current view, change the view by choosing a different type of view; or pull down the New View submenu, choose the desired view, then choose the Restrict... menu item.

- Unrestrict... – Control the contents of the current view. The Unrestrict View options are equivalent to options available when using command-line interface NOSHOW commands.

  When you choose Unrestrict..., a dialog box is displayed, enabling you to specify an object (or objects) to be restricted with the options you specify.

  To view an object other than the object type in your current view, change the view by choosing a different type of view; or pull down the New View submenu, choose the desired view, then choose the Unrestrict... menu item.

- Close View – Close the current view window when there are multiple CMS views open.

  The Close View menu item is located in both the File and View menus, and has the same function in both menus.

- Update – Collapse previously expanded objects and display an updated view window, including any changes made to the library search list.

2.2.4. Maintenance Menu

Choose the Maintenance menu items to perform the following operations:

- Insert – Display a list of the following types of objects on which you can perform insertion transactions:
  - Elements
  - Groups
  - Generations
- Remove – Display a list containing the following types of objects on which you can perform remove transactions:
  - Elements
  - Groups
  - Generations
- New – Display a list containing the following types of objects on which you can operate:
  - Groups
  - Classes
- Copy – Display a list containing the following types of objects which you can copy:
  - Elements
  - Groups

  - Classes

  This menu enables you to copy one or more existing objects and create a new object (or objects) in the same library or another library. The original object is left unchanged.

  If you copy an object to the same library, the object must have a different name.

- Modify – Display a list containing the following types of objects whose characteristics can be changed:

  - Elements
  - Groups
  - Classes
  - Generations
  - Libraries
  - Reservation

- Delete – Display a list containing the following types of objects that can be deleted:

  - Elements
  - Groups
  - Classes
  - Generations
  - History

- Verify... – Instruct CMS to perform a series of consistency checks on your CMS library. CMS verifies libraries to confirm that the library structure and library files are in a valid form.

  By default, CMS verifies all the elements in each library in the library search list. To specify that CMS verify only the first occurrence of each element in the search list, use the Occlude option.

  When you choose Verify..., a dialog box is displayed, enabling you to specify options for the library to be verified.

- Review... – Communicate information about the status of generations of elements. Using the Review menu item, you can mark a generation to be examined and commented on by other team members.

  The generation can then be accepted, rejected, or the review canceled. To display pending reviews, do the following:

  1. Pull down the View menu.
  2. Choose the Review submenu.

  You can also pull down the View... menu item again and choose the Restrict... menu item. This enables you to restrict the information displayed by the Review View.

You can view remarks made by other users by doing the following:

1. Click on an element generation.
2. Pull down the View menu.
3. Pull down the Expand submenu.
4. Choose the Children menu item.

- Remark... – Add a remark to the library history. The remark is recorded in the library history in the following format: `date time username REMARK "remark"`

  Use the remark to describe a transaction. You can use any characters; however, the length of the remark cannot exceed 256 characters.

- Set ACL... – Manipulate the ACL on various objects in the library. An ACL consists of access control entries (ACEs) that grant or deny access to a command or other object to specified users.

  Generally, there are two ways in which you can use ACLs on objects:

  - To control and restrict access to commands
  - To control and restrict access to other objects (elements, groups, classes, the element list, the group list, the class list, library history, and library attributes)

When you choose one of these menu items, a dialog box is displayed, enabling you to view and specify options for that operation.

2.2.5. Data Menu

Choose the Data menu item to perform the following operations:

- Element/File Differences... – Compare two files, two generations of elements, or a file and a generation of an element.

  If CMS finds differences, it creates a file that contains the lines that differ between them, and delivers a copy of the file to your current, default directory.

  If the files are the same, CMS issues a message to that effect and does not create a differences file. If you have turned off the Differences Only button, CMS creates a file, even if there are no differences.

- Class Differences... – Compare the member generations between two classes.

  If CMS finds differences, it creates a file that lists the members that differ between them, and delivers a copy of the file to your current, default directory.

  If the classes are the same, CMS issues a message to that effect and does not create a differences file. If you have turned off the Differences Only button, CMS creates a file, even if there are no differences.

- Annotate... – Create a line-by-line file listing of the changes made to each specified element generation. CMS places this file in your current, default directory or a directory you specify.

The Annotate function documents the development of an element, and creates an output file that contains an annotated listing. Unless you specify a different name, CMS names the file the same as the element name. The file type is .ANN. The annotated listing file contains two parts:

- History – Includes the generation number, date, time, user, and remark associated with each generation of the element
- Source file listing – Lists all the lines inserted or modified from generation 1 to the specified generation.

When you choose one of these menu items, a dialog box is displayed, enabling you to view and specify options for that operation.

2.2.6. Options Menu

Choose the Options menu items to perform the following operations:

- Show Command... – Enter CMS command-line commands at the CMS prompt. The output appears in the CMS Command window.

  When you choose the Show Command... menu item, a dialog box is displayed with a display window, a smaller input window containing the CMS prompt (CMS>), and the Clear Command Window and Cancel buttons.

- Message Logging... – Direct CMS to display error, success, and informational messages using the options you choose.

- Initial Library... – Specify a library or libraries to be automatically opened each time you invoke CMS.

  When you choose Initial Library..., a dialog box is displayed, enabling you to specify one or more library specifications.

- Known Libraries... – Specify multiple library names that CMS stores and displays each time you enter CMS.

  When you enter CMS, the libraries you specified are shown in the Open Library dialog box.

- View... – Specify the default view you want displayed each time you invoke CMS. You can specify one of the following views:

  - Element
  - Group
  - Class
  - Reservation
  - History
  - Review
  - Command

You can also specify the style in which CMS displays the view (textual, outline, or tree) or to invoke views on fetch and reserve operations from LSE.

- Default Occlusion... – Set default occlusion information.
- Restrict... – Customize the display of options for the view type you specify. Using the Restrict submenu is equivalent to using command-line interface SHOW commands.

  Use the Restrict submenu to restrict the display of options *before* you display a view.
- Save Settings, Restore Settings, Restore System Settings – Enable systemwide defaults, thus overriding any current customizations.

Note

Some customizations take effect immediately, whereas others might take effect the next time you invoke the associated view. Still others take effect the next time you invoke the CMS DECwindows interface (for example, the Initial Library menu item).

2.2.7. Help Menu

You obtain help in the DECwindows Motif environment by pulling down the Help menu. Help provides brief information about screen objects, concepts, and tasks that you can perform in CMS.

The CMS DECwindows Motif interface has online help that provides complete information on all screen objects, including scroll bars, icons, menus, dialog boxes, text fields, buttons, and functions. The online help is context-sensitive. To get online help, do the following:

1. Position the pointer on the desired object.
2. Press and hold the Help key while you press MB1.
3. Release both keys.

A Help window opens to display information about the object.

2.3. Displaying CMS Information in DECwindows Motif

You display and obtain information about CMS objects through views. Views replace the CMS SHOW commands.

In the DECwindows Motif environment, CMS provides the following types of views:

- Element
- Group
- Class
- Reservation

- History
- Review
- Command

When you invoke the CMS DECwindows Motif interface for the first time, CMS displays an Element View. This is a view of all elements in your current library. However, if you have opened multiple libraries, CMS displays each library name. To obtain a different view, do the following:

1. Pull down the View menu.
2. Choose the desired view.

CMS displays the appropriate view for the type you choose. For example, if you choose a group view, CMS displays the names of all the groups in the library. However, if you have more than one library open, CMS displays only each library name. You must then expand each library into the groups it contains.

2.3.1. Displaying More Than One View

A single view can display only one type of information at a time; however, you can display multiple view windows. To obtain multiple view windows, do the following:

1. Pull down the View menu.
2. Choose the New menu item; the New submenu is displayed.
3. Choose the desired view.

CMS displays an additional window with the view you choose.

You can display any number of views that you want; each view is independent of other views. By using CMS views, you can choose objects on which you want to perform functions.

2.3.2. Restricting Views

You can restrict views to display objects meeting certain criteria. For example, to restrict a Reservation View to display only reservations made by a particular user, do the following:

1. Pull down the View menu.
2. Choose the Reservation menu item.
3. Pull down the View menu.
4. Choose the Restrict... menu item.

A dialog box is displayed, enabling you to specify the user name for which CMS should display reservations.

2.3.3. Customizing Your Initial View

CMS enables you to customize your CMS session by specifying which view you want displayed on startup. To customize your CMS session, do the following:

1. Pull down the Options menu.
2. Choose the View... menu item.
3. Choose the desired view.
4. Pull down the Options menu.
5. Choose the Save Attributes menu item.

You can also obtain information about CMS objects by expanding them. See *Section 2.3.4, "Expanding and Collapsing CMS Objects"* for more information.

2.3.4. Expanding and Collapsing CMS Objects

The CMS DECwindows Motif interface provides the following ways to expand and choose objects:

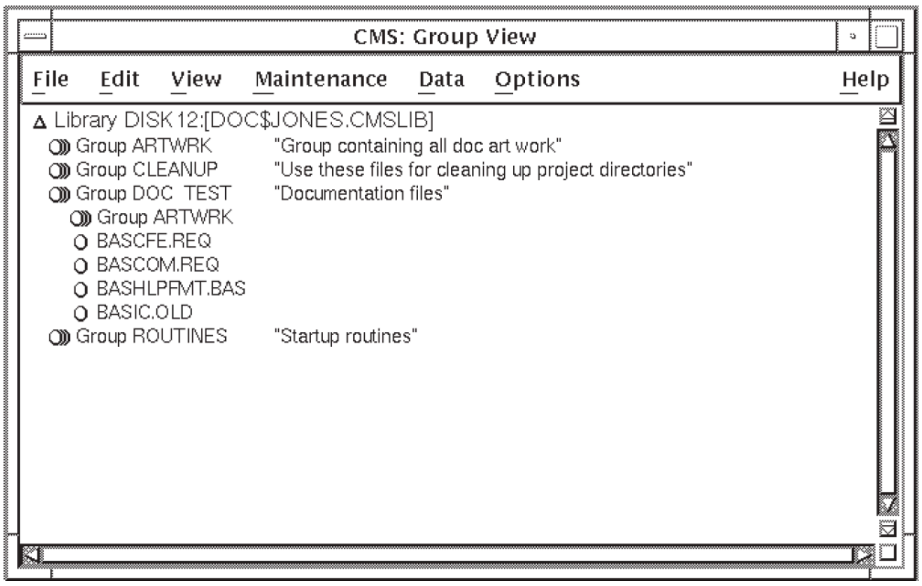
- Double click on an object to expand it.
- Choose a menu item, then specify the name of the object in the associated dialog box. Or, first click on an object and then choose a menu item and provide information about it in the associated dialog box.
- Click on an object, then press MB3 to obtain a pop-up menu.

The following sections describe these methods.

2.3.4.1. Double Clicking

Figure 2.1, "Expanding a Group" shows the group DOC_TEST expanded to show its children.

Figure 2.1. Expanding a Group



```
Δ Library DISK12:[DOC$JONES.CMSLIB]
   Group ARTWRK       "Group containing all doc art work"
   Group CLEANUP      "Use these files for cleaning up project directories"
   Group DOC TEST     "Documentation files"
     Group ARTWRK
     BASCFE.REQ
     BASCOM.REQ
     BASHLPFMT.BAS
     BASIC.OLD
   Group ROUTINES     "Startup routines"
```

To expand this group using double clicking, do the following:

1. Pull down the View menu.

2. Choose the Group menu item.

3. Double click on the desired group (in this example, group DOC_TEST), or choose the Expand item from the View menu.

Group DOC_TEST expands into its components, including elements and any other groups contained in group DOC_TEST. Double clicking on the element BASCOM.REQ expands it into its generations.

Note

If an item is expanded fully, double clicking collapses the information into the previous level of information.

You can also expand an object by choosing a function. For example, to expand the group DOC_TEST, do the following:

1. Click on the desired object (in this example, group DOC_TEST).

2. Pull down the View menu.

3. Choose the Expand menu item; the Expand submenu is displayed.

4. Choose the Children submenu item.

Section 2.3.4.2, "Choosing a Function" contains more information about choosing a function.

2.3.4.2. Choosing a Function

Most of the functions performed on CMS objects are grouped into two menus: File and Maintenance. You use the File menu to manipulate library and element activities, such as creating new libraries or fetching, reserving, replacing, or creating new elements. You use the Maintenance menu to perform organizational or maintenance operations on libraries and library elements. These include modifying elements, inserting elements into or removing elements from various groups or classes, and so on.

To choose an object and perform a specific operation, use one of the following methods:

- Click on an object, then choose a menu item and provide information about the object in the associated dialog box. For example, to reserve an element, do the following:

  1. Click on an element.

  2. Pull down the File menu.

  3. Choose the Reserve... menu item.

  A dialog box is displayed, with the name of the element you have chosen in the Selected list box. You can then enter additional information about the element and the reserve function, and click on the OK button.

- Choose a menu item, then specify the name of the object in the associated dialog box. For example, to reserve an element, do the following:

  1. Pull down the File menu.

  2. Click on the Reserve... menu item.

3. Click on the Element field in the Reserve... dialog box.

4. Fill in the Element field with the name of the element you want to reserve.

   You can then enter additional information about the element and the reserve function, and click on the OK button.

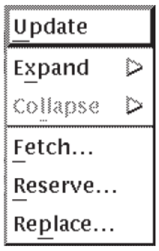
2.3.4.3. Using the Pop-Up Menu

CMS provides a pop-up menu enabling you to quickly access some of the most commonly used CMS functions. You can use the pop-up menu with any CMS object that can be used in those functions.

To get the pop-up menu, press and hold MB3. Or, to first choose an object for the operation, click on the object, then press and hold MB3 to get the pop-up menu.

Figure 2.2, "CMS Pop-Up Menu" shows the pop-up menu.

Figure 2.2. CMS Pop-Up Menu



2.4. DECwindows LSE/CMS Integration

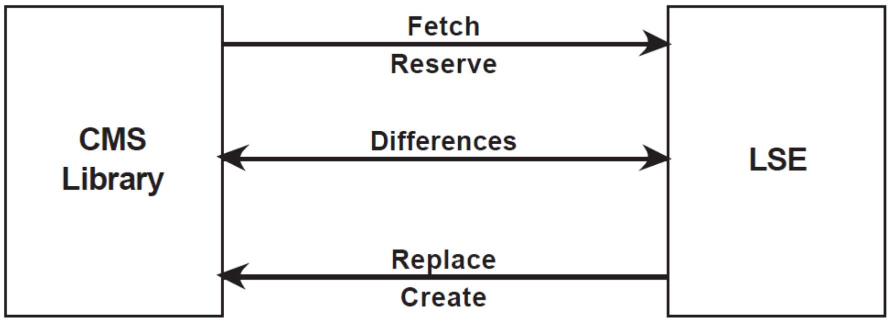
LSE is integrated with CMS to ease the management of source code between the two DECset components, as follows:

- From within LSE, you can enter commands or select menu choices to manipulate CMS elements, LSE buffers, or disk files.

- From within CMS, you can select menu choices to manipulate CMS elements, LSE buffers, or disk files.

As shown in *Figure 2.3, "LSE/CMS Integration"*, you can fetch and reserve an element from a CMS library, edit the file or perform differences, or replace or create a file to the library.

Figure 2.3. LSE/CMS Integration



2.4.1. CMS Functions from LSE

From LSE, you can perform all CMS operations on the CMS library of your choice. To see which library is set, issue the following command:

```
LSE> CMS SHOW LIBRARY
```

The following example shows a typical system response that might appear in the LSE buffer:

```
Your CMS library list consists of:   DISK11:[EXCERPTS.CMS.VMS]
```

To reset to another CMS library, issue a command similar to the following:

```
LSE> CMS SET LIBRARY DISK11:[SUMMARIES.CMS.VMS]
```

The following list further describes the major CMS functions available from LSE integrated functions, and presents examples:

- Reserve (and unreserve) an element in the CMS library into an LSE buffer. Use the LSE File menu or the LSE command line. The following example shows how to reserve generation 12 of a CMS element at the LSE command line.

  ```
  LSE> CMS RESERVE COPY.PAS/GENERATION=12
  ```

- Replace an element into the CMS library from an LSE buffer. Use the LSE File menu or the LSE command line. The following example shows how to replace a CMS element at the LSE command line.

  ```
  LSE> CMS REPLACE COPY.PAS "Nov 2005 update"
  ```

- Perform CMS differences between any combination of CMS elements and disk files or between CMS classes, putting the results into a disk file. Use the LSE command line only.

  The following example shows how to perform differences between generation 15of a CMS element and version 2 of the related disk file at the LSE command line.

  ```
  LSE> CMS DIFFERENCES COPY.PAS/GENERATION=15 COPY.PAS;2
  ```

All other CMS operations are available via the LSE command-line interface.

2.4.2. LSE Functions from CMS

From the CMS File pull-down menu, you can perform the following operations:

- Create an element in the CMS library from the current LSE buffer.
- Fetch a generation of an element from the CMS library into an LSE buffer.
- Reserve an element in the CMS library into an LSE buffer.
- Replace an element into the CMS library from an LSE buffer.
- Perform CMS differences between two generations of elements.
- Perform CMS differences between the current LSE buffer and its corresponding CMS element or a file.

- Perform CMS differences between two classes.

From the CMS Options pull-down menu, you can perform the following operations:

- Use a view to fetch a generation of an element from the CMS library into an LSE buffer.
- Use a view to reserve an element in the CMS library into an LSE buffer.

2.4.3. Creating an Element in the CMS Library

There are two ways to create an element in the CMS library:

- *Create the CMS element in CMS.*

  You can create an element in the CMS library from the contents of the current LSE buffer via the CMS pull-down menu. If the specified file name is currently in an LSE buffer, the LSE buffer is written to disk and that version of the CMS element is created.

  To create an element, do the following:

  1. From the CMS File menu, choose New, Element to access the New Element dialog box.
  2. Specify an LSE edit buffer in the Element text field, or accept the current LSE edit buffer (if LSE is running). The buffer name must be a valid CMS element name, as defined in *Section 10.2.4, "Element Names"*. If the text in the current LSE buffer has changed from an older version on disk, a new version is written to disk at the same time the new element is created in the CMS library.

     You can also import a file by clicking the Input File toggle button and typing a file name in the associated text field. The Input File and Element choices are mutually exclusive. That is, you can specify either an input file or an LSE edit buffer, but not both.

- *Create the CMS element in LSE.*

  To create a CMS element from the LSE command line, issue a command similar to the following:

  ```
  LSE> CMS CREATE ELEMENT COPY.PAS "Dec 2005 update"
  ```

  If the specified file is in the current LSE buffer and is different from the latest version on disk, a new version is saved and that version is used for the CMS element.

2.4.4. Fetching a Generation of an Element From the CMS Library

To fetch a generation of a CMS element, use one of the following methods:

- *Fetch a CMS element using CMS.*

  You can fetch a generation of an element from the CMS library into an LSE buffer via the CMS File menu, or by double clicking on the element name. If DECwindows LSE is running, the specified CMS element is fetched and displayed in an LSE buffer with the same name.

- *Fetch a CMS element using LSE.*

To fetch a CMS element in LSE, either select Fetch from the File menu, or issue a command similar to the following at the command line:

```
LSE> CMS FETCH COPY.PAS "Modify Dec 2005 update"
```

To fetch an earlier generation, specify the /GENERATION= *n* qualifier. If DECwindows LSE is running, the specified CMS element is fetched and displayed in an LSE buffer with the same name. For a different file name, use the /OUTPUT=*file-spec* qualifier.

2.4.5. Reserving an Element in the CMS Library

To reserve or unreserve a CMS element, use one of the following methods:

- *Reserve a CMS element using CMS.*

  You can reserve an element in the CMS library into an LSE buffer via the CMS File menu. If DECwindows LSE is running, the specified CMS element is reserved and displayed in an LSE buffer with the same name.

- *Reserve a CMS element using LSE.*

  To reserve a CMS element in LSE, either select Reserve from the File menu, or issue a command similar to the following at the command line:

  ```
  LSE> CMS RESERVE COPY.PAS "Modify Dec 2005 update"
  ```

  To reserve an earlier generation, specify the /GENERATION= *n* qualifier. If DECwindows LSE is running, the specified CMS element is reserved and displayed in an LSE buffer with the same name. For a different file name, use the /OUTPUT=*file-spec* qualifier.

2.4.6. Replacing an Element into the CMS Library

To replace a CMS element, use one of the following methods:

- *Replace a CMS element using CMS.*

  You can replace an element into the CMS library from an LSE buffer via the CMS File menu. If the specified file name is currently in an LSE buffer, the LSE buffer is written to disk and that version of the CMS element is replaced. To replace the element, do the following:

  1. From the CMS File menu, choose Replace to access the Replace dialog box.
  2. Verify the selected CMS elements in the Selected list box, change element names or generations, or click on Cancel to return to the CMS window. If you did not select an element in the CMS window, the cursor is displayed in the Element text field for your input. However, if you did select an element, the Element text field is unavailable.

  To avoid creating a new generation if the input file has no changes from the reserved generation, activate the Create New Generation Only if Changed toggle button in the Replace Options dialog box.

- *Replace a CMS element using LSE.*

  To replace a CMS element in LSE, either select Replace from the File menu, or issue a command similar to the following at the command line:

```
LSE> CMS REPLACE COPY.PAS "Dec 2005 update, mod. Jan 2006"
```

To avoid creating a new generation if the input file has no changes from the reserved generation, use the /IF_CHANGED qualifier.

2.4.7. Performing CMS Differences Operations

To perform a CMS differences operation, use one of the following methods:

- *Perform CMS differences using CMS.*

  Access either the Element/File Differences or Class Differences dialog box from the Data pull-down menu in DECwindows CMS. Use the Primary Input region to identify the first item to be compared, and the Secondary Input region for the second item. Select the two items using the following buttons and fields:

  - *Selected*—If you selected a CMS object, that object is displayed in the Selected field of the Primary Input region. You can either accept that object or specify another object. If you did not select a CMS object, the Selected area is inactive.

    When comparing elements, CMS uses the highest mainline generation (1+) by default, unless you selected a specific generation. To compare any other generation, supply the exact generation number in the form ELEMENT\n.

  - *Generation (Element differences only)*—To compare a CMS element, click the Generation toggle button and enter the generation value in the text field.

  - *Element/File (Element differences only)*—Click on the Element/File label and specify either an OpenVMS file specification or a CMS element. The OpenVMS file can be specified without a version number, but a CMS element *must* be specified with a generation number. For a CMS element, click the Generation button to specify that the file is an element and not an OpenVMS file. If no CMS element was selected, the text field remains blank in both the Primary Input and Secondary Input areas.

- *Perform CMS differences using LSE.*

  Enter the command and specify which items should be used to perform the operation. Enter only the file name for an LSE buffer, add version numbers for disk files, or add the /GENERATION= *n* qualifier for CMS elements, as shown in the following examples:

  - This performs differences between generation 2 of a CMS element and version 8 of a disk file.

    ```
    LSE> CMS DIFFERENCES COPY.PAS/GENERATION=2 COPY.PAS;8
    ```

  - This performs differences between generation 2 of a CMS element and generation 1 of the same CMS element.

    ```
    LSE> CMS DIFFERENCES COPY.PAS/GENERATION=2 COPY.PAS/GENERATION=1
    ```

2.5. CMS Command Correspondence

Most command-line interface CMS commands have a corresponding menu path in the DECwindows Motif interface. However, the following CMS commands are not included in the CMS DECwindows Motif interface:

- CONVERT LIBRARY
- RETRIEVE ARCHIVE
- SHOW ARCHIVE
- SHOW LIBRARY

You can invoke the CMS command line from the DECwindows Motif interface by entering command mode. To enter command mode, do the following:

1. Pull down the Options menu.
2. Choose the Show Command... menu item.

A dialog box is displayed, containing an output window and the CMS command-line prompt. Enter CMS command-line interface commands at the CMS prompt (CMS>). CMS displays the resulting command output in the output window (see *Figure 2.4, "Command Mode"*).

2.6. Small Screen Support

The CMS DECwindows Motif interface enables you to change the default values for window sizes and font sizes, so you can view all the CMS information even on a small PC screen.

The CMS default values for text fonts, window sizes, and other window resources are contained in the file CMS$DW_DEFAULTS.DAT in the directory DECW$SYSTEM_DEFAULTS. Create a local copy of this file to the directory DECW$USER_DEFAULTS, then read the value descriptions in the resource file and modify the defaults to your preferences.

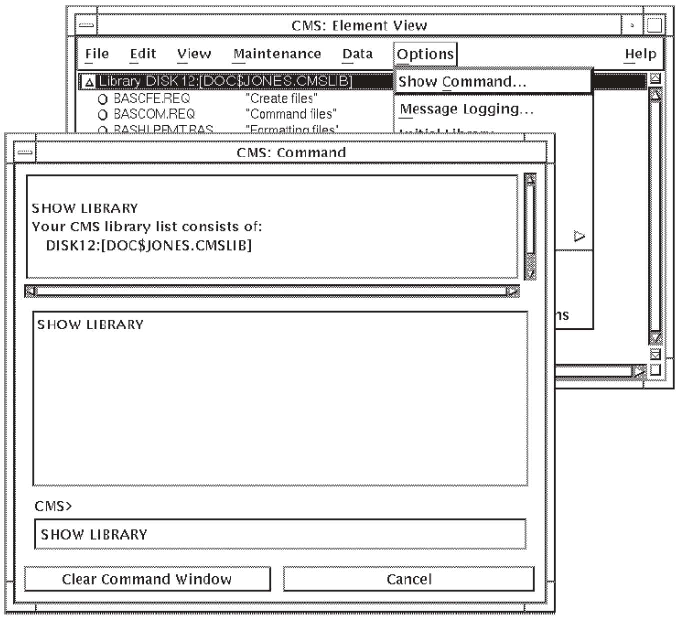
2.7. Customizing Your CMS DECwindows Motif Interface

The CMS DECwindows Motif interface enables you to conveniently customize many options, including the following:

- Message-logging options
- The initial library to open each time you enter CMS
- A library (or libraries) that are most commonly used, which you can specify once and then conveniently open by choosing them from a list
- The default view to be displayed each time you enter CMS
- The default occlusion
- Default restrictions for each view type

Figure 2.4, "Command Mode" shows the Command dialog box. The command SHOW LIBRARY has been entered at the CMS command-line prompt (CMS>), and the resulting information is displayed in the output box.

Figure 2.4. Command Mode



```
Library DISK12:[DOC$JONES.CMSLIB]
  BASCFE.REQ      "Create files"
  BASCOM.REQ      "Command files"
  BASHLPRMT.BAS   "Formatting files"
```

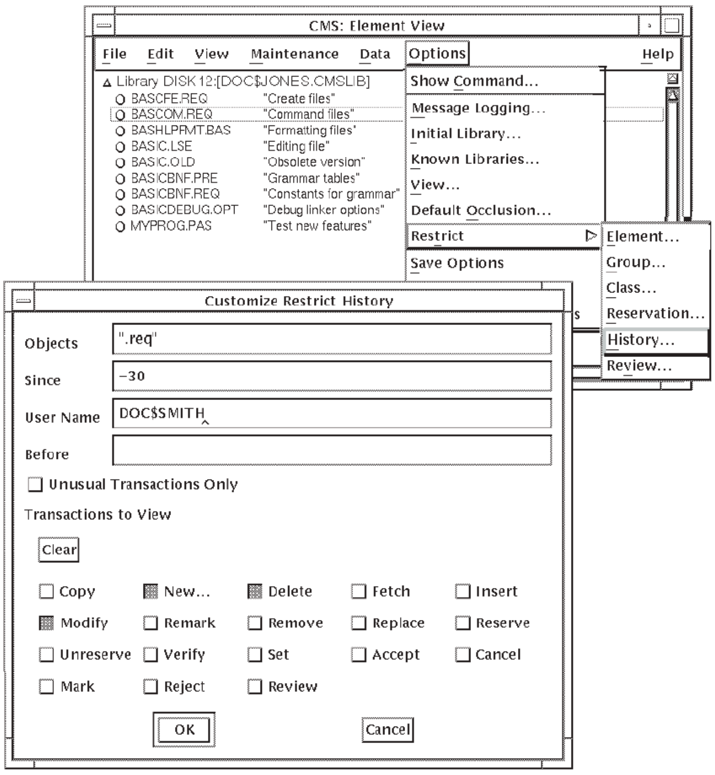
```
SHOW LIBRARY
Your CMS library list consists of:
  DISK12:[DOC$JONES.CMSLIB]
```

```
SHOW LIBRARY
```

CMS> SHOW LIBRARY

In *Figure 2.5, "Restricting History"*, the Customize Restrict History dialog box is shown. The user has specified that the history view contain only elements with the file type .REQ that have been modified, created, or deleted in the last 30 days by user SMITH.

Figure 2.5. Restricting History



```
Library DISK12:[DOC$JONES.CMSLIB]
  BASCFE.REQ      "Create files"
  BASCOM.REQ      "Command files"
  BASHLPRMT.BAS   "Formatting files"
  BASIC.LSE       "Editing file"
  BASIC.OLD       "Obsolete version"
  BASICBNF.PRE    "Grammar tables"
  BASICBNF.REQ    "Constants for grammar"
  BASICDEBUG.OPT  "Debug linker options"
  MYPROG.PAS      "Test new features"
```

Objects	".req"
Since	-30
User Name	DOC$SMITH
Before	

☐ Unusual Transactions Only

☐ Copy	▨ New...	▨ Delete	☐ Fetch	☐ Insert
▨ Modify	☐ Remark	☐ Remove	☐ Replace	☐ Reserve
☐ Unreserve	☐ Verify	☐ Set	☐ Accept	☐ Cancel
☐ Mark	☐ Reject	☐ Review		

Chapter 3. Libraries

A CMS library consists of a set of defined objects that can be operated on by CMS commands. A CMS library resides in a directory that has been initialized for use solely by CMS.

This chapter describes how to create and use CMS libraries, control occlusion of CMS objects, and library locking.

3.1. Creating Libraries

This section describes how to create a CMS library. First, you must create a directory to contain the library; then you create the library, and create elements in it.

You can also optionally create a reference copy directory. A reference copy directory is a directory used for storing copies of the latest generation on the main line of descent for specified elements in a CMS library. See *Section 3.1.4, "Creating a Reference Copy Directory"* for more information.

3.1.1. Creating the Directory

You create a directory to contain your CMS library by using the DCL command CREATE/ DIRECTORY. The format of the command is as follows: `CREATE/DIRECTORY directory-specification`

For example:

```
$ CREATE/DIRECTORY [PROJECT.CMSLIB]
```

The name PROJECT identifies the first-level directory. This command creates the empty subdirectory [.CMSLIB] within the directory [PROJECT]. For more information on the CREATE/DIRECTORY command, see the *VSI OpenVMS DCL Dictionary*.

To create a first-level directory, you must have write access to the master file directory (MFD) on the volume on which you are creating the directory. Normally, on a system volume, only users with a system user identification code (UIC) or the SYSPRV or BYPASS user privilege are allowed write access to the MFD to create a first-level directory. To create a subdirectory, you must have write access to the next higher directory level. For more information on directory specifications, see *Chapter 10, "Command Syntax"*.

Note

You should not place any version limit on a CMS library; CMS automatically purges and deletes unused files within a library. A library must have a file retention count of at least 2 to allow error recovery in case of system failure.

CMS limits directory trees to a depth of eight. Because CMS might create subdirectories, you should not create a library in an eighth-level directory.

If you want to place access control lists (ACLs) on the library directory, you should do so before you create the library, so files created during library creation are assigned the correct protection. See *Chapter 7, "Security Features"* for more information on ACLs.

3.1.2. Creating the Library

You create a CMS library with the CREATE LIBRARY command. The CREATE LIBRARY command creates CMS control files in the specified directory. The directory must exist and must be empty. Once you create a library in a directory, CMS uses that directory to locate and store files. Note that your default directory cannot be a CMS library. After you create a library with the CREATE LIBRARY command, all subsequent CMS commands refer to this library until the end of the terminal session, until you specify a different library with the SET LIBRARY command, or until you deassign the library list with the SET NOLIBRARY command.

The following command initializes a library in the empty directory [PROJECT.CMSLIB]:

```
$ CMS CREATE LIBRARY [PROJECT.CMSLIB]
_Remark: test procedure library
%CMS-S-CREATED, CMS library DISKX:[PROJECT.CMSLIB] created
%CMS-I-LIBIS, Library is DISKX:[PROJECT.CMSLIB]
%CMS-S-LIBSET, CMS library set
```

Caution

Once the library is created, you should access it only through a CMS interface. If a library has been accessed by means other than CMS, such as copying the file through a DCL command, it might result in unrecoverable library corruption. Files that have been placed into the library directory by means other than CMS can be deleted by CMS when the library is verified and repaired (see *Chapter 9, "Library Maintenance"*).

You can create more than one library with the CREATE LIBRARY command by specifying a list of directory specifications separated by commas. For more information, see *Section 3.2, "Using Libraries"*.

The CREATE LIBRARY command also allows you to optionally specify a directory to be used for maintaining reference copies of library elements. For information about using reference copy directories, see *Section 3.1.4, "Creating a Reference Copy Directory"*.

3.1.3. Creating Elements in the Library

You store a file in a CMS library with the CREATE ELEMENT command. CREATE ELEMENT uses the input file you provide to create the first version of an element. This first version represents generation 1 of the element. An element represents all the versions of a particular file as it is developed. Every element in the CMS library must have a unique name.

The following is an example of the CREATE ELEMENT command:

```
$ CMS CREATE ELEMENT OUTPUT.FOR "ascii output format routines"
%CMS-S-CREATED, element [PROJECT.CMSLIB]OUTPUT.FOR created
```

This command creates the element named OUTPUT.FOR. Generation 1 of element OUTPUT.FOR now exists in the library.

The file specified in the CREATE ELEMENT command must be present in your current, default directory (unless you specify a different location by using the /INPUT qualifier). CMS deletes all copies of that file from the default or specified directory after creating the new element. You can override this default by specifying the /KEEP or /RESERVE qualifier on the CREATE ELEMENT or MODIFY ELEMENT command, or library-wide by specifying the /KEEP qualifier on the CREATE LIBRARY or MODIFY LIBRARY command. The contents of the file used to create the element become generation 1 of that element.

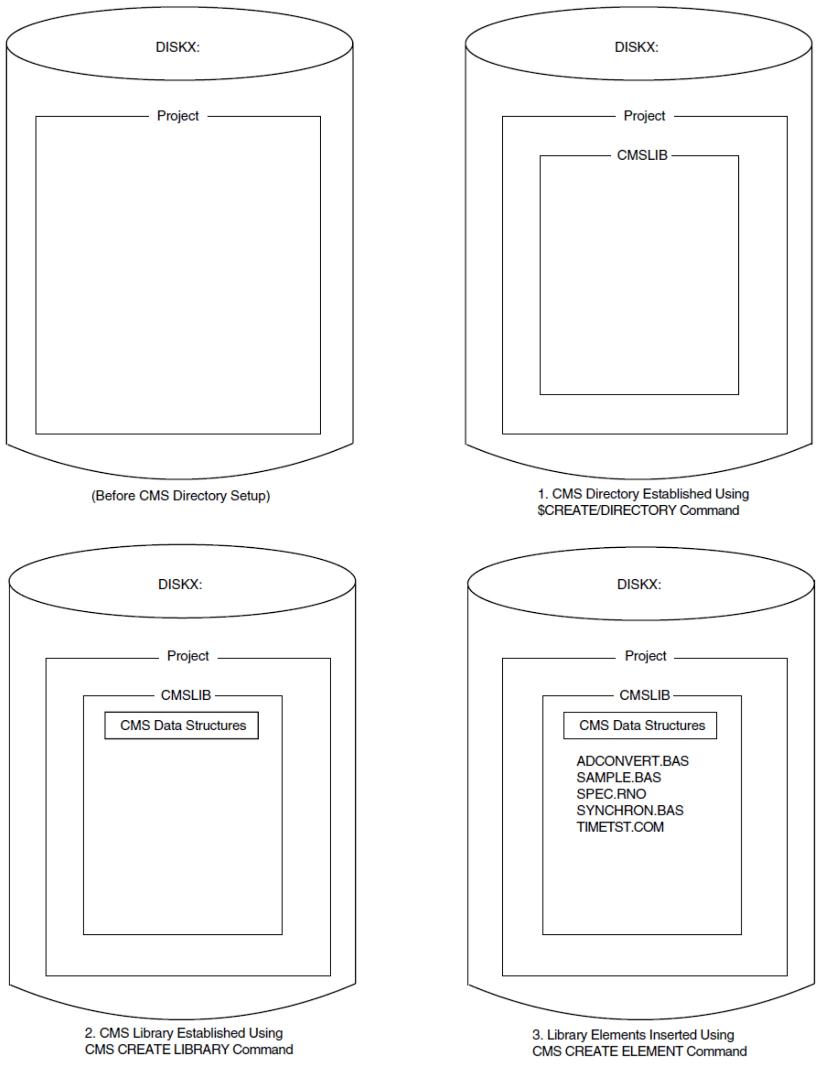
CMS can store and operate on nontext files; however, CMS cannot store directory files.

There is no explicit limit on the number of elements (or groups or classes) that can exist in a library. However, there might be limits imposed by your system configuration, including system, process, disk space, and virtual memory limitations.

To create an element in the library, you must have read access to the file from which you are creating the element.

Figure 3.1, "Building a CMS Library" shows the process of establishing a library and creating elements in it. See *Chapter 4, "Elements and Generations"* for more information about elements.

Figure 3.1. Building a CMS Library


DISKX:
Project
(Before CMS Directory Setup)
DISKX:
Project
CMSLIB
1. CMS Directory Established Using
$CREATE/DIRECTORY Command
DISKX:
Project
CMSLIB
CMS Data Structures
2. CMS Library Established Using
CMS CREATE LIBRARY Command
DISKX:
Project
CMSLIB
CMS Data Structures
ADCONVERT.BAS
SAMPLE.BAS
SPEC.RNO
SYNCHRON.BAS
TIMETST.COM
3. Library Elements Inserted Using
CMS CREATE ELEMENT Command

3.1.4. Creating a Reference Copy Directory

A **reference copy directory** is a directory in which CMS maintains a copy of the latest generation on the main line of descent of each element.

The reference copy directory cannot be a CMS library, nor can it be a subdirectory of a CMS library directory. Although CMS allows different libraries to be assigned the same reference copy directory, it is strongly recommended that you assign each CMS library its own unique reference copy directory.

To establish a reference copy directory, first create a directory (see *Section 3.1.1, "Creating the Directory"*), then use the /REFERENCE_COPY qualifier with the CREATE LIBRARY or MODIFY LIBRARY command. The /REFERENCE_COPY qualifier directs CMS to store the name of this directory in the library, creating a permanent association between the CMS library and this directory (unless you enter a MODIFY LIBRARY/NOREFERENCE_COPY command, which removes this association). For example:

```
$ CREATE/DIRECTORY [PROJECT.CMSLIB]
$ CREATE/DIRECTORY [PROJECT.REFCOPY]
$ CMS CREATE LIBRARY [PROJECT.CMSLIB]/REFERENCE_COPY=[PROJECT.REFCOPY]
_Remark: Master library with reference copies
%CMS-S-CREATED, library DISKX:[PROJECT.CMSLIB] created
```

In this example, the first CREATE/DIRECTORY command creates the CMS library directory [PROJECT.CMSLIB]; the second CREATE/DIRECTORY command creates the reference copy directory [PROJECT.REFCOPY]. The CREATE LIBRARY command initializes a CMS library in the [PROJECT.CMSLIB] directory and creates a permanent association between the CMS library and the [PROJECT.REFCOPY] directory.

Once a reference copy directory is established for a library, CMS maintains reference copy files in that directory. Every time you create a new main-line generation of an element (by using CREATE ELEMENT or REPLACE), CMS updates the reference copy of that element. Existing elements in the library will not have the **reference copy** attribute set. Use the /REFERENCE_COPY qualifier on the MODIFY ELEMENT command to enable the **reference copy** attribute on those elements for which reference copies are to be maintained. See *Section 4.5.3, "The Reference Copy Attribute"* for more information.

The following example assigns an existing CMS library a reference copy directory and creates reference copies for existing elements:

```
$ CREATE/DIRECTORY [PROJECT.REFCOPY]
$ CMS
CMS> SET LIBRARY [PROJECT.CMSLIB]
CMS> MODIFY LIBRARY/REFERENCE_COPY=[PROJECT.REFCOPY]
_Remark: Establish reference copy directory
CMS> MODIFY ELEMENT/REFERENCE_COPY *.* "enable reference copy"
```

The MODIFY LIBRARY command establishes the directory [PROJECT.REFCOPY] as the reference copy directory for the current CMS library [PROJECT.CMSLIB]. The MODIFY ELEMENT command changes the **reference copy** attribute for all currently existing elements and creates reference copies for them. Use the SHOW LIBRARY/FULL command to display the directory specification of a reference copy directory.

If you do not want some elements to have reference copies, modify those elements with the / NOREFERENCE_COPY qualifier on the MODIFY ELEMENT command. For more information, see the descriptions of the CREATE ELEMENT and MODIFY ELEMENT commands in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

CMS does not create reference copies for any variant generations. CMS maintains a reference copy only of the latest generation on the main line of descent of each element.

3.2. Using Libraries

When you invoke CMS, you must explicitly set up a library environment to tell CMS which library (or libraries) you want to use. This sets a library search list. You do this by either creating a new library or libraries (with the CREATE LIBRARY command), or by selecting an existing library or libraries (with the SET LIBRARY command).

A CMS library search list is a list of one or more libraries. When CMS operates on multiple libraries, it accesses them in the order you specified when you set the library list. If you invoke CMS and do not establish at least one library in the library search list, you receive an error indicating that your library environment is undefined and your library search list is empty.

Libraries in the library search list do not need to be related; each library is complete and self-contained. You can specify libraries in any order, but a library can appear only once in the library search list. You can specify a maximum of 128 libraries in a library search list.

After you establish a library search list, that list remains in effect for all further CMS commands until you modify it with the CREATE LIBRARY command or SET [NO]LIBRARY command, or log out.

3.2.1. Setting Libraries

You set one or more libraries by entering the SET LIBRARY command, or the CREATE LIBRARY command, which performs an implicit SET LIBRARY operation. The SET LIBRARY command defines a DCL logical name, CMS$LIB, which points to the library or libraries you have selected. After you have selected a library or libraries, all CMS commands you enter refer to the CMS$LIB library list. The library list exists until you enter another SET LIBRARY, SET NOLIBRARY, or CREATE LIBRARY command, or log out.

You can enter one or more library directory names as a parameter to the SET LIBRARY command. For example, to set your library to [PROJECT.CMSLIB], enter the following command:

```
$ CMS SET LIBRARY [PROJECT.CMSLIB]
%CMS-I-LIBIS, library is DISKX:[PROJECT.CMSLIB]
%CMS-S-LIBSET, library set
%CMS-I-SUPERSEDE, library list superseded
```

This command sets (or resets) the library search list to contain only the library [PROJECT.CMSLIB].

To set your library search list to contain more than one library, you would specify multiple libraries separated by commas. For example:

```
CMS> SET LIBRARY [PROJECT1.CMSLIB],[PROJECT3.CMSLIB]
%CMS-I-LIBIS, library is DISKX:[PROJECT1.CMLSIB]
%CMS-I-LIBINSLIS, library DISKX:[PROJECT3.CMSLIB] inserted at end of
 library list
%CMS-S-LIBSET, library set
```

This command sets (or resets) the library search list to contain only the two libraries [PROJECT1.CMSLIB] and [PROJECT3.CMSLIB].

If you try to set your library to a directory that has not been initialized by CREATE LIBRARY, CMS $LIB becomes undefined and CMS issues a warning message.

3.2.2. Modifying Library Lists

To add libraries to an established library search list, use the /BEFORE and /AFTER qualifiers on the CREATE LIBRARY or SET LIBRARY command to control the placement of the new libraries in the existing library search list.

For example:

```
CMS> CREATE LIBRARY [PROJECT1.CMSLIB],[PROJECT3.CMSLIB]
CMS> SET LIBRARY [PROJECT2.CMSLIB]/BEFORE=[PROJECT3.CMSLIB]
```

In this example, the CREATE LIBRARY command establishes the library search list consisting of the two libraries [PROJECT1.CMSLIB] and [PROJECT3.CMSLIB]. The SET LIBRARY command inserts the library [PROJECT2.CMSLIB] in the library search list. The library search list now consists of three libraries: [PROJECT1.CMSLIB], [PROJECT2.CMSLIB], and [PROJECT3.CMSLIB], in that order.

If you specify either the /BEFORE or the /AFTER qualifier without a value, the new library (or libraries) are added to the existing search list either before or after the entire library list, respectively.

If you do not specify either qualifier, a new library search list is created, replacing the entire existing library search list.

To remove one or more libraries from the existing search list, use the SET NOLIBRARY command. The SET NOLIBRARY command accepts one or more library directory specifications, which are then removed from the list, leaving the rest of the list intact. If you do not specify a library directory, every library from the entire library search list is removed, and the library search list becomes undefined. For more information on the SET NOLIBRARY command, see the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

3.3. Controlling Occlusion in Multiple Libraries

CMS operates on your library search list by searching through the library (or libraries) in the list. If you have more than one library in the search list, CMS searches the libraries one at a time in the order they appear in the search list, until a specified object is found. Once the object is found, CMS performs the specified operation on the object. By default, CMS does not continue to search for the object in any of the remaining libraries.

Objects with the same name can exist in more than one library. When an object exists in more than one library of a library search list, CMS processes only the first occurrence of the specified object and ignores any later instances of that object in subsequent libraries. This behavior is **occlusion**; that is, the first instance of the object occludes any subsequent instances of that object. For example:

```
CMS> SET LIBRARY [BOOK.CMSLIB],[EXAMPLES.CMSLIB],[TEMP.CMSLIB]
CMS> FETCH TESTBAS.SDML "fetch first instance"
```

In this example, CMS searches the library list, starting with the library [BOOK.CMSLIB], then [EXAMPLES.CMSLIB], then [TEMP.CMSLIB]. When CMS locates the element TESTBAS.SDML, it fetches the element from the first library in the list in which it finds it. For example, if the element TESTBAS.SDML existed in [EXAMPLES.CMSLIB] and in [TEMP.CMSLIB], CMS would fetch the element only from [EXAMPLES.CMSLIB], because that element would occlude the element in [TEMP.CMSLIB].

You control occlusion with the /OCCLUDE qualifier. The /OCCLUDE qualifier has the following format: `/OCCLUDE[=options,...]`

You can specify the following options with this qualifier:

- [NO]CLASS—Controls occlusion for classes
- [NO]ELEMENT—Controls occlusion for elements
- [NO]GROUP—Controls occlusion for groups

- [NO]OTHER—Controls occlusion for library attributes, history, commands, the class list, the element list, and the group list
- ALL
- NONE

You can specify either ALL or NONE, or one or more of the remaining keywords in any combination. If you do not specify a keyword on the /OCCLUDE qualifier, the default is /OCCLUDE=ALL. The ALL keyword enables occlusion for all four object types; the NONE keyword disables occlusion for all four object types. To disable occlusion for a specific object, use the /OCCLUDE qualifier with a negated keyword. For example:

```
$ CMS SET LIBRARY [WORK.CMSLIB],[PROJECT.CMSLIB]
%CMS-I-LIBIS, library is DISKX:[WORK.CMSLIB]
%CMS-I-LIBINSLIS, library DISKX:[PROJECT.CMSLIB] inserted at end of library
 list
%CMS-S-LIBSET, library set
%CMS-I-SUPERSEDE, library list superseded


$ CMS FETCH SAMPLE.PAS/OCCLUDE=NOELEMENT "fetch all instances"
%CMS-S-FETCHED, generation 1 of element DISKX:[WORK.CMSLIB]SAMPLE.PAS
 fetched
%CMS-I-FILEXISTS, file already exists, DISKX:[WORK]SAMPLE.PAS;2 created
%CMS-S-FETCHED, generation 1 of element DISKX:[PROJECT.CMSLIB]SAMPLE.PAS
 fetched
```

This command produces two copies of SAMPLE.PAS—the latest generation from library [WORK.CMSLIB], and the latest generation from library [PROJECT.CMSLIB].

Note that first CMS fetches the first instance of the file SAMPLE.PAS and then fetches the second instance, which creates a newer version of SAMPLE.PAS.

3.3.1. Occlusion of Multiple Object Types in a Command

Many CMS commands allow you to specify more than one object type on a command line. For instance, you can specify an element specification consisting of a list of element names and group names, as in the following example:

```
$ CMS FETCH CODE,BUILD.COM "fetch all code and the build procedure"
```

In this example, the group CODE represents all program source modules, and the element BUILD.COM is the build procedure to compile and link the program.

You can also specify a command in which objects of one type are inserted into or removed from objects of a different type. For example:

```
$ CMS INSERT ELEMENT MAIN.BAS CODE "insert main module into CODE group"
```

This command inserts the element MAIN.BAS into the group CODE.

When you specify multiple object types in a CMS command, CMS simultaneously performs occlusion on all applicable objects. Specifically, in the preceding example, CMS simultaneously performs occlusion on the elements and the groups, and ignores occlusion for other object types. A special case occurs when you use a group name as an element specification. In this case, the elements in the group occlude subsequent instances of those elements (if element occlusion is enabled). In such cases, CMS performs element occlusion even if the specification contained only group names.

The following two examples show the difference between using a group name as an element specification and using a comma-separated list of the same element names that are in the group. In these examples, the default directory is [WORK], the current library search list is set to [WORK.CMSLIB], [PROJECT.CMSLIB]; the group SAMPLES is in [PROJECT.CMSLIB] and contains the elements SAMPLE.PAS and SAMPLE.DAT. The library [WORK.CMSLIB] does not contain any groups, but does contain the same elements. The examples correspond with *Figure 3.2, "Library Occlusion"*.

```
CMS> FETCH SAMPLES
_Remark: fetch 1st instance of element generations from group SAMPLES
%CMS-I-FETCHED, generation 2 of element DISKX:[PROJECT.CMSLIB]SAMPLE.DAT
 fetched
%CMS-I-FETCHED, generation 2 of element DISKX:[PROJECT.CMSLIB]SAMPLE.PAS
 fetched
%CMS-I-FETCHES, 2 elements fetched

CMS> FETCH/OCCLUDE=NOELEMENT SAMPLES
_Remark: fetch all instances of element generations from group SAMPLES
%CMS-I-FETCHED, generation 2 of element DISKX:[PROJECT.CMSLIB]SAMPLE.DAT
 fetched
%CMS-I-FETCHED, generation 2 of element DISKX:[PROJECT.CMSLIB]SAMPLE.PAS
 fetched
%CMS-I-FETCHES, 2 elements fetched
```

In this case, the group SAMPLES is used as the element specification. Note that the /OCCLUDE qualifier has no effect; two elements are fetched in each case. Although the two elements SAMPLE.DAT and SAMPLE.PAS exist in the first library [WORK.CMSLIB], they are not fetched because CMS looks for the elements in group SAMPLES, which is in the second library [PROJECT.CMSLIB]. Because there are no further occurrences of SAMPLES, the two elements in the second library are fetched.

```
CMS> FETCH SAMPLE.DAT,SAMPLE.PAS
_Remark: fetch first instance of sample elements
%CMS-I-FETCHED, generation 1 of element DISKX:[WORK.CMSLIB]SAMPLE.DAT
 fetched
%CMS-I-FETCHED, generation 1 of element DISKX:[WORK.CMSLIB]SAMPLE.PAS
 fetched
%CMS-I-FETCHES, 2 elements fetched\
CMS> FETCH/OCCLUDE=NOELEMENT SAMPLE.DAT,SAMPLE.PAS
_Remark: fetch all instances of sample elements
%CMS-I-FILEXISTS, file already exists, DISKX:[WORK]SAMPLE.DAT;2 created
%CMS-I-FETCHED, generation 1 of element DISKX:[WORK.CMSLIB]SAMPLE.DAT
 fetched
%CMS-I-FILEXISTS, file already exists, DISKX:[WORK]SAMPLE.DAT;2 created
%CMS-I-FETCHED, generation 1 of element DISKX:[WORK.CMSLIB]SAMPLE.PAS
 fetched
%CMS-I-FILEXISTS, file already exists, DISKX:[WORK]SAMPLE.DAT;3 created
%CMS-I-FETCHED, generation 1 of element DISKX:[PROJECT.CMSLIB]SAMPLE.DAT
 fetched
%CMS-I-FILEXISTS, file already exists, DISKX:[WORK]SAMPLE.PAS;3 created
%CMS-I-FETCHED, generation 1 of element DISKX:[PROJECT.CMSLIB]SAMPLE.PAS
 fetched
%CMS-I-FETCHES, 4 elements fetched
```

In the first command, CMS assumes /OCCLUDE=ALL, and fetches only the first instance of each of the elements SAMPLE.DAT and SAMPLE.PAS. In the second command, because the /

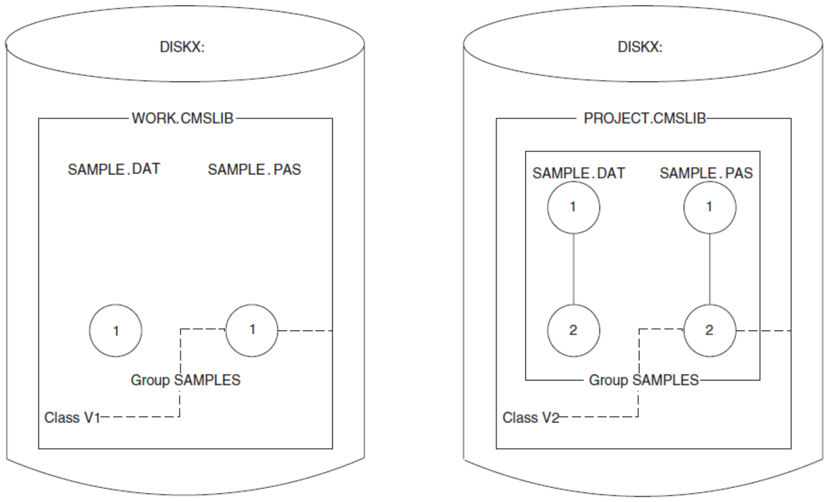
OCCLUDE=NOELEMENT qualifier was specified, CMS fetches all occurrences of each element from both libraries.

Thus, an element specification consisting of a group containing a set of elements, and an element specification consisting of a list of the same set of elements, are not equivalent. In the first case, CMS locates the first instance of the group. Previous instances of the elements in the group might exist in an earlier library, but are not selected because they are not located in the library with the specified group. In the second case, the first instance of each of the specified elements is found, regardless of which library they might be in. In fact, they might be found in libraries in the list prior to the library in which the group is found.

3.3.2. Examples

The following examples show how to control occlusion on various CMS objects. For the following examples, assume the library is set to [WORK.CMSLIB],[PROJECT.CMSLIB]. The library [WORK.CMSLIB] contains the two elements SAMPLE.DAT and SAMPLE.PAS, with generation 1 of the element SAMPLE.PAS inserted into class V1. The library [PROJECT.CMSLIB] contains the two elements SAMPLE.DAT and SAMPLE.PAS. These two elements are inserted into the group SAMPLES. Generation 2 of element SAMPLE.PAS is inserted into class V2. *Figure 3.2, "Library Occlusion"* matches the following examples.

Figure 3.2. Library Occlusion



1.
```
$ CMS FETCH SAMPLE.PAS "fetch the first instance"
%CMS-S-FETCHED, generation 1 of element DISKX:[WORK.CMSLIB]SAMPLE.PAS
 fetched
%CMS-S-FETCHES, 1 element fetched
```

   In this example, CMS assumes the default value of the /OCCLUDE qualifier (/OCCLUDE=ALL), and fetches only the first instance of SAMPLE.PAS.

2.
```
$ CMS FETCH/OCCLUDE=NOELEMENT SAMPLE.PAS "fetch all instances"
%CMS-I-FILEXISTS, file already exists, DISKX:[WORK]SAMPLE.PAS;2 created
%CMS-S-FETCHED, generation 1 of element DISKX:[WORK.CMSLIB]SAMPLE.PAS
 fetched
%CMS-I-FILEXISTS, file already exists, DISKX:[WORK]SAMPLE.PAS;3 created
%CMS-S-FETCHED, generation 2 of element DISKX:[PROJECT.CMSLIB]SAMPLE.PAS
 fetched
%CMS-S-FETCHES, 2 elements fetched
```

In this example, because the /OCCLUDE qualifier is specified with the negated keyword NOELEMENT, CMS retrieves all (both) instances of SAMPLE.PAS. Note that the second instance of SAMPLE.PAS is fetched into the next higher version of the output file, which is then placed into your default directory.

3.
   ```
   $ CMS FETCH SAMPLE.PAS/GENERATION=V1 "default occlusion"
   %CMS-S-FETCHED, generation 1 of element DISKX:[WORK.CMSLIB]SAMPLE.PAS
    fetched
   %CMS-S-FETCHES, 1 element fetched

   $ CMS FETCH SAMPLE.DAT/GENERATION=V1 "SAMPLE.DAT not in class V1"
   %CMS-E-NOFETCH, error fetching element DISKX:[WORK.CMSLIB]SAMPLE.DAT
   -CMS-E-GENNOTFOUND, generation V1 of DISKX:[WORK.CMSLIB]SAMPLE.DAT not
    found
   %CMS-E-ERRFETCHES, 0 elements fetched and 1 error occurred

   $ CMS FETCH SAMPLES/GENERATION=V1
   _Remark: SAMPLES group not in 1st library where class V1 is located
   %CMS-E-NOFETCH, error fetching element SAMPLES
   -CMS-E-NOTFOUND, Group SAMPLES not found
   %CMS-E-ERRFETCHES, 0 elements fetched and 1 error occurred

   $ CMS FETCH SAMPLE.PAS/GENERATION=V2 "element found but not class"
   %CMS-E-NOFETCH, error fetching element DISKX:[WORK.CMSLIB]SAMPLE.PAS
   -CMS-E-GENNOTFOUND, generation V2 of DISKX:[WORK.CMSLIB]SAMPLE.PAS not
    found
   -CMS-E-NOTFOUND, Class DISKX:[WORK.CMSLIB]V2 not found
   %CMS-E-ERRFETCHES, 0 elements fetched and 1 error occurred

   $ CMS FETCH/OCCLUDE=NOELEMENT SAMPLE.PAS/GENERATION=V2
   _Remark: element found but not class
   %CMS-E-NOFETCH, error fetching element DISKX:[WORK.CMSLIB]SAMPLE.PAS
   -CMS-E-GENNOTFOUND, generation V2 of DISKX:[WORK.CMSLIB]SAMPLE.PAS not
    found
   -CMS-E-NOTFOUND, Class DISKX:[WORK.CMSLIB]V2 not found
   %CMS-S-FETCHED, generation 2 of element DISKX:[PROJECT.CMSLIB]SAMPLE.PAS
    fetched
   %CMS-E-ERRFETCHES, 1 element fetched and 1 error occurred)
   ```

   This example shows occlusion when multiple object types are present. Classes V1 and V2 exist in the first and second libraries, respectively.

   Note that in the last case, an error diagnostic is generated when the first instance of SAMPLE.PAS is found and the class V2 is not found; when both the element and the class are found, the specified generation is successfully fetched.

4.
   ```
   $ CMS VERIFY SAMPLE.DAT/OCCLUDE=NOELEMENT
   %CMS-I-VERCLS, class list verified
   %CMS-I-VERCMD, command list verified
   %CMS-I-VERELE, element list verified
   %CMS-I-VERGRP, group list verified
   %CMS-I-VERRES, reservation list verified
   %CMS-I-VERFRE, internal free space list verified
   %CMS-I-VERFRE, internal free space list verified
   %CMS-I-VERFRE, internal free space list verified
   %CMS-I-VERFRE, internal free space list verified
   %CMS-I-VERFRE, internal free space list verified
   ```

```
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERARC, archive control block verified
%CMS-I-VER2, internal contiguous space verified
%CMS-I-VERCON, control file verified
%CMS-I-VEREDF, element DISKX:[WORK.CMSLIB]SAMPLE.DAT verified
%CMS-I-VEREDFS, element data files verified
%CMS-S-VERIFIED, library DISKX:[WORK.CMSLIB] verified
%CMS-I-VERCLS, class list verified
%CMS-I-VERCMD, command list verified
%CMS-I-VERELE, element list verified
%CMS-I-VERGRP, group list verified
%CMS-I-VERRES, reservation list verified
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERFRE, internal free space list verified
%CMS-I-VERARC, archive control block verified
%CMS-I-VER2, internal contiguous space verified
%CMS-I-VERCON, control file verified
%CMS-I-VEREDF, element DISKX:[PROJECT.CMSLIB]SAMPLE.DAT verified
%CMS-I-VEREDFS, element data files verified
%CMS-S-VERIFIED, library DISKX:[PROJECT.CMSLIB] verified
```

In this example, because the /OCCLUDE qualifier is specified with the negated keyword NOELEMENT, CMS verifies both libraries and both instances of the element SAMPLE.DAT.

3.4. Library Locking

CMS allows multiple read operations in the library at the same time. Read operations are operations that do not change any information in the library; for example, ANNOTATE, SHOW commands, SET LIBRARY, FETCH without a remark, and DIFFERENCES. Read operations allow users to use any combination of these commands in the library without interfering with each other in any way.

CMS controls concurrent access to the library by using the OpenVMS locking mechanism. The locking mechanism does not allow multiple write or multiple read and write operations in the library at the same time. Write operations are operations that change information in the library; for example, CREATE, FETCH with a remark, INSERT, MODIFY, REMOVE, SET ACL, RESERVE, and REPLACE.

When CMS has locked the library during a write operation, any access attempts made by other users are not allowed until the write operation is complete.

If the library remains locked for an extended period, CMS periodically issues messages informing you that the library is still in use. Your command is processed as soon as the lock preventing your library access is released.

When you enter a command, CMS attempts to lock the library only for the appropriate type of access for that command. If no other locks prevent your lock from being granted, you gain immediate access to the library, and your command is processed. If the library is locked for an access type incompatible with that required by your command (for example, a user entered a SHOW GENERATION command that locks the library for read access, and you enter a REPLACE command that locks the library for write access), CMS informs you that the library is locked and issues the following error message:

```
%CMS-I-INUSE, library library-specification is in use, please wait
```

CMS processes read and write attempts on the library in order.

Assume the following series of actions:

- User A has currently entered a command causing a library lock for read access (such as a SHOW GENERATION command).
- User B enters a command requiring write access (such as a REPLACE command), thus causing CMS to lock out user B.
- User C then enters a command requiring read access (such as an ANNOTATE command).

CMS will not process user C's command until user B's command has been processed, even though the current library lock (user A's read lock) allows user C's command to gain access to the library. This prevents a chain of compatible lock requests from locking an incompatible lock request out of the library for a prolonged period of time.

If your command cannot gain access to the library after 15 minutes, the waiting loop expires and CMS issues a message requesting that you try the command again later. You can use Ctrl/C at any point to abort the command.

Chapter 4. Elements and Generations

A CMS library is a collection of files, which represent elements and element generations. An element is the basic structural unit in a library. An element consists of one file and all its versions, called generations. This chapter describes elements and their generations in detail.

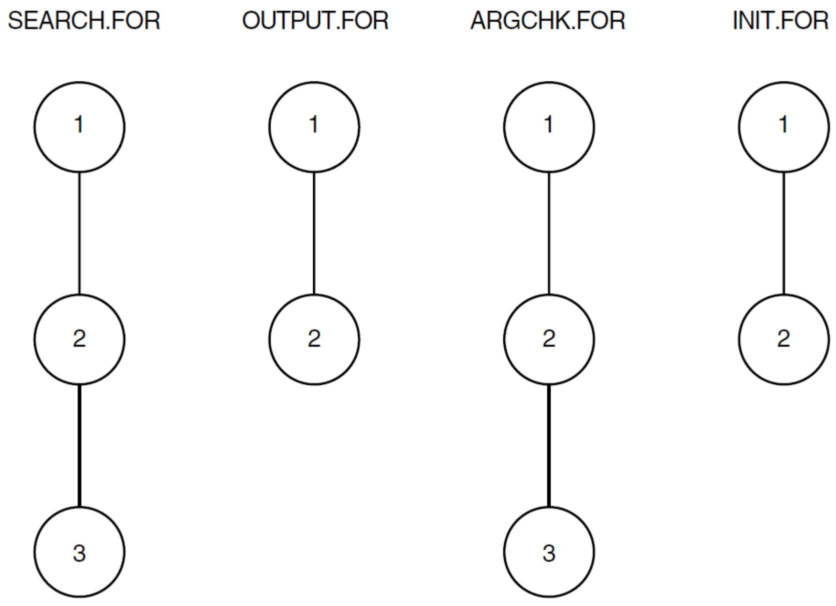
4.1. The Relationship Between Elements and Generations

When you place a file in a CMS library for the first time, CMS uses that file to create an element; that file becomes generation 1 of that element. An element generation represents a specific version of an element. Each time you reserve and replace an element in the library, CMS creates a new generation. CMS can store multiple generations of an element.

CMS assigns a permanent generation number to each new generation. This number is unique for each generation of a particular element. A CMS generation number is not the same as a version number that OpenVMS assigns to files; file version numbers have no significance to CMS.

Figure 4.1, "Elements and Their Generations" shows four elements and their generations in a simple CMS library.

Figure 4.1. Elements and Their Generations



This library contains three generations of the element SEARCH.FOR. The first generation was created with the CREATE ELEMENT command. Then, a generation of the element was reserved from and replaced into the library twice, creating generations 2 and 3. Similarly, the library contains two generations of OUTPUT.FOR, three generations of ARGCHK.FOR, and two generations of INIT.FOR. CMS stores the entire text of the first generation of an element. Then, in successive generations, CMS stores only the lines that change from one generation to the next (see *Section 4.4, "Delta Files"* and *Appendix B, "CMS Library Storage Method"* for more information).

The following example shows how to reserve a generation of an element named INIT.FOR and replace it in the CMS library, thereby creating a new generation.

```
$ CMS RESERVE INIT.FOR "change block header offset"
%CMS-S-RESERVED, generation 2 of element DISKX:[PROJECT.CMSLIB]INIT.FOR
 reserved
     .
     .
     .
$ CMS REPLACE INIT.FOR "header offset and additional free space added"
%CMS-S-GENCREATED, generation 3 of DISKX:[PROJECT.CMSLIB]element INIT.FOR
 created
```

The RESERVE command retrieves the latest main-line generation of element INIT.FOR, which is generation 2. The file is created in your current default directory and generation 2 is marked as reserved. The REPLACE command returns the contents of the file to the CMS library and assigns the next number in sequence to the new generation. Because generation 2 was reserved, the replacement transaction creates generation 3. See *Section 4.2.3, "Reserving an Element Generation"* and *Section 4.2.4, "Replacing an Element Generation"* for more information on the RESERVE and REPLACE commands, respectively.

4.2. Manipulating Elements and Generations

The following sections describe how to create, fetch, reserve, replace, monitor, display, and delete elements and generations in a CMS library.

4.2.1. Creating Elements and Generations

You create an element with the CREATE ELEMENT command. Each time you reserve and replace a generation of an element, you create a new generation of that element (see *Section 4.2.4, "Replacing an Element Generation"*).

In the CREATE ELEMENT command, you specify the name and type of the file that is to become the name of the element. Within a library, all element names must be unique. The file-name component cannot be 00CMS because that name is reserved for CMS. Specify the file with the following syntax: `filename.type`

For example:

```
$ CMS CREATE ELEMENT INIT.FOR "initialization routines"
%CMS-S-CREATED, element DISKX:[PROJECT.CMSLIB]INIT.FOR created
```

This command creates an element named INIT.FOR from the file INIT.FOR. CMS searches for the file named INIT.FOR in your default directory; use the /INPUT qualifier on the CREATE ELEMENT command to specify a different location, a different file name, or both.

When an element is created, CMS deletes all versions of the file in your default directory used to create the new element. Use the /KEEP or /RESERVE qualifier to prevent CMS from deleting any files. This can be specified at the element or library level.

```
$ CMS CREATE ELEMENT/keep/binary FILE1.C "Creating binary element"
%CMS-S-CREATED, element DISK1:[DIR.CMSDIR]FILE1.C created
```

This command creates an element named FILE1.C. /BINARY qualifier is used so that CMS creates the element as binary type and treats it as binary type for future transactions.

4.2.2. Fetching an Element Generation

The FETCH command copies the contents of an element generation into a file. Unlike the RESERVE command, FETCH does not mark an element generation as reserved, and you cannot replace a fetched copy in the library. You can fetch a copy of a generation of an element regardless if the element is reserved.

For example, to retrieve a copy of the latest main-line generation of an element named TIMTST.COM, enter the following command:

```
$ CMS FETCH TIMTST.COM
_Remark: Testing storage blocks
%CMS-S-FETCHED, generation 2 of element DISKX:[PROJECT.CMSLIB]TIMTST.COM
 fetched
```

CMS retrieves the latest main-line generation of the element TIMTST.COM (generation 2). To retrieve an earlier generation (or variant generation; see *Chapter 6, "Variants and Merging"*), specify the /GENERATION qualifier on the FETCH command.

CMS creates a file with the same name as the fetched element, and places this file in your current, default directory. Use the /OUTPUT qualifier on the FETCH command to specify a different file name, a different location, or both.

4.2.3. Reserving an Element Generation

After creating an element in a CMS library, use the RESERVE command to retrieve a copy of a generation of the element to make changes to it. When you reserve a generation, CMS retrieves a copy of the generation of the element and marks that generation as reserved in the CMS library. You can then edit, compile, test, and debug the file as necessary. Most of your work with the CMS library consists of reserving library element generations for work and replacing the modified files back into the library as new generations.

When you reserve a generation of an element, CMS creates a file with the same name as the element, and places this file in your current, default directory. Use the /OUTPUT qualifier on the RESERVE command to specify a different location, a different file name, or both.

CMS prompts you to enter a remark when you reserve an element generation. You should use the remark to explain why you are reserving the generation; the remarks provide a permanent record of your work. If you need to revise the remark at a later time, you can use the MODIFY RESERVATION command to enter a new remark string.

For example, to reserve the element SYNCHRON.BAS, enter the following command:

```
$ CMS RESERVE SYNCHRON.BAS
_Remark: losing sample from one data line
%CMS-S-RESERVED, generation 2 of element DISKX:[PROJECT.CMSLIB]SYNCHRON.BAS
 reserved
```

CMS copies the element generation into a file that is created in your current default directory. CMS marks the generation with your reservation and records the transaction in the library transaction history.

CMS retrieves the latest main-line generation of the element SYNCHRON.BAS (generation 2). To reserve an earlier generation (or a variant generation; see *Chapter 6, "Variants and Merging"*), specify the /GENERATION qualifier on the RESERVE command. For example:

```
$ CMS RESERVE SYNCHRON.BAS/GENERATION=1
```

```
_Remark: Commenting data line sampling code
%CMS-S-RESERVED, generation 1 of element DISX:[PROJECT.CMSLIB]SYNCHRON.BAS
 reserved
```

A copy of the first generation of the element is then placed in the current, default directory.

CMS allows you to concurrently reserve more than one generation of the same element, or the same generation more than once. If any generation of an element is already reserved (by you or another person) CMS issues a message about the reservation already in effect. You then have the option to proceed with your reservation or quit. If you choose to proceed with the reservation, the element is considered to have concurrent reservations. While you have an element generation reserved, any user who reserves or attempts to reserve a generation of the same element receives a CMS message indicating that you have reserved the element generation. See *Section 4.3.2, "Concurrent Reservations"* for more information on concurrent reservations.

If you reserve an element generation and then decide not to modify it, you can cancel your reservation with the UNRESERVE command. CMS records the cancellation in the library history. CMS does not modify the library element and does not create a new generation of the element when you cancel a reservation. The UNRESERVE command is useful if you reserve a wrong element, or if you do not want your modifications to become part of the element. For example, to unreserve a generation of the element named SYNCHRON.BAS, enter the following command:

```
$ CMS UNRESERVE SYNCHRON.BAS
_Remark: element not applicable-wrong file
%CMS-S-UNRESERVED, element DISKX:[PROJECT.CMSLIB]SYNCHRON.BAS unreserved
```

Normally, CMS allows you to unreserve only your own reservation. However, if you hold BYPASS privilege or if an access control entry (ACE) on the element grants you BYPASS access, you can cancel any reservation of that element held by another user. The cancellation of the reservation is then logged in the history file under your name. See *Chapter 7, "Security Features"* for more information.

4.2.4. Replacing an Element Generation

After modifying a reserved generation, use the REPLACE command to replace the latest version of the modified file into the library. CMS then deletes all copies of that file from your directory (unless you specify the /KEEP or /RESERVE qualifier), assigns a new CMS generation number to the newly created element generation, terminates your reservation, and records the transaction. For example:

```
$ CMS RESERVE SPEC.RNO
_Remark: Misspelling in Reliability section
%CMS-S-RESERVED, generation 3 of element DISKX:[PROJECT.CMSLIB]SPEC.RNO
 reserved
     .
     .
     .
$ CMS REPLACE SPEC.RNO
_Remark: Reliability section typo fixed
%CMS-S-GENCREATED, generation 4 of element DISKX:[PROJECT.CMSLIB]SPEC.RNO
 created
```

In this example, the current generation (generation 3) is reserved to correct a typographical error, and then replaced.

To avoid creating a new generation if the input file has no changes from the reserved generation, use the /IF_CHANGED qualifier on the REPLACE command. You can also use the / INSERT_INTO_CLASS qualifier to insert the generation into one or more classes automatically. For more information, see the description of the REPLACE command in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

Normally, CMS allows you to replace only your own reservation. However, if you hold BYPASS privilege or if an ACE on the element grants you BYPASS access, you can replace any reservation of that element held by another user. This mechanism allows you to designate a single person who is responsible for reviewing and entering all changed reservations into the library, for example. See *Chapter 7, "Security Features"* and *Chapter 8, "Event Handling and Notification"* for more information on ACEs.

CMS allows you to concurrently reserve more than one generation of the same element, or the same generation more than once. When you replace the generations that are concurrently reserved by you, you must specify the /GENERATION or /IDENTIFICATION_NUMBER qualifier on the REPLACE command. See *Section 4.3.3, "Concurrent Replacements"* for information on replacing concurrent reservations.

4.2.5. Monitoring Element Changes

You can monitor changes made to elements by using the notification ACE or the **review** attribute.

The notification ACE enables you to specify a list of people to be notified when particular events occur in a CMS library. See *Chapter 8, "Event Handling and Notification"* for more information on CMS notification.

The **review** attribute enables you to specify that newly created element generations are to be placed on a review pending list. You can then associate review remarks with a generation under review. To assign the **review** attribute, use the /REVIEW qualifier on either the CREATE ELEMENT or the MODIFY ELEMENT command. The **review** attribute specifies that any new generations of that element are marked as pending review. You can also mark a specific generation for review by using the MARK GENERATION command. To determine which generations have reviews pending, use the SHOW REVIEWS_PENDING command.

When you review a generation, you can accept or reject the generation, cancel the review, or enter review comments. For more information on review, see *Section 4.5.4, "The Review Attribute"* and the descriptions of the following commands in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

- ACCEPT GENERATION
- CANCEL REVIEW
- MARK GENERATION
- REJECT GENERATION
- REVIEW GENERATION
- SHOW REVIEWS_PENDING

4.2.6. Displaying Information About Elements and Generations

You can view information about elements and generations in a CMS library with the SHOW commands. The SHOW ELEMENT command displays information about some or all of the elements in the current library. For example:

```
$ CMS SHOW ELEMENT
Elements in CMS Library DISKX:[PROJECT.CMSLIB]
ADCONVERT.BAS "analog to digital conversion routines"
ERRMSG.TXT    "initial load"SAMPLE.BAS    "Sampling module"
SPEC.RNO      "ADS functional specification"
SYNCHRON.BAS  "Synchronization routines"
TIMTST.COM    "Command procedure for tests"
```

In this case, SHOW ELEMENT displays an alphabetical list of all the elements in the project library [PROJECT.CMSLIB], along with their remarks. You can also use the SHOW ELEMENT/MEMBER command to display the element name, creation remark, and the name of any groups to which the element belongs.

If you need information about a specific generation of an element, use the SHOW GENERATION command. If you omit a generation number, CMS assumes the last generation on the main line of descent. If you include a generation number, specify it with the /GENERATION qualifier. For example:

```
$ CMS SHOW GENERATION SYNCHRON.BAS/GENERATION=3
Element generations in CMS Library DISKX:[PROJECT.CMSLIB]
SYNCHRON.BAS    3    26-JUN-2005 09:44:12 KELLEY "a/d conversion
 integrated"
```

This command displays the characteristics of generation 3 of the element SYNCHRON.BAS.

To discover which generation of an element is in a particular class, use the SHOW GENERATION command and specify the class name as the generation expression. For example:

```
$ CMS SHOW GENERATION/GENERATION=BASELEVEL1 SYNCHRON.BAS
Element generations in CMS Library DISKX:[PROJECT.CMSLIB]
   SYNCHRON.BAS    3    26-JUN-2005 09:44:12 KELLEY "a/d conversion
 integrated"
```

This command displays the generation of element SYNCHRON.BAS that is in class BASELEVEL1.

The CMS command SHOW GENERATION/MEMBER lists the element name, generation number, and names of any classes to which the element generation belongs. The CMS command SHOW GENERATION/MEMBER defaults to the latest generation. To discover if the generation that is in the class is not the latest generation of the element, use the /DESCENDANTS qualifier. For example:

```
$ SHOW GENERATION/MEMBER/DESCENDANTS TEST.FOR
Element generations in CMS Library DISKX:[PROJECT.CMSLIB]
   TEST.FOR
     2     18-OCT-1994 15:04:54 SMITH "header changed"

     1     26-SEP-1994 13:38:04 SMITH ""
             Member list:        RELEASE1
```

This command displays the generation number, date, time, remark, and the name of any classes to which each generation belongs for all the generations of the element TEST.FOR.

Whenever you create, reserve, or replace a library element, CMS stores information about the transaction in the library's history file. The SHOW HISTORY command allows you to review a chronological list of all library transactions. For example:

```
$ CMS SHOW HISTORY
History of CMS Library DISKX:[PROJECT.CMSLIB]
2-MAY-2005 14:22:16 WHIPPLE CREATE LIBRARY DISKX:[PROJECT.CMSLIB] "a/d data
       sampling library"
```

```
2-MAY-2005 14:26:47 MARTIN CREATE ELEMENT SPEC.RNO "ADS functional
       specification"
8-JUN-2005 12:09:02 WHIPPLE CREATE ELEMENT ADCONVERT.BAS "analog to digital
       conversion routines"
8-JUN-2005 12:25:41 WHIPPLE CREATE ELEMENT SAMPLE.BAS "Sampling module"
8-JUN-2005 12:29:24 HENRY CREATE ELEMENT SYNCHRON.BAS "Synchronization
       routines"
8-JUN-2005 14:01:36 HENRY CREATE ELEMENT TIMTST.COM "Command procedure for
       tests"
9-JUN-2005 14:47:40 DAVIS RESERVE SYNCHRON.BAS(1) "losing sample from one
       data line"
```

CMS does not record transactions that do not alter the library. CMS logs FETCH transactions only if you supply a remark.

You can use the SHOW HISTORY command with the /UNUSUAL qualifier to report any abnormal library transactions that occurred, such as two reservations in effect for the same element at the same time.

The SHOW RESERVATIONS command lists element generations that are currently reserved (or by identification number, if the element is concurrently reserved), who reserved each generation, when it was reserved, and why it was reserved. For example:

```
$ CMS SHOW RESERVATIONS
Reservations in CMS Library DISKX:[PROJECT.CMSLIB]
SAMPLE.BAS  (1)  JIMK     1
      30-JUN-2005 11:19:29 "add code for more data lines"
SYNCHRON.BAS  (1)  KELLEY   3
      18-JUN-2005 09:42:03 "integrate a/d conversion"
```

You can use the SHOW GROUP/CONTENTS command to display the contents of a group. For example:

```
$ CMS SHOW GROUP/CONTENTS TIME_TST
Groups in CMS Library DISKX:[PROJECT.CMSLIB]TIME_TST
           "comparison testing prototype group"
    SYNCHRON.BAS
    TIMTST.COM
```

This command lists the elements contained in group TIME_TST.

4.2.7. Deleting Generations

To delete one or more generations of an element from the library, use the DELETE GENERATION command. This command is useful if you replaced a wrong version of a file into the CMS library, or if you want to remove old generations of elements. This command permanently removes information about a generation from the corresponding element in the library. If the latest main-line generation is deleted, the next latest main-line generation is placed into the reference copy directory. Deleting a generation does not remove changes from subsequent generations that were originally made in the deleted generation and thus exist in subsequent generations.

Deleting unneeded generations allows operations that access generations (for example, FETCH, REPLACE, and RESERVE) to complete faster because the number of generations to be searched is reduced (see *Section 9.3.2, "Deleting and Archiving Element Generations"* for more information).

When you delete an element generation, you can optionally use the /ARCHIVE qualifier to direct CMS to create an archive file. When you specify /ARCHIVE, CMS creates a file containing all the information from the deleted generation, and places it in your default directory. For more information, see *Section 9.3.2, "Deleting and Archiving Element Generations"* and the description of DELETE GENERATION/ARCHIVE in the online help and the VSI DECset for OpenVMS Code Management System Reference Manual.

4.3. Concurrency

This section describes how CMS organizes concurrent changes to library elements and how to resolve conflicting changes to those elements. A concurrent change occurs when two or more people work on an element at the same time and make separate changes to the element.

If you cannot avoid making a concurrent reservation, be aware that some additional effort is involved when you replace concurrent reservations. The following sections describe how to reserve a generation of an element that has prior reservations, and replace the reservation into the library.

4.3.1. Concurrent Access

CMS allows you to control concurrent access to an element by using the **concurrent** attribute. You define the **concurrent** attribute by specifying the /[NO]CONCURRENT qualifier. The library-wide default is /CONCURRENT, which can be changed at the library or element level.

You can prohibit concurrent access by specifying the /NOCONCURRENT qualifier on the CREATE ELEMENT command for a new element, or by using the MODIFY ELEMENT command to change the attribute of an existing element. You cannot modify concurrent access to an element while a generation of the element is reserved. When you prohibit concurrent access to an element, only one reservation of the element is allowed at a time until you use the MODIFY ELEMENT/CONCURRENT command to allow concurrent access.

You can temporarily prohibit concurrent access for the duration of a reservation by specifying the / NOCONCURRENT qualifier on the RESERVE command. If you reserve a generation of an element in this way, you must replace it or cancel the reservation (with the UNRESERVE command) before you or anyone else can reserve any generation of the element.

4.3.2. Concurrent Reservations

If a generation of the element you want to reserve is already reserved and concurrent access is not prohibited, CMS accepts your RESERVE command and the remark you enter with it, but warns you that an element generation is currently reserved and by whom, and prompts you for confirmation before proceeding. If you continue with the reservation, the element is then marked as being concurrently reserved, and it retains that status until all reservations of the element are ended. For example:

```
$ CMS RESERVE BASTEST.GNC
_Remark: reserving for final production
Element BASTEST.GNC currently reserved by:
      (1)    DAVIS       3     28-JAN-2005  09:27:46  "for testing"
Proceed?  [Y/N] (N):
```

If you type NO or press Return, CMS does not execute the reserve transaction. If you type YES, CMS places a copy of the generation in your current, default directory, marks the element as concurrently reserved, and records the reservation transaction in the library history. CMS records the transaction as an unusual occurrence. For information about unusual occurrences, see *Chapter 9, "Library Maintenance"*.

CMS allows multiple reservations by a single user; that is, you can reserve more than one generation of the same element, and you can also reserve the same generation more than once. CMS assigns a unique identification number to each reserved generation. The identification number appears first on each line. Use the SHOW RESERVATIONS command to determine the identification number of each reservation. You must use the /IDENTIFICATION_NUMBER qualifier to replace a concurrent reservation (see *Section 4.3.3, "Concurrent Replacements"*).

4.3.3. Concurrent Replacements

When a concurrent reservation ends (when you replace the element generation with the REPLACE command), it is called a **concurrent replacement**. When you replace an element that another user has concurrently reserved, CMS reports that a prior concurrent reservation was made and specifies who the second reserver was, even if the second reserver has already replaced the element. For example:

```
$ CMS REPLACE BASTEST.GNC
_Remark: replacing after completing edits
Concurrent replacements
      (1)    DAVIS       2     28-JAN-2005  09:27:46  "for testing"
Proceed?  [Y/N] (N):
```

If you type NO or press Return after the Proceed? prompt, CMS does not execute the replacement transaction. If you type YES, CMS proceeds with the command and records the transaction as an unusual occurrence. For information about unusual occurrences, see *Chapter 9, "Library Maintenance"*.

At least one reserver must replace a concurrent reservation as a variant generation. You replace the concurrent reservation as a variant generation by specifying the /VARIANT qualifier on the REPLACE command. This begins a variant line of descent. Either user can then merge the variant generation back into the original line so both sets of program modifications appear in one generation, or the variant line of descent can be continued (see *Chapter 6, "Variants and Merging"* for more information).

CMS allows you to concurrently reserve a specific generation more than once. When you replace the generations that are concurrently reserved by you, you must specify which reservation is to be replaced. You can do this with either the /GENERATION qualifier or the /IDENTIFICATION_NUMBER qualifier on the REPLACE command.

You can use /GENERATION as long as the concurrent reservations are not on the same generation. If you have more than one concurrent reservation for the same generation, you must identify the specific reservation to be replaced. Each reservation is assigned an identification number. Use the SHOW RESERVATIONS command to determine the identification number of each reservation. The identification number appears first on each line. If you use the /IDENTIFICATION_NUMBER qualifier, you do not need to also use the /GENERATION qualifier. For example:

```
$ CMS REPLACE BASTEST.GNC/IDENTIFICATION_NUMBER=2
_Remark: replacing after completing edits
Element BASTEST.PAS currently reserved by:
      (1)    DAVIS       3     28-JAN-2005  09:27:46  "for testing"
Proceed?  [Y/N] (N):
```

In this example, the /IDENTIFICATION_NUMBER qualifier specifies that the second reserved generation be replaced into the CMS library. CMS reports other existing reservations you hold for that element (in this case, the first reserved generation), and then prompts you to proceed.

You must also use the /GENERATION or /IDENTIFICATION_NUMBER qualifier if you are replacing another user's reservation. For more information, see *Section 7.3, "OpenVMS BYPASS Privilege and CMS BYPASS Access"* and the description of the REPLACE command in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

4.4. Delta Files

For each element stored in the library, CMS maintains a **delta file**—a single file containing a representation of the contents of all the generations of that element.

In addition to the actual data, the delta file contains **control records**. Control records tell CMS which data records are valid for which specific generations of the element. When you retrieve a generation of an element, CMS includes records that are valid and excludes records that are not valid for that generation.

One of the effects of the delta file method of storing information is that retrieval times are consistent within a given element. For example, it takes a similar amount of time to fetch generation 100 of an element or generation 1 of that same element. Another effect is that generation deletion does not necessarily produce a significantly smaller delta file, because records that are valid in a generation being deleted might also be valid (and, in fact, are likely to be valid) in later or earlier generations that are not being deleted.

See *Appendix B, "CMS Library Storage Method"* for more information on how CMS stores library information.

4.5. Element Attributes

The CREATE ELEMENT and MODIFY ELEMENT commands enable you to specify the following element attributes:

- The **concurrent** attribute controls whether concurrent reservations of an element are allowed (See *Section 4.3.2, "Concurrent Reservations"* for more information).
- The **history**, **notes**, and **position** attributes enable you to manipulate the format of historical information associated with an element.
- The **reference copy** attribute directs CMS to maintain a reference copy of an element.
- The **review** attribute directs CMS to mark newly created generations of an element as pending review.

You can use the SHOW ELEMENT/FULL command to display the current settings of these attributes. For more information, see the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

4.5.1. The History Attribute

When the **history** attribute is defined for an element, CMS includes the element generation history in the output file when you retrieve a generation of an element from the library with the FETCH or RESERVE command. This history is a list of the transactions that created each generation of the element. Each transaction record consists of the generation number, user, date, time, and remark associated with the generation.

Use the /HISTORY qualifier to define the **history** attribute for an element. You can either establish the **history** attribute when the element is created with the CREATE ELEMENT command, or change the **history** attribute of an existing element with the MODIFY ELEMENT command. You can cancel the **history** attribute by using the /NOHISTORY qualifier on the MODIFY ELEMENT command. You can also specify the /[NO]HISTORY qualifier on the FETCH and RESERVE commands to temporarily override the element's **history** attribute.

The format of the /HISTORY qualifier is as follows: `/HISTORY="string"/NOHISTORY`

"string"

Specifies the format of each line of the element history. The string must contain exactly one occurrence of the history format parameter, can contain only printing ASCII characters and the space and tab characters, and must begin and end with a quotation mark ("). The history format parameter consists of a number sign (#) followed by an uppercase or lowercase letter H or B. For example:

```
"!#H""
/*#b*/"
```

The exclamation point (!) and the slash-asterisk characters (/* */) indicate comments.

Use the letter B to direct CMS to include the history at the beginning of the file and H to include the history at the end of the file. The history text is inserted into the string wherever the #H or #B history format parameter occurs. To include a number sign (#) in the string, type it twice (##). To include a quotation mark in the string, type it twice (""). If the file contains source code, you must include comment indicators or delimiters applicable to your source code in the string so the program can be compiled or assembled. The history is then treated as a comment.

Note

Because of Record Management Services (RMS) record storage restrictions, CMS cannot include history text in files with fixed-length records. If you try to fetch or reserve a generation of an element that has history enabled and the generation has fixed-length records, you receive the following message:

```
%CMS-I-NOHISTNOTES, history and notes will not be included in output file
```

The history includes a line for each generation of an element. Each line consists of the text contained in the quoted string, with #H or #B replaced by the creation information for that generation. The history region is delimited by the following line:

```
CMS REPLACEMENT HISTORY, Element element-name
```

This line enables the REPLACE command to distinguish the history from the rest of the file when it is returned to the library. CMS does not consider history text to be part of the file. Instead, the history is added to the file when it is retrieved from the library and removed when the file is replaced into the library. The generation numbers of a retrieved generation and its ancestors are marked with an asterisk (*).

Do not insert or modify text in the history section while editing a file in your directory. CMS expects only history lines between the two header lines. The REPLACE command reports an error if it finds any other text where the history should be, and the command is not executed. You must then delete the extra text with an editor and reenter the REPLACE command.

The following command establishes the **history** attribute for a file that contains a Pascal program:

```
$ CMS MODIFY ELEMENT SEMANTICS.PAS/HISTORY="{#H}" "est. history attribute"
```

When the default generation of SEMANTICS.PAS is retrieved with the FETCH or RESERVE command, the history at the end of the program looks like this:

```
{ CMS REPLACEMENT HISTORY, Element SEMANTICS.PAS }
{ *6    28-JUL-2005 10:00:54 EDGAR   "formal parameter list support added"}
{ *5    24-JUL-2005 16:10:14 DAVIS   "actual parameter list support added"}
{ *4    20-JUL-2005 12:22:07 MARTIN  "preliminary work on routine calls
 done"}
{ 3C1   17-JUL-2005 12:15:45 JEFF    "error checking on CASE statement
 works"}
{ *3    11-JUL-2005 11:57:18 MALER   "CASE statement support"}
{ *2     7-JUL-2005 11:56:05 HENRY   "FOR loop support done"}
{ *1     9-JUN-2005 18:11:25 BARRETT "semantic analysis module"}
{ CMS REPLACEMENT HISTORY, Element SEMANTICS.PAS }
```

The braces ({}) indicate comments in Pascal. Because the braces surround the lines of the history, the history lines are ignored by the Pascal compiler. The history is delimited with the header line. Each existing generation of the element is listed. The generation numbers of the specified generation and its ancestors are marked with an asterisk. Generation 6 was retrieved; therefore, that generation and its ancestors are marked with an asterisk. Generation 3C1 is not an ancestor because it is on a variant line of descent.

Note

Some language processors do not accept a file that has data after the formal end of the program. If you use #H in the definition of the **history** attribute for an element, the element file might not be compatible with these processors. If this occurs, you can specify the /NOHISTORY qualifier with the RESERVE and FETCH commands. When you use this qualifier, CMS does not include the history in the file placed in your directory. Also, because CMS wraps history lines at 132characters, you can use the / NOHISTORY qualifier with history lines that are longer than 132 if your file is to be used by a processor or compiler that does not accept 132-character lines.

See *Section 4.5.5, "Examples of Using Element Attributes"* for an example of using the **history** attribute.

4.5.2. The Notes and Position Attributes

When you use the RESERVE or FETCH command and the **notes** and **position** attributes are defined for an element, CMS appends notes to lines of the file. A note appears on every line that has been modified since generation 1 as close to the position specified by the **position** attribute as possible. Notes can be one or both of the following:

- Generation numbers indicating the latest generation in which the line was inserted or modified
- ASCII text contained in the quoted string parameter of the /NOTES qualifier

You use the /NOTES qualifier to define the **notes** attribute, and the /POSITION qualifier to define the column in which the note should start. You can establish these attributes when the element is created with the CREATE ELEMENT command, or you can change the attributes of an existing element with the MODIFY ELEMENT command. Any element that has the **notes** attribute must have the **position** attribute and vice versa. Use the /NONOTES qualifier with the MODIFY ELEMENT command to cancel both attributes.

You can also specify the /[NO]NOTES and /POSITION qualifiers with the FETCH and RESERVE commands to temporarily override the element's **notes** and **position** attributes.

The format of the /NOTES qualifier is as follows:

```
/NOTES="string"
```

```
/NONOTES
```

"string"

Specifies the format of the notes. The string can contain only ASCII characters; it must begin and end with a quotation mark ("). The notes string cannot exceed 100 characters. The string can optionally contain one occurrence of the notes format parameter. The notes format parameter consists of a number sign (#) followed by an uppercase or lowercase letter G. For example:

```
"!#G"
"/*#g*/"
```

The exclamation point (!) and the slash-asterisk characters (/* */) indicate comments.

To include a number sign (#) in the string, type it twice (##). To include a quotation mark in the string, type it twice (""). If the file contains source code, you must include comment indicators applicable to your source code in the string so the program can be compiled or assembled. The notes are then treated as comments. See *Section 4.5.5, "Examples of Using Element Attributes"* for an example.

A note for a line consists of the text contained in the quoted string. The notes parameter is replaced by the number of the generation in which the line was inserted or most recently modified.

Note

Because of Record Management Services (RMS) record storage restrictions, CMS cannot include notes text in files with fixed-length records. If you attempt to fetch or reserve a generation of an element that has notes enabled and the generation has fixed-length records, you receive the following message:

```
%CMS-I-NOHISTNOTES, history and notes will not be included in output file
```

A note for a line appears at the position specified by the /POSITION qualifier. The /POSITION qualifier is required when /NOTES is specified.

The format of the /POSITION qualifier is as follows:

```
/POSITION=n
```

n

Specifies the character position at which the notes are to appear on the line. The position value must be an integer in the range 1 to 511.

The note is placed to the right of the text of the line. If the length of the line is less than *n*, the note begins at position *n*. If the length of the line is greater than or equal to *n*, the note begins at the next tab stop after the end of the text of the line. (Tab stops are at position 9 and at every 8 characters thereafter.)

CMS does not consider notes to be part of the element generation. Instead, notes are added to the file when it is retrieved from the library and removed when the file is replaced into the library. If, while editing the file, you add text after the note or within the note, CMS does not recognize it as a note; and therefore, replaces it as part of the generation. If you add text that looks like a note, CMS interprets it as a note and removes it before replacing the file.

See *Section 4.5.5, "Examples of Using Element Attributes"* for an example of using the notes and position attributes.

4.5.3. The Reference Copy Attribute

An element reference copy is a copy of the latest main-line generation of an element. CMS maintains reference copies of the latest generations of selected library elements in a non-library directory.

If you have established a reference copy directory for a library, each newly created element is automatically set with the /REFERENCE_COPY qualifier. New elements inherit the **reference copy** attribute from the library setting.

When the **reference copy** attribute is enabled for an element, CMS creates a reference copy by fetching a copy of the latest main-line generation into the reference copy directory. If, for any reason, the reference copy directory cannot be updated, CMS does not create the new generation.

You can use the /REFERENCE_COPY qualifier to define the **reference copy** attribute for a single element. Either establish the **reference copy** attribute when the element is created with the CREATE ELEMENT command, or change the **reference copy** attribute of an existing element with the MODIFY ELEMENT command. You can prevent CMS from creating a reference copy by specifying the /NOREFERENCE_COPY qualifier with the CREATE ELEMENT or MODIFY ELEMENT command.

The format of the /[NO]REFERENCE_COPY qualifier is as follows:

```
/REFERENCE_COPY
/NOREFERENCE_COPY
```

For more information on reference copies, See *Section 3.1.4, "Creating a Reference Copy Directory"* and the /[NO]REFERENCE_COPY qualifier in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

4.5.4. The Review Attribute

When the **review** attribute is enabled for an element, CMS places any newly created generations of that element on the review pending list, and marks them for review. You can associate review remarks with a generation under review by using the REVIEW GENERATION command (see the online help or the VSI DECset for OpenVMS Code Management System Reference Manual).

You use the /REVIEW qualifier to define the **review** attribute for an element. You can establish the **review** attribute when the element is created with the CREATE ELEMENT command, or you can change the attribute of an existing element with the MODIFY ELEMENT command. You can cancel the **review** attribute by using the /NOREVIEW qualifier on the MODIFY ELEMENT command.

The format of the /[NO]REVIEW qualifier is as follows:

```
/REVIEW
/NOREVIEW
```

To determine what generations are under review, use the SHOW REVIEWS_PENDING command, which also shows any review comments. Once a generation is under review, a user trying to retrieve that generation with a FETCH command is informed that a review is pending. If you retrieve the generation with the RESERVE command, you are informed that a review is pending and are prompted for confirmation to continue. The messages are issued until the generation's review status is resolved. A generation with a review pending cannot be deleted.

You can resolve a generation's review status in one of three ways: accept the generation with the ACCEPT GENERATION command, cancel the review with the CANCEL REVIEW command, or reject the generation with the REJECT GENERATION command. If you accept the generation or cancel the review, CMS halts review-related messages and confirmations on subsequent reservation attempts. If you reject the generation, CMS issues a message indicating that the generation was reviewed and rejected. The generation is still accessible so the problems in it that caused the rejection can be corrected.

A generation created from a generation that currently has a review pending or that was previously rejected is automatically marked for review, regardless of the setting of the element's **review** attribute.

You can also use the MARK GENERATION and REVIEW GENERATION commands to mark a generation for review and to review the generation. For more information, see the descriptions of these commands in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

See *Section 4.5.5, "Examples of Using Element Attributes"* for an example of using the **review** attribute.

4.5.5. Examples of Using Element Attributes

The following example shows how to create the element DOC.C with the **history**, **notes**, and **position** attributes.

```
$ CMS CREATE ELEMENT DOC.C/HISTORY="!#H"/NOTES="!#G"/POSITION=80
_Remark: Require file for multiple reservations
%CMS-S-CREATED, element DISKX:[PROJECT.CMSLIB]DOC.C created
```

This command creates an element called DOC.C. The element contains data structures written in the C programming language. The /HISTORY qualifier specifies that history is to be appended to the file when it is retrieved from the library. Each line of the history is preceded by a slash and asterisk (/*), which indicates a comment in the C language. The /NOTES and /POSITION qualifiers specify that generation numbers are to be embedded in the lines of the file at position 80. The generation numbers are preceded by an exclamation point (!).

The history and notes are embedded in the file DOC.C when it is retrieved with the RESERVE or FETCH command. Alternatively, you can specify /NONOTES or /NOHISTORY with the FETCH or RESERVE command to direct CMS to omit the notes or history in the file.

In the following example, three generations of the file DOC.C exist and the element is retrieved with the FETCH command.

```
$ CMS FETCH DOC.C
_Remark: take a look at history and notes specifications
%CMS-S-FETCHED, generation 3 of element DISKX:[PROJECT.CMSLIB]DOC.C fetched
```

The FETCH command retrieves generation 3 of the element. (If the /NONOTES and /NOHISTORY qualifiers had been specified on the FETCH command, the retrieved file would not contain the notes and history embedded by the element creation.) The file that is delivered to the user's directory is shown in *Example 4.1, "An Element with History and Notes Attributes"*.

Example 4.1. An Element with History and Notes Attributes

```
/* Tests of CMS$CREATE_LIBRARY return status values */ ❶
#include <stdio.h>                 /* Input/output utilities */ ❶
#include "[]easy.h"  /* Descriptor utilities   */ ❶

globalvalue cms$_created;
globalvalue cms$_nocreate;
```

```
main()
{

   /* Dynamic descriptor allocation */ ❶

       DYNDESC(d_libspec);
       DYNDESC(d_cmsdir);                                          !2 ❷

   /* Local variables */ ❶
     int lib_db[50];
     int status;
     char *libspec = "testlib:";
     char *cmsdir = "[.cmsdir]";                                   !2 ❷

     DYNDESC_A(d_libspec, libspec);
     DYNDESC_A(d_cmsdir, cmsdir);

/* The actual tests */
     status = cms$create_library(&lib_db,&d_cmsdir);               !2 ❷
     if (status == cms$_nocreate)
         printf("\nReturn status = CMS$_NOCREATE\n");
     else
         printf("\n*** Unrecognized return status ***\n");
     printf("\n\nCorrect operation\n");
     status = cms$create_library(&lib_db,&d_libspec);
     if (status == cms$_created)
         printf("\nReturn status = CMS$_CREATED\n");
     else
         printf("\n*** Return status is unsuccessful ***\n");      !3 ❷
}
! CMS REPLACEMENT HISTORY, Element DOC.C  ❷
! *3    23-AUG-2005 12:19:50 DOC$SMITH "" ❷
! *2    10-AUG-2005 12:17:44 DOC$JONES "" ❷
! *1     4-AUG-2005 12:14:16 DOC$SMITH "" ❷
! CMS REPLACEMENT HISTORY, Element DOC.C  ❷
```

Key to example:

❶ Indicates comments existing in the file

❷ Indicates comments supplied by CMS

The numbers at position 80 (preceded by an exclamation point) to the right of the code denote the generation in which the lines were inserted or most recently modified. Lines not changed since generation 1 have no notes. The history starts and ends with the following title:

```
! CMS REPLACEMENT HISTORY, Element DOC.C.
```

The history shows the transactions that created each generation of the element.

Example 4.2, "Example of Using the Review Attribute" shows the process of marking generations for review, displaying the list of generations in the library that are on the review pending list, rejecting a generation, and reserving a generation that has been rejected.

Example 4.2. Example of Using the Review Attribute

```
CMS> MARK GENERATION BASCHAP*.SDML
_Remark: need to review chapters for BASIC manual
```

```
%CMS-I-MARKED, generation 1 of element DISKX:[PROJECT.CMSLIB]BASCHAP1.SDML
 marked for review
%CMS-I-MARKED, generation 1 of element DISKX:[PROJECT.CMSLIB]BASCHAP2.SDML
 marked for review
%CMS-I-MARKED, generation 1 of element DISKX:[PROJECT.CMSLIB]BASCHAP3.SDML
 marked for review
%CMS-I-MODIFICATIONS, 3 modifications completed
CMS> SHOW REVIEWS_PENDING
Reviews pending in CMS Library DISKX:[PROJECT.CMSLIB]
BASCHAP1.SDML    DAVIS      1      28-JAN-2005 15:48:25 "creating Chapter 1
 INTRO"
BASCHAP2.SDML    DAVIS      1      28-JAN-2005 15:48:29 "creating Chapter 2
 SYNTAX"
BASCHAP3.SDML    DAVIS      1      28-JAN-2005 15:48:32 "creating Chapter 3
 NEW FEATURES"
CMS> FETCH BASCHAP3.SDML
_Remark: new features still applicable?
Generation 1 of element BASCHAP3.SDML has a review pending
%CMS-S-FETCHED, generation 1 of element DISKX:[PROJECT.CMSLIB]
BASCHAP3.SDML fetched
     .
     .
     .
CMS> REJECT GENERATION BASCHAP3.SDML "new features made into separate
 section, not an entire chapter"
%CMS-S-REJECTED, generation 1 of element DISKX:
[PROJECT.CMSLIB]BASCHAP3.SDML rejected
CMS> RESERVE BASCHAP3.SDML "need to pull section"
Generation 1 of element BASCHAP3.SDML has been rejected
Proceed?  [Y/N] (N): YES
%CMS-S-RESERVED, generation 1 of element DISKX:
[PROJECT.CMSLIB]BASCHAP3.SDML reserved
```

Chapter 5. Groups and Classes

This chapter describes how to create and use groups and classes.

5.1. Overview

Groups and classes are mechanisms that you can use to organize a CMS library. Both groups and classes are typically used in a library; although each mechanism creates a different library structure, both can be used in the same library without conflict.

5.1.1. Groups

A group is a collection of elements or other groups, or a combination of both. You combine one or more elements into a group that you can then manipulate as a single unit. For example, you might create a group that contains all the files that process error messages, a group that contains all the chapters and appendixes in a book, or a group that contains the modules needed to build a part of a database.

Even if an element is in a group, you can still manipulate the element as an object that is separate from the group. A group can also belong to one or more other groups. The only restriction is that a group cannot be a member of itself; that is, it cannot directly or indirectly be a subgroup of itself.

5.1.2. Classes

A class is a set of specific generations of elements that can be manipulated as a unit. A class can hold only one generation of any element.

You use classes to represent the state of development of a system or set of elements at a particular time or stage. You can think of a class as a picture taken of a library at a particular time. For example, you might create a class named FIRST_DRAFT that contains only those generations of elements that were used in producing the first draft of a manual.

Typically, you create a class to contain generations of all the components of a software system for a release version of a product. You can establish classes for different stages or milestones. For example, you could establish one class for implementation, a second for testing, and a third for generations that have completed the first two stages. As each module progresses through each stage, you assign each generation to an appropriate class; thus, you can easily determine your progress by displaying the contents of the different classes, and you can later reconstruct any stage of development.

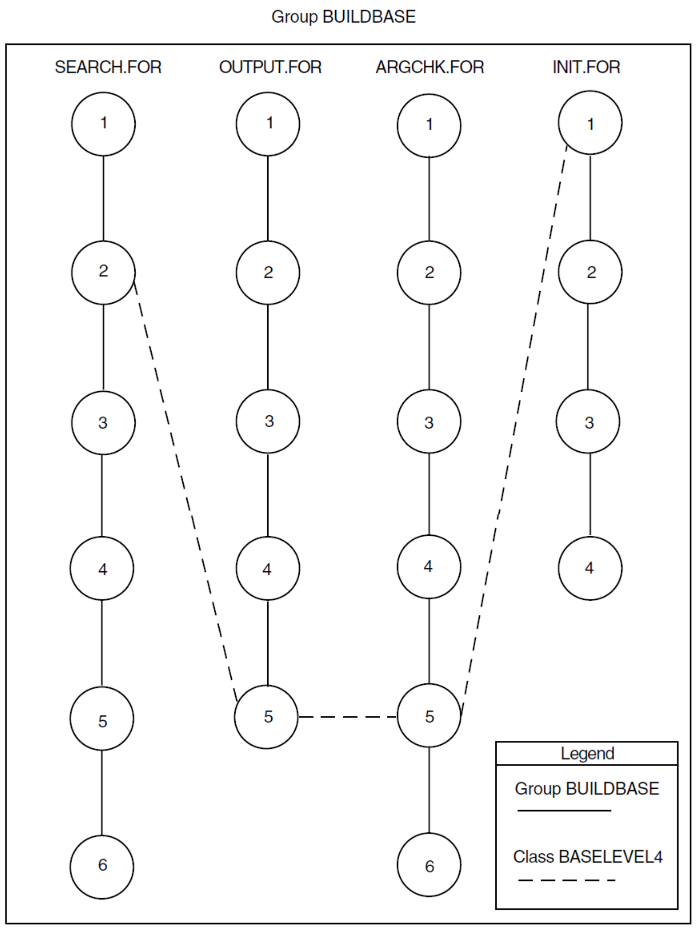
Once you insert an element generation into a class, further changes made to the element are not reflected in the contents of that class.

5.1.3. The Difference Between Groups and Classes

When you use groups, you manipulate elements. A group is an entity that enables you to give a name to a set of elements in the library and manipulate the set of elements with that name. You typically use groups to associate elements together. For example, you could create a group containing all the art figures in a manual, or a group containing all source modules that contain callable entry points.

When you use classes, you manipulate specific generations of elements. A class is an entity that enables you to give a name to a set of specific generations of elements in the library and manipulate the set with that name. In contrast to groups, classes contain only one generation from an element. You typically use classes to take a “timed snapshot” of a set of generations; that is, the generations that are meaningful to a project at a particular time. For example, you could create a class containing the specific generations that are included in a code freeze or field-test kit, or a class containing the specific generations that make up the state of the project on some other significant date. *Figure 5.1, "Groups and Classes"* shows the relationship between a group and a class.

Figure 5.1. Groups and Classes



The circles in the figure represent four elements and their generations. The number in each circle represents the generation number of the element generation. These four elements and their respective generations are contained in group BUILDBASE, a group containing the modules needed to build part of a database.

If you retrieve group BUILDBASE, you receive the latest generation on the main line of descent of each of the following elements in the group:

SEARCH.FOR, generation 6
OUTPUT.FOR, generation 5
ARGCHK.FOR, generation 6
INIT.FOR, generation 4

The dashed line that connects element generations represents class BASELEVEL4. Class BASELEVEL4 contains the element generations that comprise the state of the library on March 12, the date the project moved to base level 4.

If you retrieve class BASELEVEL4, you receive the following element generations:

SEARCH.FOR, generation 2
OUTPUT.FOR, generation 5

ARGCHK.FOR, generation 5
INIT.FOR, generation 1

5.2. Manipulating Groups

The following sections describe how to create and use groups.

5.2.1. Creating Groups

Groups can contain elements, other groups, or a combination of both. You establish an empty group with the CREATE GROUP command. For example:

```
$ CMS CREATE GROUP USER_MANUAL "user documentation"
%CMS-S-CREATED, group DISKX:[PROJECT.CMSLIB]USER_MANUAL created
```

This command creates an empty group named USER_MANUAL.

5.2.2. Inserting Elements into Groups

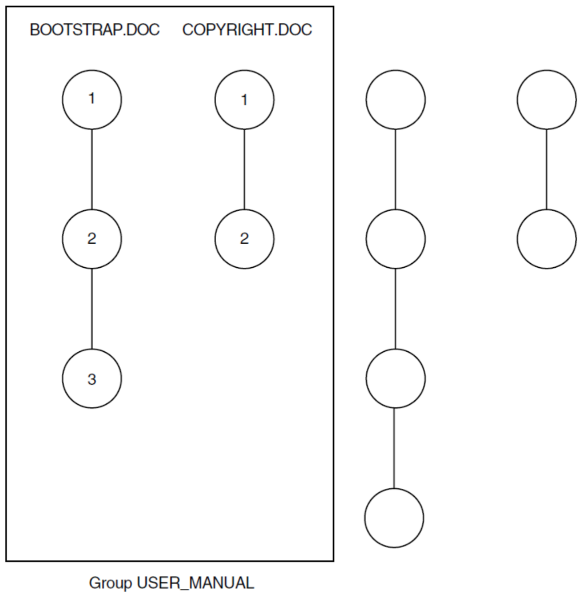
After you establish a group, you place one or more elements in the group with the INSERT ELEMENT command.

The following command inserts the elements COPYRIGHT.DOC and BOOTSTRAP.DOC into the group named USER_MANUAL:

```
$ CMS INSERT ELEMENT COPYRIGHT.DOC,BOOTSTRAP.DOC USER_MANUAL
_Remark: copyright page
%CMS-I-INSERTED, element DISKX:[PROJECT.CMSLIB]COPYRIGHT.DOC inserted into
 group DISKX:[PROJECT.CMSLIB]USER_MANUAL
%CMS-I-INSERTED, element DISKX:[PROJECT.CMSLIB]BOOTSTRAP.DOC inserted into
 group DISKX:[PROJECT.CMSLIB]USER_MANUAL
%CMS-S-INSERTIONS, 2 insertions completed
```

Figure 5.2, "Generations in a Group" shows the group USER_MANUAL, which contains two elements, BOOTSTRAP.DOC and COPYRIGHT.DOC.

Figure 5.2. Generations in a Group



This figure shows that all generations of the two elements are associated with the group. Therefore, you can access any generation of the elements in a group.

The element expression specified on the INSERT ELEMENT command can be one or more element names, group names, or a wildcard expression (for information about element expressions, see *Section 10.2.5, "Element Expressions"*). If you specify a group name with the INSERT ELEMENT command, CMS enters the names of all of the elements in that group into the destination group. For instance, if you use INSERT ELEMENT to insert the contents of group A into group B, the contents of group B are not affected by any subsequent changes of the contents of group A.

You can also use the INSERT GROUP command to insert groups (and, thus indirectly, elements) into a group. For example:

```
$ CMS INSERT GROUP USER_MANUAL CODE_AND_DOCS
%CMS-S-INSERTED, group DISKX:[PROJECT.CMSLIB]USER_MANUAL inserted into
DISKX:[PROJECT.CMSLIB]group CODE_AND_DOCS
```

This command inserts the group USER_MANUAL into the group CODE_AND_DOCS. The INSERT GROUP command enters the group name USER_MANUAL into the list of entries for the group CODE_AND_DOCS. If the contents for the group USER_MANUAL change, the elements accessible through CODE_AND_DOCS also change.

5.2.3. Retrieving and Removing Elements from a Group

After you create a group and insert elements or other groups into that group, you can retrieve all generations of elements in the group with a single FETCH or RESERVE command. For example:

```
$ CMS FETCH USER_MANUAL "copy for internal sites"
%CMS-I-FETCHED, generation 4 of element DISKX:[PROJECT.CMSLIB]
BOOTSTRAP.DOC fetched%CMS-I-FETCHED, generation 1 of element DISKX:
[PROJECT.CMSLIB]
COPYRIGHT.DOC fetched%CMS-S-FETCHES, 2 elements fetched
```

When you enter the FETCH command, CMS places a copy of the latest generation on the main line of descent of each element belonging to the group named USER_MANUAL into your current, default directory.

By default, when you retrieve a group of elements, you get the latest generation on the main line of descent of each element in the group. By using the /GENERATION qualifier, you can gain access to a specific generation. Note that when you use the /GENERATION qualifier with groups, the generation expression is applied across the group. Thus, if you were to fetch a group of elements and you specified / GENERATION=2, CMS would retrieve the second generation of each element in the group.

The REMOVE ELEMENT command enables you to remove an element from a group; however, it does not alter or delete the element itself. For example:

```
$ CMS REMOVE ELEMENT SPEC.RNO DOCUMENTATION
_Remark: User's manual ready for first review
%CMS-S-REMOVED, element DISKX:[PROJECT.CMSLIB]
SPEC.RNO removed from group DISKX:[PROJECT.CMSLIB]
DOCUMENTATION
```

This command removes the element SPEC.RNO from the group DOCUMENTATION.

You can also use the REMOVE GROUP command to remove groups from other groups. For example:

```
$ CMS REMOVE GROUP USER_MANUAL CODE_AND_DOCS "removing group"
```

```
%CMS-S-REMOVED, group DISKX:[PROJECT.CMSLIB]
USER_MANUAL removed from group DISKX:[PROJECT.CMSLIB]
CODE_AND_DOCS
```

This command removes the group USER_MANUAL from the group CODE_AND_DOCS. However, CMS does not delete or alter the groups being removed.

5.2.4. Displaying the Group Structure of a Library

To find out what groups are defined in your library, use the SHOW GROUP command. CMS lists group names in alphabetical order with the remark associated with the group. To obtain a list of all elements and groups in a specific group, use the SHOW GROUP command with the /CONTENTS qualifier. For example, to display the contents of the group named DATA_ROUTINES, you would type the following command:

```
$ CMS SHOW GROUP/CONTENTS DATA_ROUTINES

Groups in CMS Library DISKX:[PROJECT.CMSLIB]
DATA_ROUTINES "routines for input & conversion"
    ADCONVERT.BAS
    SAMPLE.BAS
```

5.2.5. Deleting Groups

The DELETE GROUP command deletes one or more groups from a CMS library. The group must be empty prior to deletion. If the group contains any content, use the REMOVE ELEMENT command, REMOVE GROUP command, or the DELETE GROUP command with the /REMOVE_CONTENTS qualifier. For example:

```
CMS> DELETE GROUP TIME_TST "superseded by comparison tests"
%CMS-S-DELETED, group DISKX:[PROJECT.CMSLIB]TIME_TST deleted
```

This command deletes the group named TIME_TST.

If the group is not empty, or if it belongs to another group, CMS returns an error and does not delete the group. Use the REMOVE ELEMENT or REMOVE GROUP command to remove elements or groups from the group before entering the DELETE GROUP command.

5.3. Manipulating Classes

The following sections describe how to create and use classes.

5.3.1. Creating Classes

You establish an empty class with the CREATE CLASS command. For example:

```
$ CMS CREATE CLASS INTERNAL_RELEASE "for use in-house only"
%CMS-S-CREATED, class DISK:[PROJECT.CMSLIB]
INTERNAL_RELEASE created
```

This command creates a class called INTERNAL_RELEASE. The class does not yet contain any element generations.

5.3.2. Inserting Element Generations into Classes

You place an element generation into a class with the INSERT GENERATION command.

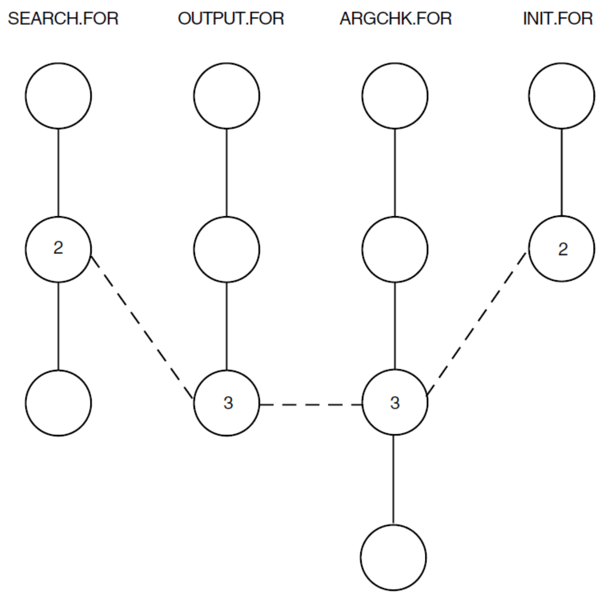
The following commands place generations of INIT.FOR and ARGCHK.FOR into the class INTERNAL_RELEASE:

```
CMS> INSERT GENERATION INIT.FOR INTERNAL_RELEASE
_Remark: Initialization routine for demo
%CMS-S-GENINSERTED, generation 2 of element DISKX:[PROJECT.CMSLIB]
INIT.FOR inserted in class DISKX:[PROJECT.CMSLIB]INTERNAL_RELEASE
CMS> INSERT GENERATION ARGCHK.FOR/GENERATION=3 INTERNAL_RELEASE
_Remark: Demo semantic analyzer
%CMS-S-GENINSERTED, generation 3 of element DISKX:[PROJECT.CMSLIB]
ARGCHK.FOR inserted in class DISKX:[PROJECT.CMSLIB]INTERNAL_RELEASE
```

The INSERT GENERATION command uses the latest generation on the main line of descent, unless you specify the /GENERATION qualifier. A class can contain no more than one generation of an element. A generation can belong to zero, one, or more classes.

Figure 5.3, "The Relationship Between Groups and Elements" shows the relationship of elements, generations, and the class INTERNAL_RELEASE.

Figure 5.3. The Relationship Between Groups and Elements



The class INTERNAL_RELEASE contains generation 2 of INIT.FOR, generation 2 of SEARCH.FOR, generation 3 of OUTPUT.FOR, and generation 3 of ARGCHK.FOR.

You can also insert a generation into one or more classes using the /INSERT_INTO_CLASS qualifier to the REPLACE command. See the VSI DECset for OpenVMS Code Management System Reference Manual for information on using this qualifier.

5.3.3. Retrieving and Removing Generations from a Class

You can retrieve an element generation from a class by specifying the class name on the /GENERATION qualifier on the FETCH or RESERVE command. A class can contain no more than one generation of an element; the class name specifies the generation of the element to be retrieved. For example:

```
$ CMS RESERVE SEARCH.FOR/GENERATION=INTERNAL_RELEASE
```

```
_Remark: add support for alternate two-character graphics
%CMS-S-RESERVED, generation 2 of element DISKX:[PROJECT.CMSLIB]SEARCH.FOR
 reserved
```

This command reserves generation 2 of SEARCH.FOR, because that generation belongs to the class INTERNAL_RELEASE.

If a class is established to contain each version or base level of a system, you can accurately reconstruct any previous version of the system. For example, if the users of your system use version 1, the element generations that constitute version 1 could belong to the class VER1. If the software has changed because you are in the process of developing version 2 and a bug is reported in version 1, you can retrieve the generation of the element in which the bug appeared because you know that it belongs to class VER1.

The REMOVE GENERATION command enables you to remove an element generation from a class. For example:

```
$ CMS REMOVE GENERATION DISPLAY.BAS BASELEVEL1 "no longer needed"
```

In this example, a generation of the element DISPLAY.BAS is removed from class BASELEVEL1. CMS then revises information about BASELEVEL1 so no generation of DISPLAY.BAS is included in the class.

5.3.4. Displaying the Class Structure of a Library

To find out which classes are defined in your library, use the SHOW CLASS command. CMS lists class names in alphabetical order with the remark that is associated with the class. To obtain a list of all generations in a specific class, use the SHOW CLASS command with the /CONTENTS qualifier. For example:

```
$ CMS SHOW CLASS/CONTENTS BASELEVEL1
Classes in CMS Library DISKX:[PROJECT.CMSLIB]
BASELEVEL1 "Specifying all generations for first base level"
    ADCONVERT.BAS       5
    DISPLAY.BAS         2
    SAMPLE.BAS          6
    SYNCHRON.BAS        4
```

This command displays all the elements and their generations in class BASELEVEL1.

5.3.5. Deleting Classes

The DELETE CLASS command deletes one or more classes from a CMS library. The class must be empty prior to deletion. If the class contains any content, use the REMOVE GENERATION command or the DELETE GROUP command with the /REMOVE_CONTENTS qualifier. For example:

```
$ CMS DELETE CLASS PRE_RELEASE "no longer necessary"
%CMS-S-DELETED, class DISKX:[PROJECT.CMSLIB]PRE_RELEASE deleted
```

This command deletes the class named PRE_RELEASE.

If any element generations belong to the class, CMS issues an error message and does not delete the class. Use the REMOVE GENERATION command to remove element generations from a class before entering the DELETE CLASS command.

5.4. Group and Class Attributes

You can change the name, remark, and **read-only** attribute of both groups and classes by using the MODIFY GROUP and MODIFY CLASS commands.

You can use the /NAME qualifier on the MODIFY GROUP command to change the name of a group created with the CREATE GROUP command. Similarly, you can use the /NAME qualifier on the MODIFY CLASS command to change the name of a class created with the CREATE CLASS command.

You can use the /REMARK qualifier on the MODIFY GROUP and MODIFY CLASS commands to specify a new remark to be substituted for the remark created with the CREATE GROUP and CREATE CLASS commands.

You can use the /READ_ONLY qualifier on the MODIFY GROUP and MODIFY CLASS commands to assign read-only access to groups or classes. A group or a class that is set to read-only access cannot be changed; you cannot insert or remove any items to or from the group or class. In addition, you cannot change the name of a group or a class that is set to read-only access.

The following example sets the group DIAGNOSTICS to read-only access:

```
$ CMS MODIFY GROUP DIAGNOSTICS/READ_ONLY
_Remark: diagnostics for use with V2 compiler
```

After this command has been executed, the group cannot be altered. To change the group, use the /NOREAD_ONLY qualifier with the MODIFY GROUP command. Similarly, you can use the /READ_ONLY and /NOREAD_ONLY qualifiers with the MODIFY CLASS command to enable or disable modifications to a class.

In addition, you can use the SET ACL and SHOW ACL commands to specify and display access control lists for groups and classes (as well as for other CMS library objects). See Chapters *Chapter 7, "Security Features"* and *Chapter 8, "Event Handling and Notification"* for more information.

Chapter 6. Variants and Merging

This chapter provides information on lines of descent, creating variant lines of descent, and merging element generations.

6.1. Lines of Descent

The line of descent for a specified generation consists of all ancestors and direct descendants of that generation. The main line of descent consists of generation 1 and its direct descendants (generation 2, generation 3, and so on). A variant line of descent contains one or more variant generations; for example, the line of descent for generation 3A1B2 consists of the following generations: 1, 2, 3, 3A1, 3A1B1, 3A1B3, and so on. Generation 1 is the beginning of every line of descent.

Some projects require alternate development paths, such as a trial development of a slightly different internal program structure, a change in the scope of an existing program, or a version to run on a different operating system. Variant generations are the mechanism that CMS uses to organize concurrent, parallel changes to a library element.

6.1.1. Creating a Variant Generation

To create a variant generation, use the /VARIANT= *x* qualifier on the REPLACE command. This creates a variant line of descent that CMS can distinguish from the main line of descent. The parameter *x*, called the variant letter, is any single alphabetic character (A through Z). If you enter the variant letter as a lowercase character, CMS converts it to uppercase. CMS copies the replaced file into the library and labels the variant generation by appending the variant letter and the number 1 to the generation number of the ancestor generation.

For example, if you reserved generation 7 of an element named INIT.FOR, you could create a variant as follows:

```
$ CMS REPLACE INIT.FOR/VARIANT=T
_Remark: Routine added for multi-user system
%CMS-S-GENCREATED, generation 7
T1 of element DISKX:[PROJECT.CMSLIB]INIT.FOR created
```

The variant letter (in this case, T) identifies the new line of descent. The variant letter has no meaning to CMS; you can use it for mnemonic purposes. For instance, you can choose a variant letter that indicates the purpose of the variant line, such as F for fixes, E for enhancements, and so forth.

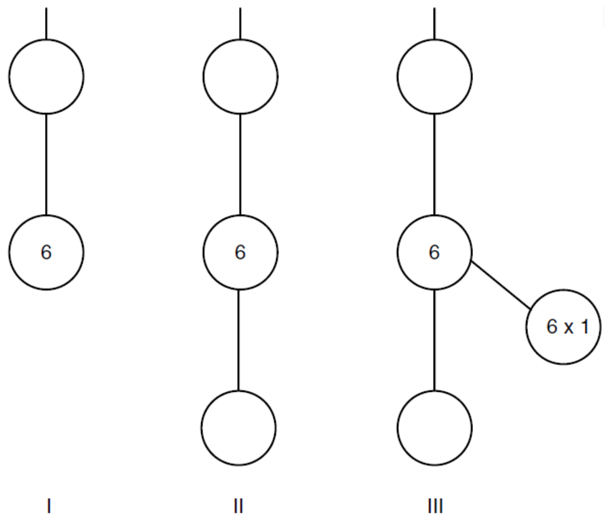
The number after the variant letter identifies successive generations on that new line of descent. For example, if you reserve and replace generation 7T1 of INIT.FOR, generation 7T2 is created. Each variant can have variants of its own using the same method; for example, you could replace a variant to generation 7T1 with the REPLACE/VARIANT=E command to create generation 7T1E1.

You can create a variant line for any reason; however, there are two cases in which you must create a variant in order to replace an element.

First, when two or more reservations are in effect for the same generation of the same element at the same time, all but one of the reservations must be replaced as a variant. CMS manages concurrent changes by allowing only one replacement to become the next generation on the same line of descent. The other replacements must begin variant lines of descent; the changes can then be merged back into the original line of descent (see *Section 6.2.1, "Merging Element Generations"*).

Figure 6.1, "Creating a Variant Generation" shows one element at three different stages of development. In stage I, the element has six generations. At this point, two users reserve generation 6 of the element. The users replace their reservations, creating generation 7 (stage II) and variant generation 6X1 (stage III).

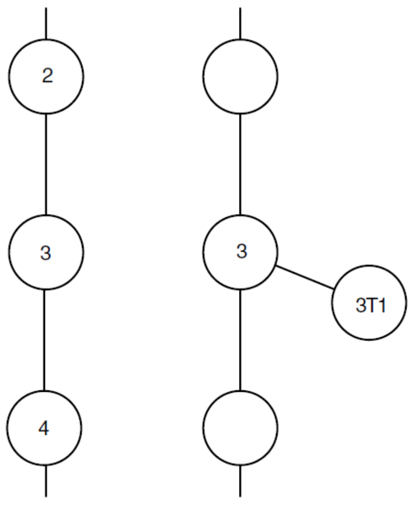
Figure 6.1. Creating a Variant Generation



Second, when you reserve a generation other than the most recent one on a line of descent, you must always create a variant successor because the successor on the same line of descent already exists. For example, if you reserved an earlier generation to modify software that has already been released, you must create a variant to store the modification. The change can then be merged into the original line of descent (see *Section 6.2, "Merging Two Generations of an Element"*).

Figure 6.2, "Extending a Variant Generation from an Earlier Generation" shows one element at two stages of development. If you reserve generation 3 of the element, you must create a variant (shown here as generation 3T1) when you replace the generation with the REPLACE command, because generation 4 already exists.

Figure 6.2. Extending a Variant Generation from an Earlier Generation



6.1.2. Accessing Variant Generations

Variant generation numbers can be used like any other generation numbers. You retrieve a variant generation of an element by using the /GENERATION qualifier with the FETCH or RESERVE command. You must specify a generation number or a class name when you use the /GENERATION qualifier. When you replace a reserved variant generation, the new generation is created on the same variant line. For example:

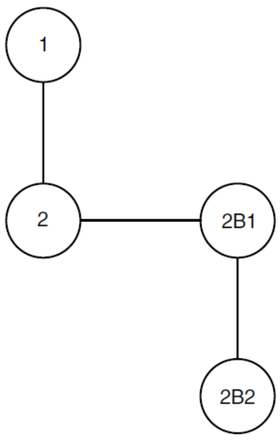
```
$ CMS RESERVE SEMANTICS.PAS/GENERATION=3C1
_Remark: checks for multiple CASE labels
%CMS-S-RESERVED, generation 3C1 of element DISKX:
[PROJECT.CMSLIB]SEMANTICS.PAS reserved
     .
     .
     .
(modify and test element file)
     .
     .
     .
$ CMS REPLACE SEMANTICS.PAS
_Remark: error checking on multiple CASE labels done
%CMS-S-GENCREATED, generation 3C2 of element DISKX:
[PROJECT.CMSLIB]SEMANTICS.PAS created
```

In this example, the /GENERATION qualifier on the RESERVE command specifies that generation 3C1 is to be reserved. The REPLACE command returns the element to the library and creates generation 3C2, which is on the same line of descent as its predecessor, 3C1.

6.1.3. Ancestor and Descendant Generations

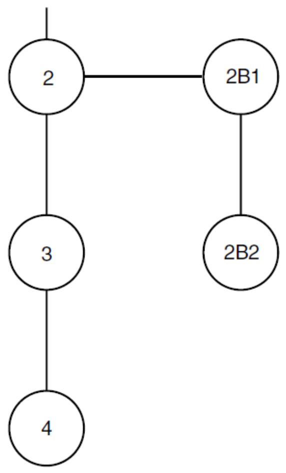
The ancestors of a generation on the main line of descent are all the preceding generations back to the first generation of the element (generation 1). The ancestors of a variant generation are all the preceding generations on the variant line of descent, which includes all generations on the path back to the first generation of the element. *Figure 6.3, "Ancestors on a Tree of Descent"* shows the path to the ancestors of generation 2B2.

Figure 6.3. Ancestors on a Tree of Descent



The descendants of a generation consist of all successive generations (on the same line of descent) and all their variant generations. *Figure 6.4, "Descendants on a Tree of Descent"* shows the paths that connect the descendants of generation 2.

Figure 6.4. Descendants on a Tree of Descent



To display the ancestors or descendants of a generation, use the SHOW GENERATION command with the /ANCESTORS or /DESCENDANTS qualifier, respectively.

6.2. Merging Two Generations of an Element

At some point in the development cycle, you might want to combine changes made in two generations of an element. For example, if concurrent changes are made to a generation of an element, those changes must be replaced as two separate generations, at least one of which must be a variant. The changes made in these new generations can now be merged into a single generation of the element.

Two conditions are necessary for a merge transaction:

- The generations must belong to the same element; that is, you cannot merge generations of different elements.
- One generation cannot be an ancestor of the other; that is, they must be on different lines of descent. For example, in *Figure 6.4, "Descendants on a Tree of Descent"*, you could merge generation 2B1 and 3 or 2B2 and 3, but you could not merge generations 2 and 3, or 2 and 2B1, or 2 and 2B2.

The following sections describe how to merge generations and how the merging process works.

6.2.1. Merging Element Generations

When you merge generations, CMS identifies the generation you specify with the /MERGE qualifier, the generation being fetched or reserved, and all edits on the lines of descent for the two generations. (The two generations used in the merge transaction cannot be on the same line of descent.)

CMS then compares the changes that have been made in both generations being merged against the lines of descent. CMS looks for identical regions of text between each of the generations being merged and the lines of descent. These identical regions provide “anchor” points. The location of changes is determined relative to these anchor points—not relative to any particular line number. For example, CMS could consider line 200 in one generation being merged to be at the same location as line 500 in the other generation.

Any changes found in only one of the generations are included in the new file. These are called successful merges. Identical changes (modifications, insertions, or deletions) made at identical locations in the merged generations are also included in the new file (also called successful merges). Different changes made at identical locations are flagged by CMS and require manual resolution. These changes are called merge conflicts. (*Section 6.2.2, "Conflicts in the Merging Process"* explains how conflicts between two generations are treated in the merging process.) CMS then creates an output file containing the results of the merge transaction, and places it in your current default directory. CMS assigns the current time as the creation and revision time of the output file; the output file does not inherit these values from the reserved generation.

Note

Because of Record Management Services (RMS) record storage restrictions, CMS does not merge element generations that have fixed-length records of different size. CMS does merge element generations that have fixed-length records with identical formats, however. If you try to merge fixed-length records of different size, you receive the following error messages:

```
CMS-E-SIZEMISMAT, cannot merge generations with different size records
CMS-E-GENRECSIZE, generation ## has ##-byte records, ## has ##-byte records
```

If at least one of the merged generations has variable-length records, no restrictions apply, and the resulting generation has variable-length records.

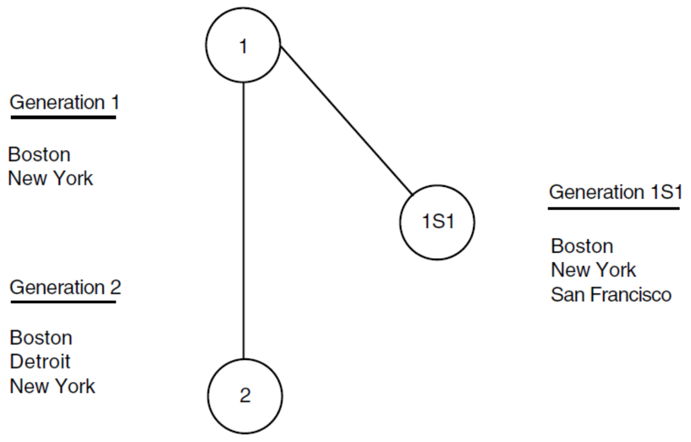
The /MERGE qualifier identifies the element generation that CMS merges into the generation being retrieved with the FETCH or RESERVE command.

For example, the following command merges generation 3A1 of the element DATACHAP.TXT into generation 7B3:

```
$ CMS FETCH DATACHAP.TXT/GENERATION=7B3/MERGE=3A1
```

Figure 6.5, "The Relationship Between a Generation and an Element" shows the contents of three generations of the element CITY.TXT (generations 1, 2, and 1S1) and the relationship between the element generations.

Figure 6.5. The Relationship Between a Generation and an Element



In this example, generation 2 is the most recent on the main line of descent. Therefore, you can merge generations 2 and 1S1 with the following command:

```
$ CMS RESERVE CITY.TXT/MERGE=1S1
_Remark: merge generations 2 and 1S1
%CMS-I-MERGECOUNT, 2 changes successfully merged with no conflicts
%CMS-S-RESERVED, generation 2 of element DISKX:[PROJECT.CMSLIB]CITY.TXT
 reserved and merged with generation 1S1
```

This command merges generation 1S1 into generation 2 of CITY.TXT. The output file (named CITY.TXT) contains the text common to both generations and the changes made to both generations. (The file is placed in your current default directory, or, if you use the /OUTPUT qualifier, another location.) CMS marks generation 2 of the element CITY.TXT as reserved. The generation indicated by the /MERGE qualifier (in this example, generation 1S1) is not reserved.

CMS determines the changes made in each of the generations being merged by comparing them against generation 1, which is the line of descent. In this case, the changes were made to different parts of the file; thus, no conflicts exist. The resulting file looks like this:

```
BOSTON
DETROIT
NEW YORK
SAN FRANCISCO
```

The line DETROIT is the only difference between generation 1 and generation 2. This change occurs after the line BOSTON in the line of descent. The line SAN FRANCISCO is the only difference between generation 1 and generation 1S1. This change occurs after the line NEW YORK in the line of descent. Because the changes in generations 1S1 and 2 occur at different places in the lines of descent, both changes can be applied without conflict.

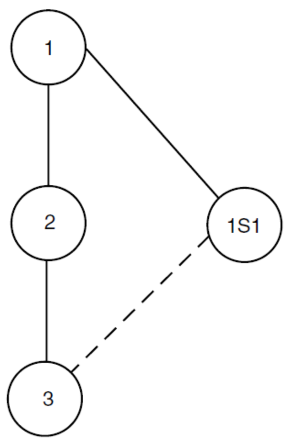
The merge transaction combines two lines of descent in a file outside the library. When you merge with the RESERVE command, you can subsequently replace the element in the library. The following command replaces the file created by merging generation 1S1 into generation 2 of CITY.TXT:

```
$ CMS REPLACE CITY.TXT "completed new format"
%CMS-S-GENCREATED, generation 3 of element DISKX:[PROJECT.CMSLIB]CITY.TXT
 created
```

The generation created by the replacement is a successor only to the generation that was reserved. Because generation 2 was specified as the retrieved generation when it was reserved, the REPLACE command creates generation 3.

Figure 6.6, "A Generation After Replacement in the Library" shows the relationship of the generations of CITY.TXT after the replacement transaction. Note that no ancestor or line of descent relationship exists between generation 1S1 and generation 3.

Figure 6.6. A Generation After Replacement in the Library



If you do not want to create a new generation but want to produce a merged file, use the FETCH/ MERGE command to merge two lines of descent. You can also use the ANNOTATE/MERGE command to create a single file that contains the text common to both generations and the changes made to both generations. For more information, see the ANNOTATE command in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

For information on verifying the merge transaction, see *Section 6.2.3, "Verifying Merged Changes"*.

6.2.2. Conflicts in the Merging Process

Different changes made at identical locations in a generation are called conflicting changes. A conflicting change can be one of the following:

- An insertion of one or more lines
- A deletion of one or more lines
- A replacement of *n* lines by *m* lines (*n* might be equal to *m*)

If CMS detects conflicting changes in the merged generations, it notifies you by including the changes from both generations in the resulting file and surrounding them with asterisks.

Suppose that generation 2 of the element CITY.TXT contains an additional line of text and looks like the following:

```
BOSTON
DETROIT
NEW YORK
PORTLAND
```

Under these circumstances, the same merge transaction described in *Section 6.2.1, "Merging Element Generations"* produces different results:

```
$ CMS RESERVE CITY.TXT/MERGE=1S1
_Remark: merge two generations
%CMS-W-MERGECONFLICT, 1 change successfully merged with 1 conflict
%CMS-S-RESERVED, generation 2 of element DISKX:[PROJECT.CMSLIB]
CITY.TXT reserved and merged with generation 1S1
```

The resulting file looks like this:

```
BOSTON
DETROIT
NEW YORK
*************** Conflict 1
 ***********************************************
PORTLAND
********************************************************************************
SAN FRANCISCO
******** End of Conflict 1
 ***********************************************
```

When the two generations are merged, one change is successfully merged and one conflict exists. The line DETROIT from generation 2 is applied to the lines of descent without conflict. That is, there is no change from generation 1S1 in the same location. However, the line PORTLAND from generation 2 and the line SAN FRANCISCO from generation 1S1 both occur at the same location. Each conflict is flagged with the word “Conflict” and a sequential conflict number in a line of asterisks. (For files with short fixed-length records, CMS attempts to fit the “Conflict” label; if the Conflict label does not fit, CMS outputs only asterisks.) Following the asterisks, CMS displays the conflicting segments of text.

When CMS reports conflicts from a merge transaction, you must resolve conflicting lines with a text editor. For example, you might want to delete one set of changes.

Note

You must delete the conflict flags (the lines containing asterisks). If you do not delete them and the merged element is reserved, the REPLACE command replaces those lines into the library.

6.2.3. Verifying Merged Changes

The merging process is based solely on the text in the files being merged and is performed with no understanding of the meaning of that text. Thus, the resulting file from a “successful” merge might not have the desired form. For example, consider a document where both changes include the same paragraph, but at different places in the file. The successfully merged copy will contain a redundant paragraph. Or consider simultaneous changes made to a code module where one change deleted an unused routine whereas the other called that routine. The merged version would contain the call but no routine to be called, yet the merge would be considered successful by CMS.

You should always verify that the merge transaction had the intended results. You can use the ANNOTATE/MERGE command to produce an annotated listing that shows all changes made to a file, or you can use the DIFFERENCES/FULL command to compare the contents of the files. If you use the DIFFERENCES command, perform the differences transaction three times: once against the new file and each of the merged generations (to ensure that their contents were preserved) and once against the new file and the lines of descent.

In addition, because CMS does not understand the meaning of the text in the files being merged, where applicable you should always compile and link the file as a precautionary measure.

For more information on the ANNOTATE and DIFFERENCES commands, see the online help or the VSI DECset for OpenVMS Code Management System Reference Manual.

Chapter 7. Security Features

You can use two types of security mechanisms to protect your CMS library and the objects in your library:

- Standard OpenVMS file protection mechanisms based on user identification codes (UICs) and access control lists (ACLs)
- CMS ACLs

You use OpenVMS file protection mechanisms to control access to OpenVMS files and directories. In general, UIC-based protection is useful for denying or granting access to a user or group of users (as defined by the UIC group number) or to all users on the system. OpenVMS ACL-based protection is useful for specifying access for a collection of users who are not in the same UIC group.

CMS ACLs are useful for controlling access to CMS objects and to CMS operations (commands) performed on those objects. Generally, you should use CMS ACLs whenever CMS-specific control is needed instead of, or in addition, to OpenVMS protection mechanisms. CMS ACLs are very similar to OpenVMS ACLs; the difference is that although OpenVMS ACLs are used to specify read, write, execute, and delete access, CMS ACLs are used to specify access types for CMS operations.

This chapter describes both security mechanisms; however, you should fully understand the composition of OpenVMS ACLs and their syntax requirements before using CMS ACLs. For more information on ACLs, see the VSI OpenVMS operating system documentation.

7.1. OpenVMS File Access

When you try to access a directory or file, OpenVMS determines whether you are allowed access by checking the protection mask against your UIC (unless there is an ACL on the directory or file that grants immediate access to the directory or file). Specifically, OpenVMS follows these steps to determine whether a user is allowed access to a particular directory or file:

1. It evaluates any ACLs and grants or restricts the associated access.
2. If an ACL does not specifically grant or deny access to the user, or if there is no associated ACL, it uses UIC-based protection to determine access.

BYPASS privilege or GRPPRV, READALL, or SYSPRV privileges can grant the user access, even if it is denied by the UIC- or ACL-based protection schemes.

To fully access a CMS library, you must have the following OpenVMS protection scheme:

- Read and write access to the CMS library directory
- Read and write access to the 01CMS.CMS control file
- Read, write, and delete access to the 01CMS.HIS control file
- Read and delete access to the element data files
- Execute access to the CMS images SYS$SYSTEM:CMS.EXE, SYS$SHARE:CMSSHR.EXE, and SYS$SHARE:CMSPROSHR.EXE

To access a remote CMS library, you need to either:

- Mount the library via the Distributed File System (DFS) and then access it locally on the DFS device.
- Run CMS on the remote node and point its DECwindows interface back to the local node.

To use the default event action handler, you need the following access:

- Execute access to the CMS image SYS$SHARE:CMS$EVENT_ACTION.EXE

To define CMS messages to the OpenVMS Message Utility, you need the following access:

- Execute access to the CMS image SYS$MESSAGE:CMSMSG.EXE

To use the CMS DECwindows Motif interface, you additionally need the following access:

- Read access to the files DECW$SYSTEM_DEFAULTS:CMS$DW.UID, DECW $SYSTEM_DEFAULTS:CMS$DW_DEFAULTS.DAT, and SYS$HELP:CMS$DW_HELP.HLB
- Execute access to the CMS image SYS$SYSTEM:CMS$DW.EXE

If you allow read-only access to a library directory or the 01CMS.CMS file, users cannot make changes to the contents of the library. You must have delete access to an element data file to delete, reserve, or replace a generation of the element. To modify the name of an element, you must have delete access to the element data file and to its corresponding reference copy, if one exists.

You should set up a library so at least one account has read, write, and delete access to every element data file in the library. All three types of access are necessary to execute the VERIFY/RECOVER and VERIFY/REPAIR commands (see *Chapter 9, "Library Maintenance"*).

In some cases, when you use the OpenVMS file protection scheme, the methods you use to manipulate a file might modify certain fields in the file header. When you next use CMS on the library, CMS informs you that some other means has been used to access the library; you must then execute the VERIFY/ REPAIR command (see *Section 9.2.3, "Correcting Errors"*).

The following sections summarize procedures that you can use to define OpenVMS access to your CMS library.

7.1.1. Assigning UIC Protection

UIC-based protection controls access to directories and files as well as other OpenVMS objects. On OpenVMS systems, each user has an associated UIC. Typically, UICs are presented in numeric or alphanumeric format, for example, [221,253], or [PROJECT,JONES].

In addition, every file has a protection mask and owner UIC associated with it. When a user tries to gain access to a directory or file, the system first checks for existing ACLs, then, if none exist, checks the UIC-based protection mask. A UIC protection mask allows or denies the following types of access:

- Read (R)
- Write (W)
- Execute (E)

- Delete (D)

The protection mask describes the categories of users who have access to a directory or file, and the type of access each category has. The categories of users are as follows:

- System (S)
- Owner (O)
- Group (G)
- World (W)

You use the DCL command SET PROTECTION to specify a particular protection mask for a directory and its contents. The following example shows the protection mask you can use to allow system, owner, and group access to the library directory [PROJECT]CMSLIB.DIR:

```
$ SET PROTECTION=(S:RWE,O:RWE,G:RWE,W) [PROJECT]CMSLIB.DIR
```

This protection mask denies world access to the library. Similarly, you can use the SET PROTECTION command to specify a UIC protection mask for an individual file within the library directory. For example:

```
$ SET PROTECTION=(S:RWD,O:RWD,G:RWD,W) [PROJECT.CMSLIB]01CMS.CMS
```

7.1.2. Assigning OpenVMS ACL Protection

An ACL consists of access control entries (ACEs) that grant or deny access to a directory or file (or other OpenVMS object) to specific users. You use ACLs with a library directory to define access to an entire library. You use ACLs with library files to establish greater control over access to library contents. Generally, OpenVMS ACLs are used in conjunction with the standard UIC-based protection as a way to fine-tune protection.

You can use the following DCL commands to manipulate entire OpenVMS ACLs or individual ACEs:

- EDIT/ACL
- SET ACL
- SET FILE/ACL
- SET DEVICE/ACL
- SET DIRECTORY/ACL

You can use the following DCL commands to display OpenVMS ACLs:

- SHOW ACL
- DIRECTORY/ACL
- DIRECTORY/FULL
- DIRECTORY/SECURITY

See the *VSI OpenVMS DCL Dictionary* for more information on these commands. For information on ACLs and ACEs, see the *VSI OpenVMS Guide to System Security*.

7.1.2.1. Using OpenVMS ACLs on Directories

OpenVMS directory ACLs provide three means of controlling access to a directory:

- By controlling access to the directory file itself. For example:

  ```
  $ SET FILE/ACL=(IDENTIFIER=DBASEGRP,ACCESS=READ+WRITE) CMSLIB.DIR
  ```

  This ACE grants read and write access to the directory file CMSLIB.DIR to users who have the DBASEGRP identifier.

- By specifying a default UIC protection mask to be assigned to each new file created in the directory. To specify a particular UIC protection mask, use the DEFAULT_PROTECTION keyword as the first field of an ACE. For example:

  ```
  $ SET FILE/ACL=(DEFAULT_PROTECTION,S:RWED,O:RWED,G:RWED) CMSLIB.DIR
  ```

  This ACE specifies that the UIC protection (S:RWED,O:RWED,G:RWED) be applied to each new file created in the directory. (It does not affect any files that might already exist in the directory.) If no other ACEs impose stricter limitations, the system, owner, and group users are granted full access to new files in the library.

- By specifying a default ACL to be assigned to each file created in the directory. To specify a default ACL, use the OPTIONS=DEFAULT clause in the second field of an ACE that is applied to a directory file. For example:

  ```
  $ SET FILE/ACL=(IDENTIFIER=DBASEGRP,OPTIONS=DEFAULT,ACCESS=READ+ -
  _$ WRITE+DELETE) CMSLIB.DIR
  ```

  The OPTIONS=DEFAULT clause directs the operating system to duplicate this ACE in the ACL of every new file created in the directory. This ACE grants read, write, and delete access to users who have the DBASEGRP identifier.

7.1.2.2. Using OpenVMS ACLs on Files

To exercise greater control over library access, you can explicitly set the file protection for each file in the library. Once you have created the first generation of an element, you can add the necessary ACEs to the ACL for the element data file. Every time you create a new generation of the file, CMS creates a new version of the file in the library directory, and the operating system automatically duplicates the ACL.

For example, you might establish the following ACL for an element data file:

```
$ SET FILE/ACL=(IDENTIFIER=CMSMGR,ACCESS=READ+WRITE+DELETE),-
_$ (IDENTIFIER=[JONES],ACCESS=READ+WRITE+DELETE), -
_$ (IDENTIFIER=[507,*],ACCESS=READ) [PROJECT.CMSLIB.CMS$000]EXAMPLE.PAS
```

This ACL allows both the user with the CMSMGR identifier and user JONES read, write, and delete access to the element EXAMPLE.PAS. Users in the UIC group identified by number 507 can read (fetch), but cannot write (modify) the element.

You must have both read and delete access to an element data file to reserve and replace generations of the corresponding element. If you reserve a generation of an element and then the access changes (so either your account and the element data file ACE no longer have the same identifier, or you no longer have delete access to the element data file), you cannot replace the reserved generation.

Table 7.1, "File Access Required for CMS Commands" shows a list of the CMS commands and the protection required for each object (an element data file, a control file, or a library directory) that the command accesses.

Table 7.1. File Access Required for CMS Commands

Command	Library Directory and Subdirectories	01CMS. CMS	01CMS. HIS	Element Data File	Reference Copy File	Reference Copy Directory
ACCEPT GENERATION	RW	RW	RW			
ANNOTATE	R	R		R		
CANCEL REVIEW	RW	RW	RW			
CONVERT LIBRARY						
—V2-library-name	R	RW	R	R		
—V3-library-name	RW [1]					RW
COPY ELEMENT	RW [2]	RW	RW	R		RW
CREATE CLASS	RW	RW	RW			
CREATE ELEMENT	RW	RW	RW			RW
CREATE GROUP	RW	RW	RW			
CREATE LIBRARY	RW [1]					
DELETE CLASS	RW	RW	RW			
DELETE ELEMENT	RW	RW	RW	RD	RD	RW
DELETE GENERATION	RW	RW	RW	RWD	RWD	RW
DELETE GROUP	RW	RW	RW			
DELETE HISTORY	RW	RW	RWD			
DIFFERENCES [3]	R	R		R		
FETCH	RW [4]	R	RW [4]	R		
INSERT ELEMENT	RW	RW	RW			
INSERT GENERATION	RW	RW	RW			
INSERT GROUP	RW	RW	RW			
MARK GENERATION	RW	RW	RW			
MODIFY CLASS	RW	RW	RW			
MODIFY ELEMENT	RW	RW	RW	RWD	RWD	RW

Command	Library Directory and Subdirectories	01CMS. CMS	01CMS. HIS	Element Data File	Reference Copy File	Reference Copy Directory
MODIFY GENERATION	RW	RW	RW			
MODIFY GROUP	RW	RW	RW			
MODIFY LIBRARY	RW	RW	RW		R	R
MODIFY RESERVATION	RW	RW	RW	RWD	RWD	RW
REJECT GENERATION	RW	RW	RW			
REMARK	RW	RW	RW			
REMOVE ELEMENT	RW	RW	RW			
REMOVE GENERATION	RW	RW	RW			
REMOVE GROUP	RW	RW	RW			
REPLACE	RW	RW	RW	RWD	RWD	RW
RESERVE	RW	RW	RW	RWD		
RETRIEVE ARCHIVE [5]						
REVIEW GENERATION	RW	RW	RW			
SET ACL	RW	RW	RW			
SET LIBRARY	R	R				
SHOW commands	R	R				
SHOW ARCHIVE [5]						
SHOW HISTORY	R	R	R			
SHOW LIBRARY	R					
UNRESERVE	RW	RW	RW			
VERIFY/ RECOVER	RW	RW	RW	RWD		
VERIFY/ REPAIR	RW	RW	RW	RWD	RWD	RW

[1]The directory must be empty.

[2]You must have read access to the source library and both read and write access to the destination library.

[3]You must have access to the library and its contents only when you specify an element generation in the differences transaction.

[4]You must have write access to the library directory and read and write access to the history file only if you enter a remark for the fetch transaction.

[5]You must have read access to the archive file.

If you have set up a restrictive file protection scheme and there is a system failure during a CMS transaction that leaves your library in an inconsistent state, a user with sufficient access to the library and its files should execute the VERIFY/RECOVER command (see *Chapter 9, "Library Maintenance"*). You can also recover the library if you have BYPASS privilege (see *Section 7.3, "OpenVMS BYPASS Privilege and CMS BYPASS Access"*), or read, write, and delete access to all the library files.

7.2. CMS ACLs

A CMS ACL is used to control access to CMS library objects. You can assign CMS ACLs to the following types of objects:

- Elements
- Groups
- Classes
- Element list
- Group list
- Class list
- History
- Library attributes
- Commands

When there is no ACL on a command or other object, access to the command or other object is unrestricted. Assigning an ACL to an object limits access to the specified user or users.

To determine whether access to an object is allowed, CMS evaluates the ACL on that object. If no ACL exists, access to the object is granted. If an ACL does exist, CMS searches the ACL sequentially for the first ACE that the user matches. A match is determined by comparing the identifiers specified in the ACE against the identifiers held by the user. If the user holds all the identifiers specified in the ACE, that ACE is a match. CMS grants the specified access of the first ACE matched; if another ACE further down in the ACL also matches, it has no effect. If none of the ACEs match, access is denied.

Note that if you are granted access to an object by CMS ACLs, you still need access to the files via OpenVMS protection mechanisms. (However, use of the BYPASS privilege will allow you access; see *Section 7.3, "OpenVMS BYPASS Privilege and CMS BYPASS Access"* for more information.)

There are two ways in which you can use CMS ACLs:

- To control and restrict access to CMS commands

  For example, you can create an ACL specifying certain users who are not allowed to use the DELETE ELEMENT command, or users who are allowed to only use the FETCH, RESERVE, and REPLACE commands. See *Section 7.2.2, "Specifying ACLs with Commands"* for more information.

- To control and restrict access to CMS objects

  For example, you can create an ACL specifying certain users who are not allowed to insert or modify a particular element. You can put ACLs on elements, groups, and classes as well as on the element, group, and class lists. You can also put an ACL on the entire library and on the library history. See *Section 7.2.3.2, "Specifying ACLs on Element Lists, Group Lists, and Class Lists"* and *Section 7.2.3.3, "Specifying ACLs on Libraries and History"* for more information.

You can also use CMS ACLs to define CMS events (see *Chapter 8, "Event Handling and Notification"*).

7.2.1. Creating CMS ACLs

An ACL consists of ACEs that grant or deny access to a command or other object to specific users.

You can use two types of ACEs in CMS:

- Identifier ACEs—Control which users can perform which CMS operations on a specified object.
- Action ACEs—Define CMS events and specify actions to be taken when the events occur (these are described in *Chapter 8, "Event Handling and Notification"*).

The following sections describe the format of an ACE and an ACL.

7.2.1.1. ACE Format

An identifier ACE has the following format:

identifier

This field can contain any valid OpenVMS identifier. You use identifiers to specify the users in an ACL. There are three types of identifiers:

- UIC identifiers
- General identifiers
- System-defined identifiers

UIC identifiers are described in *Section 7.1.1, "Assigning UIC Protection"*. General identifiers identify groups of users on the system. For example, CMSPROJ_MEMBR or DBASEGRP are general identifiers. System-defined identifiers are described in the *VSI OpenVMS Guide to System Security*.

You can specify multiple identifiers by separating them with a plus sign (+). The plus sign indicates the logical AND operation. CMS grants the access included in the ACE only for the user who matches all the identifiers. For example:

```
IDENTIFIER=PROJ_LEADER + [PROJ,*]
```

In this example, the multiple identifier is matched only if the user both holds the PROJ_LEADER identifier and belongs to the PROJ group.

options

This field can contain the keyword DEFAULT or NONE. This option is valid only for object lists, that is, lists of elements, groups, or classes. It is not valid for commands. See *Section 7.2.3.2, "Specifying ACLs on Element Lists, Group Lists, and Class Lists"* for more information on the options clause.

access

This field specifies the type of access that CMS allows the user or users identified in the identifier clause of the ACE. You can specify multiple access types by separating them with a plus sign (+). The plus sign indicates the logical OR operation. For example:

```
IDENTIFIER=PROJ_LEADER, ACCESS=MODIFY+DELETE
```

This example indicates that both the modify and delete operations are allowed for the user holding the PROJ_LEADER identifier.

The next section provides more detail on CMS access types.

7.2.1.2. Access Types

Table 7.2, "CMS ACL Access Types" shows all the possible access types for CMS ACLs, along with the object types for which they are meaningful.

Table 7.2. CMS ACL Access Types

Access	Element	Element List	Group	Group List	Class	Class List	History	Library Attributes	Commands
ACCEPT	X	X							
ANNOTATE	X	X							
BYPASS	X	X							
CANCEL	X	X							
CONTROL	X	X	X	X	X	X	X	X	X
COPY	X	X							
CREATE		X		X		X			
DELETE	X	X	X	X	X	X	X		
EXECUTE									X
FETCH	X	X							
INSERT			X	X	X	X			
MARK	X	X							
MODIFY	X	X	X	X	X	X		X	
REJECT							X		
REMARK			X	X	X	X			
REMOVE	X	X						X	
REPAIR	X	X							
REPLACE	X	X							
RESERVE	X	X							
REVIEW	X	X							
UNRESERVE	X	X							
VERIFY	X	X					X	X	

EXECUTE Access

To perform any CMS operation, you must have EXECUTE access to the command, in addition to the appropriate access to the object or objects accessed by the command. For example, to create an element, you need the following access:

- EXECUTE access to the CREATE ELEMENT command

- CREATE access to the element list

To create an element and reserve it, you need the following access:

- EXECUTE access to the CREATE ELEMENT command
- CREATE access to the element list
- EXECUTE access to the RESERVE command
- RESERVE access to the element

To copy an element, in the source library, you need the following access:

- EXECUTE access to the COPY ELEMENT command
- COPY access to the element

To copy an element, in the destination library, you need the following access:

- EXECUTE access to the CREATE ELEMENT command
- CREATE access to the element list

CONTROL Access

To modify or delete an ACL on an object, you must have CONTROL access to the object. In addition, you must have EXECUTE access to the SET ACL command.

You can prevent other users from modifying or deleting an ACL on an object by giving only yourself CONTROL access. Note that at least one user must have CONTROL access; if not, you must use BYPASS privilege to modify or delete that ACL.

See *Section 7.2.2, "Specifying ACLs with Commands"* for information on specifying ACLs on commands. See *Section 7.2.3, "Specifying ACLs with Other CMS Objects"* for more information on specifying ACLs on other object types.

7.2.1.3. ACL Format

You use the CMS SET ACL command to specify ACEs on commands and other objects in the CMS library. The SET ACL command has the following format:

```
SET ACL /OBJECT_TYPE=type object-expression "remark"
```

The object expression depends on the object type; they must be related, as shown in *Table 7.3, "Object Types and Related Expressions"*.

Table 7.3. Object Types and Related Expressions

Object Type	Object Expression
ELEMENT	An element expression
GROUP	A group expression
CLASS	A class expression

Object Type	**Object Expression**
COMMAND	The name of a command, or a list of commands
LIBRARY	ELEMENT_LIST GROUP_LIST CLASS_LIST HISTORY LIBRARY_ATTRIBUTES

If the object type is LIBRARY, the object expression must be one or more keywords (called subtypes), as specified in *Table 7.3, "Object Types and Related Expressions"*. You can abbreviate these subtypes.

The SET ACL command is described in detail in the online help and the VSI DECset for OpenVMS Code Management System Reference Manual. Sections *Section 7.2.2, "Specifying ACLs with Commands"* and *Section 7.2.3, "Specifying ACLs with Other CMS Objects"* describe specifying ACLs with commands and other objects.

7.2.2. Specifying ACLs with Commands

Specifying a CMS ACL on a command enables you to restrict one or more users from accessing that command. This provides a broad protective mechanism that allows greater control over the CMS library than using OpenVMS ACLs and UICs.

You use CMS ACLs on commands and other objects; in most cases, using CMS ACLs on commands is the most effective method to suit most user's needs.

When you use the SET ACL command to set an ACL on a command, the object type must be COMMAND, as specified in *Table 7.3, "Object Types and Related Expressions"*. The object expression must be one of the following commands:

ACCEPT_GENERATION	MARK_GENERATION
ANNOTATE	MODIFY_CLASS
CANCEL_REVIEW	MODIFY_ELEMENT
COPY_CLASS	MODIFY_GENERATION
COPY_ELEMENT	MODIFY_GROUP
COPY_GROUP	MODIFY_LIBRARY
CREATE_CLASS	MODIFY_RESERVATION
CREATE_ELEMENT	REJECT_GENERATION
CREATE_GROUP	REMARK
DELETE_CLASS	REMOVE_ELEMENT
DELETE_ELEMENT	REMOVE_GENERATION
DELETE_GENERATION	REMOVE_GROUP
DELETE_GROUP	REPLACE
DELETE_HISTORY	RESERVE

DIFFERENCES	REVIEW_GENERATION
DIFFERENCES_CLASS	SET_ACL
FETCH	UNRESERVE
INSERT_ELEMENT	VERIFY
INSERT_GENERATION	
INSERT_GROUP	

You can display this list of commands by entering the SHOW ACL /OBJECT_TYPE=COMMAND * command. Note that commands containing two words must include an underscore.

To access a command, you must have EXECUTE access to that command.

7.2.2.1. Examples of ACLs on Commands

1.
   ```
   $ CMS SET ACL/OBJECT_TYPE=COMMAND RESERVE,REPLACE -
   _$ /ACL=(IDENTIFIER=[PROJECT,WILSON],ACCESS=EXECUTE) ""
   ```

   This command specifies that the user with the UIC [PROJECT,WILSON] is allowed EXECUTE access to the RESERVE and REPLACE commands.

2.
   ```
   $ CMS SET ACL/OBJECT_TYPE=COMMAND INSERT_ELEMENT -
   _$ /ACL=(IDENTIFIER=JONES,ACCESS=CONTROL) ""
   %CMS-S-MODACL, modified access control list for command DISKX:
   [PROJECT.CMSLIB]INSERT_ELEMENT

   $ CMS INSERT ELEMENT ELEMENT.2 GROUP2 ""
   %CMS-E-NOINSERT, error inserting DISKX:[PROJECT.CMSLIB]ELEMENT.2
   into group DISKX:[PROJECT.CMSLIB]GROUP2
   -CMS-E-NOACCESS, no execute access to INSERT ELEMENT command
   $ CMS SET ACL/OBJECT_TYPE=COMMAND INSERT_ELEMENT -
   _$ /ACL=(IDENTIFIER=JONES,ACCESS=EXECUTE+CONTROL)""
   %CMS-S-MODACL, modified access control list for command
   DISKX:[PROJECT.CMSLIB]INSERT_ELEMENT
   $ CMS SHOW ACL/OBJECT_TYPE=COMMAND INSERT_ELEMENT
   ACLs in CMS Library DISKX:[PROJECT.CMSLIB]
   INSERT_ELEMENT
            (IDENTIFIER=[WORK,JONES],ACCESS=EXECUTE+CONTROL)
   $ CMS INSERT ELEMENT ELEMENT.2 GROUP2 ""
   %CMS-S-INSERTED, element DISKX:[PROJECT.CMSLIB]ELEMENT.2
   inserted into group DISKX:[PROJECT.CMSLIB]GROUP2
   ```

   In this example, user JONES assigns an ACL containing CONTROL access to the INSERT ELEMENT command. The SHOW ACL command displays the ACL on INSERT ELEMENT. (Note that commands containing more than one word must be specified with an underscore.) The example then shows that JONES tries to insert another element into another group. The attempt fails because, although JONES has CONTROL access to the INSERT ELEMENT command, he does not also have EXECUTE access to it.

   CONTROL access allows you to modify the ACL. Because JONES has CONTROL access, he modifies the ACL to allow himself EXECUTE access to the INSERT ELEMENT command. (You must have EXECUTE access to use any commands.) He can then insert elements successfully.

3.
   ```
   $ CMS SET LIBRARY [WORK.CMSLIB],[PROJECT.CMSLIB]
   %CMS-I-LIBIS, library is DISKX:[WORK.CMSLIB]
   ```

```
%CMS-I-LIBINSLIS, library DISKX:[PROJECT.CMSLIB] inserted at end of
library list
%CMS-S-LIBSET, library set
$ CMS SET ACL/ACL=((IDENTIFIER=SMITH,ACCESS=CONTROL),(IDENTIFIER=*, -
_$ ACCESS=NONE))DELETE_ELEMENT/OBJECT_TYPE=COMMAND/OCCLUDE=NOOTHER ""
%CMS-S-MODACL, modified access control list for command
DISKX:[WORK.CMSLIB]DELETE_ELEMENT
%CMS-S-MODACL, modified access control list for command
DISKX:[PROJECT.CMSLIB]DELETE_ELEMENT
%CMS-S-MODACLS, 2 access control lists modified
```

This example shows the use of occlusion. The SET ACL command is used to restrict access to the DELETE ELEMENT command in both libraries [WORK.CMSLIB] and [PROJECT.CMSLIB]. See *Section 3.3, "Controlling Occlusion in Multiple Libraries"* for more information on occlusion.

7.2.3. Specifying ACLs with Other CMS Objects

For users requiring more restrictive control, you can fine-tune access by using CMS ACLs in combination with objects besides commands. These other objects include:

- Elements, groups, and classes
- Element lists, group lists, and class lists
- Library history and library attributes

The following sections describe these objects in detail.

7.2.3.1. Specifying ACLs on Elements, Groups, and Classes

Specifying a CMS ACL on an element, group, or class enables you to restrict one or more users from accessing that object. For example, you can create an ACL specifying certain users who are not allowed to insert or modify a particular element.

When you use the SET ACL command on an object, the object type must be ELEMENT, GROUP, or CLASS as specified in *Table 7.3, "Object Types and Related Expressions"*. The object expression must be an element, group, or class expression, respectively.

See *Table 7.2, "CMS ACL Access Types"* for all the possible access types that are allowed with these objects. Note that not all access types have meaning for all objects. For example, giving a user RESERVE access to a class is meaningless, because the RESERVE command does not operate on classes.

7.2.3.1.1. Examples of ACLs on Elements, Groups, and Classes

1.
   ```
   $ CMS SET ACL EXAMPLE.PAS/OBJECT_TYPE=ELEMENT -
   _$ /ACL=(IDENTIFIER=[555,*],ACCESS=FETCH) ""
   ```

   This command specifies that users with the UIC [555,*] are allowed only FETCH access to the element EXAMPLE.PAS.

2.
   ```
   $ CMS SET ACL/OBJECT_TYPE=ELEMENT ELEMENT.1 -
   _$ /ACL=(IDENTIFIER=JONES,ACCESS=RESERVE+CONTROL)""
   %CMS-S-MODACL, modified access control list for element
   DISKX:[PROJECT.CMSLIB]ELEMENT.1
   $ CMS SET ACL/OBJECT_TYPE=ELEMENT ELEMENT.1/ACL=(IDENTIFIER=JONES, -
   ```

```
_$ ACCESS=NONE) ""
%CMS-S-MODACL, modified access control list for element
DISKX:[PROJECT.CMSLIB]ELEMENT.1
$ CMS RESERVE ELEMENT.1 ""
%CMS-E-NOFETCH, error reserving element
DISKX:[PROJECT.CMSLIB]ELEMENT.1
-CMS-E-NOACCESS, no reserve access to element ELEMENT.1
$ CMS SET ACL/OBJECT_TYPE=ELEMENT ELEMENT.1 -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=RESERVE+CONTROL) ""
%CMS-E-NOMODACL, error modifying access control list for element
DISKX:[PROJECT.CMSLIB]ELEMENT.1
-CMS-E-NOACCESS, no control access to element ELEMENT.1
```

In this example, user JONES assigns an ACL containing RESERVE and CONTROL access to the element ELEMENT.1. Then, another user (who has BYPASS privilege) sets an ACL on ELEMENT.1 containing ACCESS=NONE, thus restricting JONES from reserving that element, and removing any prior access that JONES had assigned. JONES then tries to reserve the element. His attempt is unsuccessful because he no longer has RESERVE access to the element. He also does not have CONTROL access to the element, which would allow him to modify the ACL assigned by the second user.

3.
```
$ CMS SET ACL/OBJECT_TYPE=CLASS CLASS1 -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=CONTROL) ""
%CMS-S-MODACL, modified access control list for class
DISKX:[PROJECT.CMSLIB]CLASS1
$ CMS SHOW ACL/OBJECT_TYPE=CLASS CLASS1
ACLs in CMS Library DISKX:[PROJECT.CMSLIB]CLASS1
         (IDENTIFIER=[WORK,JONES],ACCESS=CONTROL)
$ CMS MODIFY CLASS/NOREAD_ONLY CLASS1 ""
%CMS-E-NOMODIFY, error modifying class
DISKX:[PROJECT.CMSLIB]CLASS1
-CMS-E-NOACCESS, no modify access to class CLASS1
$ CMS SET ACL/OBJECT_TYPE=CLASS CLASS1 -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=MODIFY+CONTROL) ""
%CMS-S-MODACL, modified access control list for class
DISKX:[PROJECT.CMSLIB]CLASS1
$ CMS MODIFY CLASS/NOREAD_ONLY CLASS1 ""
%CMS-S-MODIFIED, class DISKX:[PROJECT.CMSLIB]CLASS1 modified
```

In this example, user JONES assigns an ACL giving himself CONTROL access to the class CLASS1. He then tries to modify the class, but is unsuccessful because, although he has CONTROL access to the class, he does not also have MODIFY access. However, because JONES has CONTROL access, this allows him to enter the SET ACL command. He then assigns another ACL containing both CONTROL and MODIFY access to the class, then successfully modifies the class.

7.2.3.2. Specifying ACLs on Element Lists, Group Lists, and Class Lists

The difference between an object and its list is important in the understanding of CMS ACLs. Conceptually, element, group, and class lists are objects representing all the elements, groups, and classes already existing or yet to be created in a CMS library. Object lists are used solely with CMS ACLs.

When you use the SET ACL command on an object list, the object type must be LIBRARY. The object expression must be one of the following keywords: ELEMENT_LIST, GROUP_LIST, or CLASS_LIST, as specified in *Table 7.3, "Object Types and Related Expressions"* (see *Section 7.2.3.3, "Specifying ACLs on Libraries and History"* for information on the HISTORY and LIBRARY_ATTRIBUTES keywords).

See *Table 7.2, "CMS ACL Access Types"* for all the possible access types that are allowed with these objects.

Specifying an ACL on an object list grants the right to create new objects in the library. For example:

```
$ CMS SET ACL/OBJECT_TYPE=LIBRARY GROUP_LIST -
_$ /ACL=(IDENTIFIER=PROJ_TEAM,ACCESS=CREATE) ""
```

This example assigns an ACL to the group list, and allows only the holders of the identifier PROJ_TEAM to create groups in the library.

You can also specify a default ACL to be used on newly created objects in the library. You do this by specifying the OPTIONS=DEFAULT clause in the ACL of an object list. For example:

```
$ CMS SET ACL/OBJECT_TYPE=LIBRARY ELEMENT_LIST/ACL=(IDENTIFIER=PROJ_TEAM, -
_$ OPTIONS=DEFAULT, ACCESS=FETCH) ""
```

This example specifies that only holders of the PROJ_TEAM identifier can FETCH newly created elements.

Each time you create a new object, CMS searches for the ACEs containing the OPTIONS=DEFAULT clause in the ACL of the corresponding object list. If any exist, the newly created object (or objects) are automatically assigned the ACEs containing the OPTIONS=DEFAULT clause. For example, if you specify ACEs containing OPTIONS=DEFAULT in the ACL of a group list, CMS assigns the default ACEs in the ACL to any newly created groups.

OPTIONS=DEFAULT is valid only for object lists. Note that the OPTIONS=DEFAULT clause does not affect any objects already in the list, only new objects. You can assign default ACEs to existing objects by specifying the SET ACL/DEFAULT command.

Because it is not possible to assign an ACL granting CREATE access to an object that does not yet exist, the only access types that are meaningful for an object list ACE not containing the OPTIONS=DEFAULT clause are CREATE and CONTROL access. All other access types are meaningful only if the OPTIONS=DEFAULT clause is present.

Caution

Because default ACEs do not grant access, when you use default ACEs, you should assign another ACE granting yourself or another user a minimum of CONTROL access to an object; otherwise, you could restrict your own access to the object.

When you use the COPY ELEMENT command, the source element's ACL is not assigned to the target element. Instead, the target element receives the default ACL (if any) that is set on the element list.

If you do not use the OPTIONS=DEFAULT clause, newly created objects are not affected by the ACL (if any) on the object list. The OPTIONS=NONE clause indicates that new objects are not assigned that ACE from the object list. NONE is equivalent to not specifying a clause. Note that the OPTIONS=NONE clause is not displayed when you enter the SHOW ACL command.

7.2.3.2.1. Examples of ACLs on Lists

1.
   ```
   $ CMS SET ACL/OBJECT_TYPE=LIBRARY ELEMENT_LIST -
   _$ /ACL=((IDENTIFIER=JONES,OPTIONS=DEFAULT,ACCESS=RESERVE -
   ```

```
_$ +CONTROL),(IDENTIFIER=JONES,ACCESS=CREATE+CONTROL)) ""
```

This command places two ACEs on the element list. The first ACE is a default ACE, which causes all new elements created in the library to inherit an ACE giving RESERVE access to the user with the identifier JONES. The second ACE defines the access to the element list itself. Because CREATE access is specified, the user with the identifier JONES is allowed to create elements in the library. Note that both ACEs also grant control access; this is necessary to allow modification of the ACL once it has been created.

2.
```
$ CMS SET ACL/OBJECT_TYPE=LIBRARY CLASS_LIST -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=CONTROL) ""
%CMS-S-MODACL, modified access control list for subtype
DISKX:[PROJECT.CMSLIB]CLASS_LIST
$ CMS CREATE CLASS CLASS4 ""
%CMS-E-NOCREATE, error creating class
DISKX:[PROJECT.CMSLIB]CLASS4
-CMS-E-NOACCESS, no create access to CLASS_LIST
$ CMS SET ACL/OBJECT_TYPE=LIBRARY CLASS_LIST -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=CREATE+CONTROL) ""
%CMS-S-MODACL, modified access control list for subtype DISKX:
[PROJECT.CMSLIB]CLASS_LIST
$ CMS SHOW ACL/OBJECT_TYPE=LIBRARY CLASS_LIST
ACLs in CMS Library DISKX:[PROJECT.CMSLIB]
CLASS_LIST
         (IDENTIFIER=[WORK,JONES],ACCESS=CREATE+CONTROL)
$ CMS CREATE CLASS CLASS4 ""
%CMS-S-CREATED, class DISKX:[PROJECT.CMSLIB]CLASS4 created
```

In this example, JONES assigns an ACL containing CONTROL access to the class list. Assigning an ACL to the class list will affect the creation of new classes in the library. However, when he tries to create a new class, he receives an error because he does not also have CREATE access to the class list. Because he has CONTROL access, he then assigns a new ACL giving himself both CONTROL and CREATE access. He can then create new classes.

3.
```
$ CMS SET ACL/OBJECT_TYPE=LIBRARY ELEMENT_LIST -
_$/ACL=((IDENTIFIER=JONES,ACCESS=CREATE+CONTROL), -
_$
(IDENTIFIER=FLYNN,OPTIONS=DEFAULT,ACCESS=FETCH), -
_$ (IDENTIFIER=SMITH,OPTIONS=DEFAULT,ACCESS=RESERVE+REPLACE)) ""
%CMS-S-MODACL, modified access control list for subtype
DISKX:[PROJECT.CMSLIB]ELEMENT_LIST
$ CMS SHOW ACL/OBJECT_TYPE=LIBRARY ELEMENT_LIST
ACLs in CMS Library DISKX:[PROJECT.CMSLIB]
ELEMENT_LIST
         (IDENTIFIER=[WORK,JONES],ACCESS=CREATE+CONTROL)
         (IDENTIFIER=[WORK,FLYNN],OPTIONS=DEFAULT,ACCESS=FETCH)
         (IDENTIFIER=[WORK,SMITH],OPTIONS=DEFAULT,ACCESS=REPLACE
+RESERVE)
$ CMS CREATE ELEMENT ELEMENT.4 ""
%CMS-S-CREATED, element DISKX:[PROJECT.CMSLIB]ELEMENT.4 created
$ CMS SHOW ACL/OBJECT_TYPE=ELEMENT ELEMENT.4
ACLs in CMS Library DISKX:[PROJECT.CMSLIB]
ELEMENT.4
         (IDENTIFIER=[WORK,FLYNN],ACCESS=FETCH)
         (IDENTIFIER=[WORK,SMITH],ACCESS=REPLACE+RESERVE)
```

In this example, user JONES assigns an ACL on the element list. The ACL specifies the following:

- JONES is allowed CREATE and CONTROL access to the element list.
- By using OPTIONS=DEFAULT, JONES assigns user FLYNN only FETCH access to new elements created in the library.
- By using OPTIONS=DEFAULT, JONES assigns user SMITH only REPLACE and RESERVE access to new elements created in the library.

JONES then successfully creates a new element named ELEMENT.4. When the SHOW ACL command is entered, the default access on the element for each user is displayed. User JONES's access is not displayed because he has access to the element list, not the element itself.

7.2.3.3. Specifying ACLs on Libraries and History

Specifying a CMS ACL on the library or the library history enables you to restrict one or more users from certain types of access to the library, or from certain types of access to the library history. You can restrict users from the following types of access to the library:

- MODIFY
- REPAIR
- VERIFY

You can restrict users from the following types of access to the library history:

- DELETE
- REMARK

REPAIR and VERIFY Access

REPAIR access is required to use VERIFY/REPAIR on a library. If you have REPAIR access to a library object, you can enter VERIFY/REPAIR, even if you do not have VERIFY access to that library.

Because CMS cannot determine whether access control information is valid until it verifies the database, the VERIFY and REPAIR access types apply only to element data file verification. Once the database has been verified, CMS checks the following:

- Access to the VERIFY command
- VERIFY or REPAIR access to the library
- VERIFY or REPAIR access to each element

When you use the SET ACL command on a library or history, the object type must be LIBRARY, as specified in *Table 7.3, "Object Types and Related Expressions"*. The object expression must be either LIBRARY_ATTRIBUTES or HISTORY.

See *Table 7.2, "CMS ACL Access Types"* for all the possible access types allowed on a library or history.

7.2.3.3.1. Examples of ACLs on History and the Library

1.
```
$ CMS SET ACL/OBJECT_TYPE=LIBRARY HISTORY -
```

```
_$ /ACL=(IDENTIFIER=JONES,ACCESS=CONTROL) ""
%CMS-S-MODACL, modified access control list for subtype
DISKX:[PROJECT.CMSLIB]HISTORY
$ CMS REMARK "Add a remark to history"
%CMS-E-NOREMARK, error adding remark to library
-CMS-E-NOACCESS, no remark access to library history
$ CMS SET ACL/OBJECT_TYPE=LIBRARY HISTORY -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=REMARK+CONTROL) ""
%CMS-S-MODACL, modified access control list for subtype
DISKX:[PROJECT.CMSLIB]HISTORY
$ CMS REMARK "Add a remark to history"
%CMS-S-REMARK, remark added to history file
```

In this example, JONES assigns an ACL giving himself CONTROL access to the library history. He then tries to add a remark to the library history, but is unsuccessful because he does not have REMARK access to the history. He then assigns another ACL containing both CONTROL and REMARK access, and can then successfully add a remark to the library history file.

2.
```
$ CMS SET ACL/OBJECT_TYPE=LIBRARY LIBRARY_ATTRIBUTES -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=CONTROL) ""
%CMS-S-MODACL, modified access control list for subtype
DISKX:[PROJECT.CMSLIB]LIBRARY_ATTRIBUTES
$ CMS VERIFY/REPAIR
%CMS-I-VERCLS, class list verified
%CMS-I-VERCMD, command list verified
%CMS-I-VERELE, element list verified
   .
   .
   .
%CMS-I-VERCON, control file verified
%CMS-E-ERRVEREDFS, element data files verified with errors
-CMS-E-NOACCESS, no repair access to library DISKX:[PROJECT.CMSLIB]
%CMS-E-NOREPAIR, error repairing library
$ CMS SET ACL/OBJECT_TYPE=LIBRARY LIBRARY_ATTRIBUTES -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=CONTROL+REPAIR) ""
%CMS-S-MODACL, modified access control list for subtype
     DISKX:[PROJECT.CMSLIB]LIBRARY_ATTRIBUTES
$ CMS SHOW ACL/OBJECT_TYPE=LIBRARY LIBRARY_ATTRIBUTES
ACLs in CMS Library DISKX:[PROJECT.CMSLIB]LIBRARY_ATTRIBUTES
          (IDENTIFIER=[WORK,JONES],ACCESS=CONTROL+REPAIR)
$ CMS VERIFY/REPAIR
    %CMS-I-VERCLS, class list verified
    %CMS-I-VERCMD, command list verified
    %CMS-I-VERELE, element list verified
        .
        .
        .
    %CMS-I-VERCON, control file verified
    %CMS-E-VEREDFERR, element DISKX:[PROJECT.CMSLIB]ELEMENT.1 verified
 with errors
    -CMS-E-NOACCESS, no repair access to element ELEMENT.1
    %CMS-I-VEREDF, element DISKX:[PROJECT.CMSLIB]ELEMENT.2 verified
    %CMS-I-VEREDF, element DISKX:[PROJECT.CMSLIB]ELEMENT.3 verified
    %CMS-E-VEREDFERR, element DISKX:[PROJECT.CMSLIB]ELEMENT.4 verified
 with errors
    -CMS-E-NOACCESS, no repair access to element ELEMENT.4
    %CMS-E-ERRVEREDFS, element data files verified with errors
    %CMS-E-NOREPAIR, error repairing library
```

```
$ CMS SHOW ACL/OBJECT_TYPE=ELEMENT ELEMENT.1, ELEMENT.4
 ACLs in CMS Library DISKX:[PROJECT.CMSLIB]
ELEMENT.1
          (IDENTIFIER=[WORK,JONES],ACCESS=NONE)
ELEMENT.4
          (IDENTIFIER=[WORK,FLYNN],ACCESS=FETCH)
          (IDENTIFIER=[WORK,SMITH],ACCESS=REPLACE+RESERVE)
```

This example demonstrates how REPAIR access is used. First, JONES assigns an ACL to the library indicating that he is allowed CONTROL access to the library. He then tries a VERIFY/REPAIR operation on the library. This attempt is unsuccessful because he does not also have REPAIR access to the library. He assigns a new ACL containing both CONTROL and REPAIR access to the library, and tries another VERIFY/REPAIR operation on the library. This attempt is also unsuccessful because, although he has REPAIR access to the library, he does not have REPAIR access to the elements ELEMENT.1 and ELEMENT.4 (as displayed by the SHOW ACL command). When entering VERIFY/REPAIR, you must have REPAIR access both to the library and to the individual elements in the library.

7.3. OpenVMS BYPASS Privilege and CMS BYPASS Access

The OpenVMS BYPASS privilege allows a user read, write, execute, and delete access to all files, bypassing UIC protection. A user holding BYPASS privilege is also granted access to any CMS object or command, regardless of any OpenVMS or CMS protections.

Whenever you define ACLs for objects, remember that users with BYPASS privilege are granted complete access; for this reason, BYPASS privilege is usually reserved for experienced users who need this privilege.

Being granted CMS BYPASS access is not equivalent to holding OpenVMS BYPASS privilege. The CMS BYPASS access type allows you *only* to unreserve or replace another user's reservation for an element. (OpenVMS BYPASS privilege also allows you to unreserve or replace another user's reservation.)

The following example shows the use of CMS BYPASS access:

```
$ CMS SHOW RESERVATIONS
Reservations in CMS Library DISKX:[PROJECT.CMSLIB]ELEMENT.2
    (1)   FLYNN      1      12-JAN-1998 18:57:43 ""
$ CMS REPLACE ELEMENT.2/IDENTIFICATION_NUMBER=1 ""
%CMS-E-NOREPLACE, error replacing DISKX:[PROJECT.CMSLIB]ELEMENT.2
-CMS-E-IDENTNOTRES, reservation 1 is not reserved by you
$ CMS SET ACL/OBJECT_TYPE=ELEMENT ELEMENT.2 -
_$ /ACL=(IDENTIFIER=JONES,ACCESS=BYPASS+REPLACE+CONTROL) ""
%CMS-S-MODACL, modified access control list for element
DISKX:[PROJECT.CMSLIB]ELEMENT.2
$ CMS SHOW ACL/OBJECT_TYPE=ELEMENT ELEMENT.2
ACLs in CMS Library DISKX:[PROJECT.CMSLIB]
ELEMENT.2
         (IDENTIFIER=[WORK,JONES],ACCESS=CONTROL+BYPASS+REPLACE)
$ CMS REPLACE ELEMENT.2/IDENTIFICATION_NUMBER=1 ""
Element DISKX:[PROJECT.CMSLIB]ELEMENT.2 currently reserved by:
    (1)   FLYNN      1      12-JAN-1998 18:57:43 ""
Replace (1) ELEMENT.2 generation 1, held by FLYNN? [Y/N] (N): Y
```

```
%CMS-S-GENCREATED, generation 2 of element DISKX:[PROJECT.CMSLIB]ELEMENT.2
 created
```

This example shows the use of BYPASS access to replace another user's reservation. The user JONES unsuccessfully tries to replace FLYNN's reservation 1 of the element ELEMENT.2. JONES then assigns an ACL allowing him CONTROL, BYPASS, and REPLACE access to the element. CONTROL allows him to modify the ACL again after he replaces the element. BYPASS allows him to replace FLYNN's reservation. REPLACE is needed to perform the actual replacement. Both BYPASS and REPLACE are required; he can then successfully replace FLYNN's reservation of the element.

7.4. Combining OpenVMS and CMS Security Mechanisms

When CMS ACLs are used in conjunction with OpenVMS protection mechanisms, you should ensure that you allow sufficient access via OpenVMS protection so that all users can perform necessary operations, but you should not allow unnecessary access. In other words, you should set the OpenVMS file protections to allow only as much access as is needed by users to perform operations, as shown in *Table 7.1, "File Access Required for CMS Commands"*. (A set of users can be defined by their UIC, identifiers, or both.)

If a set or sets of users still need to perform a subset of operations beyond the OpenVMS protection you have set up, you can use CMS ACLs to obtain a more restrictive protection scheme.

For example, suppose a group of CMS users is divided into those holding the identifier LIBRARIAN, and those holding the identifier PROGRAMMER. Members of both groups are allowed to reserve elements, but only holders of the LIBRARIAN identifier are allowed to replace them.

As listed in *Table 7.1, "File Access Required for CMS Commands"*, both the RESERVE and REPLACE commands require the same access to all files in the library. Thus, allowing users holding the PROGRAMMER identifier sufficient access to the library files to perform a reserve operation implicitly allows them access to perform a replace operation. Using OpenVMS file protection mechanisms, it is not possible to allow access to RESERVE while disallowing access to REPLACE. However, in CMS, you can place a CMS ACL on the REPLACE command that allows access to holders of the LIBRARIAN identifier, but disallows access to holders of the PROGRAMMER identifier.

To successfully operate in a CMS library, the library directory, control files, and element data files must be accessible through the OpenVMS system (including ACLs and UIC protection mechanisms). In addition, the commands you enter in the library, and the objects referenced by those commands, must be accessible through the CMS ACL mechanism.

Note

The use of both OpenVMS and CMS ACLs does not ensure complete library security. The library can still be accessed using means other than through a CMS interface. However, keep in mind that accessing a library by means other than CMS, such as copying the file through a DCL command, can result in unrecoverable library corruption.

7.4.1. Example of Protection Scheme Using OpenVMS and CMS Mechanisms

This example shows a possible protection scheme using both OpenVMS and CMS security mechanisms.

Suppose a project team consists of the members Smith, Brown, Jones, Anderson, and Nelson. Smith is the project leader, Brown and Jones are senior developers, and Anderson and Nelson are junior developers. All project team members except Nelson hold the PROJECT identifier.

These project members require the following types of access to the library:

- Smith requires full access to the library.
- Brown and Jones are allowed to perform all operations except DELETE ELEMENT and DELETE GENERATION.
- Anderson is allowed to perform all operations except DELETE ELEMENT, DELETE GENERATION, and REPLACE.
- Nelson is allowed access only to the FETCH command.

In this example, the access required to the library files is set according to *Table 7.1, "File Access Required for CMS Commands"*. An OpenVMS ACL for each file could be set up as follows:

1.
```
Library Directory and Subdirectories:
        (IDENTIFIER=PROJECT, ACCESS=READ+WRITE)
        (IDENTIFIER=NELSON, ACCESS=READ)
        (IDENTIFIER=*, ACCESS=NONE)
        (IDENTIFIER=PROJECT,OPTIONS=DEFAULT,ACCESS=READ+WRITE+DELETE)
        (IDENTIFIER=NELSON,OPTIONS=DEFAULT,ACCESS=READ)
        (IDENTIFIER=*,OPTIONS=DEFAULT,ACCESS=NONE)

01CMS.CMS:
        (IDENTIFIER=PROJECT, ACCESS=READ+WRITE)
        (IDENTIFIER=NELSON, ACCESS=READ)
        (IDENTIFIER=*, ACCESS=NONE)

01CMS.HIS:
        (IDENTIFIER=PROJECT, ACCESS=READ+WRITE+DELETE)
        (IDENTIFIER=NELSON, ACCESS=NONE)
        (IDENTIFIER=*, ACCESS=NONE)

Element Data Files:
        (IDENTIFIER=PROJECT, ACCESS=READ+WRITE+DELETE)
        (IDENTIFIER=NELSON, ACCESS=READ)
        (IDENTIFIER=*, ACCESS=NONE)
```

   Note that Nelson is only allowed to use the FETCH command, without specifying a remark. This is due to Nelson's lack of access to the library directory and 01CMS.HIS. Also note that the ACE containing the ACCESS=NONE clause denies access to all library files to anyone not on the project team. The OPTIONS=DEFAULT ACEs on the library directory ensure that newly created element data files receive the proper ACL.

   Although the ACLs assigned to the library files provide the access needed by the members of the project team, they still do not sufficiently restrict access as originally required. To do this, CMS ACLs must be set up on the various commands. To ensure that these ACLs are not changed except by the project leader, an additional requirement is that only Smith can use the SET ACL command. Smith must also have CONTROL access to each of the commands in order to change their ACLs once they have been assigned. The CMS ACLs could be set up as follows:

2.
```
FETCH:
        (IDENTIFIER=SMITH, ACCESS=EXECUTE+CONTROL)
        (IDENTIFIER=PROJECT, ACCESS=EXECUTE)
        (IDENTIFIER=NELSON, ACCESS=EXECUTE)

DELETE ELEMENT and DELETE GENERATION:
        (IDENTIFIER=SMITH, ACCESS=EXECUTE+CONTROL)

REPLACE:
        (IDENTIFIER=ANDERSON, ACCESS=NONE)
        (IDENTIFIER=SMITH, ACCESS=EXECUTE+CONTROL)
        (IDENTIFIER=PROJECT, ACCESS=EXECUTE)

SET ACL:
        (IDENTIFIER=SMITH, ACCESS=EXECUTE+CONTROL)

All other commands:
        (IDENTIFIER=SMITH, ACCESS=EXECUTE+CONTROL)
        (IDENTIFIER=PROJECT, ACCESS=EXECUTE)
```

If an identifier does not match any ACE in an ACL (assuming an ACL exists) CMS denies access to the object. Thus, Nelson is denied access to all commands except FETCH. Even though Anderson holds the PROJECT identifier, he matches the first ACE in the ACL on the REPLACE command, and so is also denied access. Similarly, the ACE for Smith must be placed before the ACE for PROJECT; otherwise, Smith will match the PROJECT ACE and would not receive CONTROL access.

Chapter 8. Event Handling and Notification

You can specify lists of people who are to be notified when certain events occur in the library. An **event** is an operation involving one or more of the following objects:

- Elements
- Element list
- Classes
- Class list
- Groups
- Group list
- History
- Commands
- Library attributes

The following sections describe how to specify events, use the default or a user-written handler, and use notification. *Section 8.3, "Examples"* shows examples of using notification.

8.1. Event Handling

You specify and detect events by using CMS access control lists (ACLs) and access control entries (ACEs). CMS notifies users of events by processing one or more action ACEs in an object's ACL. The following sections describe how to specify events and how events are detected by means of action ACEs.

8.1.1. Specifying Action ACEs

CMS ACLs support two types of ACEs: identifier ACEs and action ACEs. You use identifier ACEs to control which users can perform which CMS operations on a specified object (see *Section 7.2.1, "Creating CMS ACLs"*). You use action ACEs to define CMS events. An action ACE enables you to specify a particular action to be taken when a CMS object is accessed in a certain way.

An action ACE has the following format:

(ACTION[=image], PARAMETER=string [,IDENTIFIER=identifier] [,OPTIONS=options] [,ACCESS=access])

The ACTION clause identifies the ACE as an action ACE; you can optionally use it to specify a shareable image containing your own event-handler routine, CMS$EVENT_ACTION (see *Section 8.1.3, "Using Your Own Event Handler"*). Do not include the .EXE file extension in the event-handler name. If you do not specify a user-written, event-handler routine on the ACTION clause, CMS uses the default event handler SYS$SHARE:CMS$EVENT_ACTION image.

If you use the default image, the string specified on the PARAMETER clause must be a valid MAIL recipient specification, such as MYNODE::JONES or @DISTLIST, or a list of specifications separated with commas. You might need to enclose the string in quotation marks if the string contains a list, period, comma, or other non-alphanumeric characters. You should also enclose the string in quotation marks when differentiating between uppercase and lowercase, or CMS will convert the string to uppercase.

You can use the NOTIFY clause as a synonym for the ACTION,PARAMETER= (that is, the ACTION clause without the =image parameter) when you specify an action ACE. For example, the following specifications are equivalent:

```
NOTIFY=@LIST
(ACTION, PARAMETER=@LIST)
```

You cannot use the NOTIFY clause, however, if you specify a user-written handler.

The IDENTIFIER clause is optional in action ACEs. If it is not specified, CMS assumes the IDENTIFIER=* clause by default. The IDENTIFIER, OPTIONS and ACCESS clauses are described in detail in *Section 7.2.1, "Creating CMS ACLs"*.

See *Section 8.3, "Examples"* for examples of action ACEs.

8.1.2. Detecting Events

A CMS event occurs only when the user has been granted the right to perform the operation and the operation has been successfully performed. Therefore, you cannot use an event handler to prevent a command from performing its operation, nor does the command fail if the event handler cannot be invoked.

Multiple events can occur as a result of a single CMS command being executed. For example, if action ACEs have been assigned to the elements A.TXT and B.TXT and to the command RESERVE, three independent events can be triggered by the command RESERVE A.TXT,B.TXT, one for each of the three objects.

An exception is the INSERT and REMOVE commands. Execution of the INSERT and REMOVE commands involves two types of objects: the objects being inserted or removed, and the objects being inserted into or removed from. CMS does not check ACLs associated with objects of the first type; therefore, insert and remove operations involving objects of the first type cannot trigger events. For example:

```
$ CMS INSERT ELEMENT A.TXT,B.TXT TEST_BAS ""
```

This command could trigger an event associated with the group TEST_BAS, but not with the elements A.TXT and B.TXT.

8.1.3. Using Your Own Event Handler

When CMS detects that a specified event has occurred, it invokes the event handler routine CMS $EVENT_ACTION in the SYS$SHARE:CMS$EVENT_ACTION image, or, if you have written your own shareable image, in your user-provided image.

You specify your own shareable image name in the ACTION clause of the ACE that is defining the event. You do this by specifying the image name after the ACTION keyword, as follows:

```
ACTION=image_name
```

See *Section 8.1.1, "Specifying Action ACEs"* for more information.

You must include a routine named CMS$EVENT_ACTION in your image. CMS dynamically activates the CMS$EVENT_ACTION routine's image, if necessary, by calling LIB$FIND_IMAGE_SYMBOL, then calls the CMS$EVENT_ACTION routine.

Your CMS$EVENT_ACTION routine should follow the rules for callback routines (see the online callable routines file). The routine calling format and arguments for CMS$EVENT_ACTION are as follows:

```
CMS$EVENT_ACTION (library_data_block,
                  user_param,
                  library_specification_id,
                  ace_parameter_id,
                  history_record_id)
```

library_data_block

Specifies the library data block (LDB) for the current library.

user_param

[ ]

Specifies the **user_arg** value passed in a call to a callable CMS routine whose action caused the event. If no user argument was specified in the user call, or if the event did not occur as a result of a user call to a callable CMS routine, the call-frame entry for **user_param** points to a location containing the value zero. In this case, **user_param** is allocated as read-only storage.

library_specification_id

Specifies a string identifier for the current CMS library directory specification.

ace_parameter_id

Specifies a string identifier for the string found in the PARAMETER clause of the current ACE (the ACE that defines the current event).

history_record_id

Specifies a string identifier for the string containing the history record written as a result of the current CMS operation (the operation that caused the current event).

The **library_data_block** argument should be used only by the default CMS$EVENT_ACTION routine; any user-written CMS$EVENT_ACTION routine should ignore it. The **user_param** argument is provided so a user-written CMS$EVENT_ACTION routine can interpret it; the default CMS $EVENT_ACTION routine ignores it. If you use the default CMS$EVENT_ACTION routine, CMS expects the **ace_parameter_id** argument to point to a string containing a list of valid MAIL recipient specifications.

When the default CMS$EVENT_ACTION routine encounters errors, it signals error conditions. If the severity code of the condition is an informational or a warning status code, CMS handles it without interrupting the execution of the CMS$EVENT_ACTION routine. On completion, the CMS $EVENT_ACTION routine returns a completion status code; this status code is signaled by CMS if it does not indicate success.

A user-provided CMS$EVENT_ACTION routine should not issue calls to callable CMS routines other than CMS$GET_STRING or CMS$PUT_STRING. Otherwise, a call issued by CMS $EVENT_ACTION might cause a new CMS event to occur, and possibly trigger an infinite chain of events. A user-provided CMS$EVENT_ACTION routine, however, can call the default CMS $EVENT_ACTION routine as part of its event-handling action.

8.2. Notification of Events

User notification is divided into two parts:

- Detecting and dispatching events
- Notifying users of those events by using the OpenVMS Mail Utility (MAIL)

The default CMS$EVENT_ACTION routine determines the name of the current user(the user whose action caused the event). It then sends one or more notification messages through MAIL. You specify the recipients of the messages in the PARAMETER clause of the ACE defining the event.

Each notification message is formatted as follows:

```
From:   <string specifying the current user>
To:     <string specifying the recipient>
Subj:   CMS notification for library <library name>
<message in a CMS history record format>
```

You use the SET ACL command to associate action ACEs with CMS library objects, and to define events involving these objects. For example:

```
CMS> SET ACL SPEC.RNO/OBJECT=ELEMENT/ACL=(NOTIFY=JOEUSER,ACCESS=MODIFY)
_Remark: send notification if element modified
```

This command specifies that a notification message be sent to user JOEUSER on the local node each time SPEC.RNO is modified.

The text of a notification message is identical to the CMS history record written about the same event. Therefore, CMS notification enables users to receive selected history records through MAIL.

Note that transactions that are not logged in the library history and therefore have no history line (such as ANNOTATE, SHOW, and DIFFERENCES) do not cause an event.

The default CMS$EVENT_ACTION routine makes only one attempt to send each notification message. If the attempt fails, the specified event is not affected in any way. No record of failed MAIL messages is maintained, although the user whose action triggered the event receives any error messages incurred by the default CMS$EVENT_ACTION routine.

To avoid duplicate MAIL messages, you should define action ACEs such that only one event occurs as a result of a single CMS command being executed. Similarly, you should carefully select the recipients of notification messages to avoid unnecessary failed MAIL messages.

8.3. Examples

The following examples show how to use ACLs and notification on objects.

1.
```
$ CMS SET ACL/OBJECT=ELEMENT EXAMPLE.PAS -
_$ /ACL=(NOTIFY=LEADER,ACCESS=MODIFY) "notify project leader"
```

This example specifies that the LEADER account is notified (through MAIL) when a user modifies the element EXAMPLE.PAS.

2.
   ```
   $ CMS SET ACL/OBJECT=GROUP DATA_STRUCTURES -
   _$ /ACL=(ACTION,PARAMETER="MYNODE::JONES",ACCESS= MODIFY)
   _Remark: notify when group DATA_STRUCTURES is modified
   ```

   This example specifies that when a user modifies the group DATA_STRUCTURES, CMS calls the default image SYS$LIBRARY:CMS$EVENT_ACTION.EXE (because you specified ACTION with no file specification) with the parameter MYNODE::JONES. CMS$EVENT_ACTION.EXE then notifies MYNODE::JONES that a user has modified the group DATA_STRUCTURES.

3.
   ```
   $ CMS SET ACL/OBJECT=ELEMENT EXAMPLE.PAS -
   _$ /ACL=(ACTION=CMSLOG_PRO,PARAMETER="NOTIFY.LOG",ACCESS=DELETE)
   _Remark: call event handler when element EXAMPLE.PAS is deleted
   ```

   This example specifies that when the element EXAMPLE.PAS is deleted, CMS calls the user-written event-handler image CMSLOG_PRO, and passes the parameter NOTIFY.LOG. Note that the event-handler routine name is specified without the .EXE file extension, and the NOTIFY.LOG parameter is enclosed in quotation marks because it contains a non-alphanumeric character (a period).

Chapter 9. Library Maintenance

CMS automatically performs maintenance on CMS libraries. You can also perform other types of maintenance to ensure a valid and responsive library.

This chapter presents information on library maintenance that CMS performs, and on the functions that CMS makes available for you to maintain your library and validate its integrity. It also provides some hints on dealing with library problems.

9.1. Command Rollback

If a CMS command is terminated before it has finished executing, CMS automatically initiates a process called *command rollback*. Rollback evaluates the state of the library and then takes appropriate action to return the library to a consistent state so you can enter subsequent CMS commands.

Depending on the point at which the transaction is terminated, rollback takes the following actions:

- If the library contents have not been modified, rollback cancels the command.
- If the transaction is terminated before the update is complete, rollback cancels the command and restores the library to the state it was in before the command was entered. CMS closes and deletes any new files that were created in the library as a result of the command. In addition, rolling back a transaction involves restoring any files in the current, default directory to the state they were in before the command was entered.

  For example, if you run out of disk space during execution of a REPLACE command, CMS might not finish integrating the changes into the element file. In this case, rollback cancels the command, deletes any files that were placed in the library as a result of the command, and restores the library and your current, default directory to the state they were in before the command was entered.
- If the library contents have been completely modified, restoration is not necessary. Rollback recognizes that the command has already been completed and takes no action. For example, a command might be terminated after execution but before control is returned to DCL command level or CMS subsystem level. In this case, the rollback mechanism determines that the command has been executed and rolling back the transaction is not necessary.

The following are examples of errors that can cause rollback:

- You press Ctrl/Y and then enter a command (except STOP); this terminates the transaction. If you enter the DCL command CONTINUE after pressing Ctrl/Y, the CMS command continues executing. This Ctrl/Y CONTINUE sequence works the same as with any DCL command.
- A system-generated error occurs (such as running out of disk space).
- Certain CMS errors occur, causing CMS to enter an error message.
- CMS is terminated by an OpenVMS exception condition.

CMS cannot initiate command rollback under the following circumstances:

- You press Ctrl/Y and then enter the DCL command STOP.

Caution

Never abort the CMS process by pressing Ctrl/Y, then entering the DCL command STOP. CMS cannot perform rollback under this circumstance. To abort CMS, press Ctrl/Y and enter the DCL command EXIT. This enables CMS to roll back the library into a usable state.

- The system is shut down during the execution of a command.
- There is a system failure as a result of a hardware or software error.
- An error occurs during the rollback process itself.

If one of these errors occurs, you must restore the library with the VERIFY/RECOVER command (see *Section 9.2.1, "Using VERIFY/RECOVER"*). CMS informs you if entering VERIFY/RECOVER is necessary.

9.2. Verifying Data in a CMS Library

The VERIFY command checks your CMS library to confirm that the library structure and library files are in a valid form. If you use the VERIFY command under normal conditions, the command executes successfully, and VERIFY returns a success code. A successful VERIFY command indicates that CMS considers the information in your CMS library to be valid.

However, as a result of certain occurrences (for example, a library file is manipulated by a program other than CMS, or the system fails), the data in a CMS library might not be valid. In these cases, when you issue VERIFY, CMS detects the corruption, and VERIFY returns an error code.

The VERIFY command checks the following conditions:

- The library must be set to a valid CMS library directory, or a list of library directories.
- The last CMS command entered on the library must have finished executing (if it did not, CMS attempts automatic recovery before continuing).
- All library control files (01CMS.type) that should be in the library are present and accessible.
- The element, reference copy, class, and group information, reservation information, command list, security information, and internal database structures are in a valid format.
- All element files have been manipulated only by CMS.
- All element files have valid checksums (see *Section 9.2.2, "Using VERIFY/REPAIR"*) indicating that data has not been lost from or added to the files.
- Only element files and other files used by CMS are present in the library (that is, there are no nonelement and no non-CMS files).
- All element files that should be there (one for each element) are present.

If the last transaction is prematurely terminated and is not automatically rolled back, use the VERIFY/ RECOVER command. If any file in the library is not closed by CMS or if the checksum for one or more files is invalid or missing, use the VERIFY/REPAIR command. When you use VERIFY/REPAIR, you must be sure that data has not been lost or added. See *Section 9.2.3, "Correcting Errors"* for more information.

You cannot use the /RECOVER and /REPAIR qualifiers on the same VERIFY command. If conditions exist that call for the execution of both VERIFY/RECOVER and VERIFY/REPAIR, you must enter VERIFY/RECOVER first, then VERIFY/REPAIR.

The following sections describe the /RECOVER and /REPAIR qualifiers in detail.

9.2.1. Using VERIFY/RECOVER

Most CMS commands update several files in the library. If a command is terminated while it is updating the library, the library can be left in a state in which some files have been modified and others have not. Usually, if a command is terminated prematurely, the rollback mechanism cancels and rolls back the transaction (see *Section 9.1, "Command Rollback"*). If CMS cannot roll back the library, you must use the VERIFY/RECOVER command to restore the library to a consistent state.

If you terminate a command at a time when the files in the library might have been left in an inconsistent state, CMS recognizes that the command execution was incomplete. When any user tries to enter a subsequent CMS command to the same library, CMS attempts automatic recovery. If automatic recovery fails, CMS advises the user to enter VERIFY/RECOVER. In this case, users cannot access the CMS library until VERIFY/RECOVER has been executed.

The VERIFY/RECOVER and VERIFY/REPAIR commands use earlier versions of files in the library to restore the library. You should not delete or purge any files from the library, because CMS performs its own cleanup functions.

The VERIFY/RECOVER command cancels only the previous transaction. If the event that causes the premature termination (for example, a system failure) also corrupts data in the library (that is, data stored in files that were present before the event), you must use other means to restore the library. VERIFY/REPAIR corrects some of the unusual occurrences within a CMS library (see *Section 9.2.2, "Using VERIFY/REPAIR"*). CMS might inform you if library repair is necessary after certain commands are issued. In this case, you receive the following message:

```
%CMS-E-USEREPAIR, use VERIFY/REPAIR
```

The VERIFY/RECOVER command affects only the currently set CMS library or libraries, not your default directory. An incomplete transaction might mean that the process of moving files into your directory or deleting files from your directory is incomplete. You must recognize these conditions yourself and, if necessary, remedy them with CMS or DCL commands.

For example, the REPLACE command generally uses a file from your current, default directory to update the element file. If the system fails during a replacement transaction, the process of updating the library file might be incomplete. CMS never deletes any files from your directory until a transaction is complete. In this case, you would need to enter the VERIFY/RECOVER command to cancel the transaction. The file that was being copied would still be in your current, default directory. Another REPLACE command creates a new generation as you originally intended.

If you have set up a restrictive file protection scheme and there is a system failure during a CMS transaction that leaves your library in an inconsistent state, a user with sufficient access to the library and its files should execute the VERIFY/RECOVER command. You can also recover the library if you have BYPASS privilege, or read, write, and execute access to all the library files. For more information, see *Chapter 7, "Security Features"*.

The following commands do not update the library and thus cannot leave the library in an inconsistent state:

ANNOTATE
DIFFERENCES
DIFFERENCES/CLASS
FETCH (no remark)
RETRIEVE ARCHIVE
SET LIBRARY
SET NOLIBRARY
SHOW commands
VERIFY (no qualifiers)

9.2.2. Using VERIFY/REPAIR

You use the VERIFY/REPAIR command when the VERIFY command informs you of one of the following conditions:

- Element data files in the library were not closed by CMS.
- The checksum of elements in the library is invalid.
- Generations in the library have an invalid maximum record size.
- The last recorded transaction time is greater than the current system time.
- The reference copy for an element is missing.
- A reference copy is found for an element with the /NOREFERENCE_COPY qualifier.
- There are duplicate reference copies for an element.
- The reference copy is invalid.

CMS uses information in the file header of a library file to confirm that the file was closed by CMS. If the file was not closed by CMS (for example, if it was opened and closed with a text editor), VERIFY/REPAIR repairs the file header so it can be successfully verified.

For each element, CMS maintains a number known as a **checksum**. A checksum is a count that varies with the number of characters and the value of the characters in a file. Every time CMS writes a file in the library, the checksum is recalculated. The VERIFY command calculates the checksum for every element in the library. If this checksum does not equal the stored value, data has probably been lost from, added to, or changed in the file.

The VERIFY/REPAIR command corrects a bad checksum by recalculating the value based on the current contents of the file and then storing this value. The contents of the file are not altered. If you know that data has been lost from or added to the element, you must correct it manually. See *Section 9.2.3, "Correcting Errors"* for more information.

The VERIFY/REPAIR command adjusts element generations that were created from files with fixed-length records by earlier versions of CMS and have a stored maximum record size of zero. VERIFY/REPAIR examines the element data file, determines what the correct size should be, and stores this value with the generation.

The VERIFY/RECOVER and VERIFY/REPAIR commands use earlier versions of files in the library to restore the library. You should not delete or purge any files from the library, because CMS performs its own cleanup functions.

9.2.3. Correcting Errors

If a program other than CMS has been used to manipulate the files in the CMS library, you might receive the following error message:

```
%CMS-E-VEREDFERR, element DISKX:[PROJECT.CMSLIB]TEST.SDML verified with
 errors
-CMS-E-NOTBYCMS, data file DISKX:[PROJECT.CMSLIB]TEST.SDML;1 not closed by
 CMS
```

If no other errors accompany this message, CMS considers the contents of the file valid, despite manipulation from the outside program. In this case, you can use the VERIFY/REPAIR command to correct any errors (however, you should always investigate your source file to ensure that your file is still valid). Some examples of what can cause

- Entering the DCL command SET PROTECTION or SET FILE/PROTECTION
- Entering the DCL command SET ACL or SET FILE/ACL
- Restoring your CMS library from backup
- Entering the DCL command COPY

Other programs (such as a text editor) can also cause this error.

CMS might also issue the following error message:

```
%CMS-E-BADCRC, bad checksum in element
```

This error is usually accompanied by the CMS-E-NOTBYCMS error. A bad checksum indicates that the contents of the element data file are different from what CMS expects. This usually means that data in the file has been corrupted. Corruption can occur if something has changed the contents of the element data file; this can happen if you alter the element data file, or if a previous version of the element data file was restored from backup. Corruption can also occur if the library directory contains a revision of the CMS database (01CMS.CMS) that does not correspond to the element data file. This typically occurs if the 01CMS.CMS file was restored from backup, but the rest of the library contains more recent versions of element files and was not restored.

You can use the VERIFY/REPAIR command to correct BADCRC errors. If CMS finds more than one version of the element file, it keeps the version containing the correct checksum, and deletes the other files. If no file exists with the correct checksum, VERIFY/REPAIR records the checksum from the most recent file, and deletes any other copies. CMS can then use that value for future checks. CMS does not attempt to alter the contents of the file.

You should use VERIFY/REPAIR to correct BADCRC errors only if you understand the source of these errors and the potential impact of repairing them.

9.2.4. Reference Copies

If a library has a reference copy directory, the VERIFY/REPAIR command performs a comparison between the reference copy and the latest generation on the main line of descent for each element in the library.

If CMS finds a reference copy for an element that does not have the **reference copy** attribute, it prompts you for confirmation, then deletes the reference copy file.

If the **reference copy** attribute is enabled for an element and you enter the VERIFY/REPAIR command, one of the following situations might occur:

- If there is no valid reference copy in the reference copy directory, CMS prompts you for confirmation to delete the remaining copies, then fetches the latest main-line generation (1+) into the reference copy directory.
- If there is more than one reference copy and there is at least one valid copy, CMS keeps the valid copy (or the latest valid generation, if more than one valid copy exists) in the reference copy directory, and deletes the remaining copies.
- If the reference copy does not exist, CMS fetches the latest main-line generation (1+) into the reference copy directory.

9.3. Maintaining Library Efficiency

The following sections describe the features that CMS provides to enable you to maintain the contents of your CMS library.

9.3.1. Deleting History Records

CMS maintains a history file in which all operations that modify the library are recorded. Each operation causes a single record (or one record for each item, when wildcards have been used) to be written into the 01CMS.HIS control file. As libraries get older, history files typically become quite large, taking up disk space and causing SHOW HISTORY performance to degrade. Because very old history is generally no longer useful, you can use the DELETE HISTORY command to reduce the size of the file.

Element generation information (for example, as displayed with the commands SHOW GENERATION, FETCH/HISTORY, and ANNOTATE) is part of each generation and is not stored in the history file; therefore, it is not affected by the deletion of the library history.

9.3.2. Deleting and Archiving Element Generations

When you enter a FETCH, RESERVE, or REPLACE command, CMS searches all the generations of a specified element for the generation you are trying to access. As libraries get older, the number of generations usually increases, and CMS commands that operate on element generations respond more slowly.

You can alleviate this problem by deleting the generations of an element that you no longer need. For example, if you have an element with 100 generations, and generation 5 was released in version 1 of your product, generation 30 was released in version 2, generation 43 was released in version 3, and you are currently developing version 4, you probably do not need to reproduce generations prior to 43, with the exception of those specific generations that went into the released versions. You can use the DELETE GENERATION command to remove the unneeded generations (for more information, see the online help or the VSI DECset for OpenVMS Code Management System Reference Manual).

When you delete a generation, the definition of the generation is permanently removed from the corresponding element in the CMS library. Deleting a generation does not remove changes from subsequent generations that were originally made in the deleted generation. If you delete a generation from the end of a line of descent, all the changes representing that generation are removed from the delta file (see *Section 4.4, "Delta Files"* and *Appendix B, "CMS Library Storage Method"*). If you remove a generation from the middle of a line of descent, changes made in that generation are propagated into the surviving descendant and combined or eliminated from the delta file if possible, because later generations still depend on those changes. You should not rely on generation deletion to reduce the size of a delta file.

If you want to delete an element generation from the CMS library but might still want to access the contents of that generation, you can use the /ARCHIVE qualifier on the DELETE GENERATION command. This qualifier directs CMS to create an archive file containing all the information from the deleted generation.

The archive file is self-contained; you do not need a CMS library to restore the contents of the file. The archive file exists outside of the CMS library and can be backed up onto tape and deleted. You can use the SHOW ARCHIVE command to display the contents of an archive file. Use the RETRIEVE ARCHIVE command to retrieve a copy of any of the generations in an archive file. You cannot restore a generation from the archive directly into the CMS library. To restore the generation, you must retrieve the generation into a file, use the RESERVE command to reserve a generation of the element in the library, then use the REPLACE command to replace the reservation, using the retrieved file as input.

Although the VERIFY command does not operate on archive files, the files store a checksum of the information in the file. The RETRIEVE ARCHIVE command issues a warning message if it finds that the checksum of the data in the file does not match the stored checksum. An incorrect checksum does not prevent you from accessing the data in the file, but it might indicate that the file is corrupt. In this case, you should restore another copy from backup.

9.4. Unusual Occurrences

An unusual occurrence results from the execution of a CMS command that might, at times, have undesirable consequences. An unusual occurrence is always logged in the library history file. The following actions cause CMS to record an unusual occurrence:

- Entering a RESERVE command that creates a concurrent reservation
- Entering a REPLACE command that creates a concurrent replacement
- Entering a REPLACE or UNRESERVE command where BYPASS access was used to manipulate another user's reservation
- Entering the VERIFY/REPAIR command
- Entering the VERIFY/RECOVER command
- Entering the CONVERT LIBRARY command
- Entering the REMARK/UNUSUAL command

The SHOW HISTORY/UNUSUAL command displays the records of transactions that caused unusual occurrences. CMS identifies unusual occurrences in the library history by displaying an asterisk in the first column of the transaction record.

When the RESERVE or REPLACE command produces an unusual occurrence, CMS informs you of the potential unusual occurrence and asks whether you want to proceed. If you answer YES, the command is executed and the transaction is recorded as an unusual occurrence.

The VERIFY/RECOVER and VERIFY/REPAIR commands are logged as unusual occurrences because they are entered when something is wrong with the CMS library structure or its files. If you enter VERIFY/RECOVER or VERIFY/REPAIR on a valid library, or if you enter the VERIFY command without qualifiers, CMS does not log an unusual occurrence.

Chapter 10. Command Syntax

This chapter describes how to enter CMS commands and gives the syntax for command parameters, qualifiers, remarks, and wildcard characters.

10.1. Command Format and Prompting

The general format of a CMS command is as follows:

```
command [keyword] [parameter] [/qualifier...] [remark]
```

A CMS command consists of the name of the command, and a keyword if it is required by the syntax of the command. For example, the RESERVE command consists of only the command name. The SHOW command requires a keyword, for example, HISTORY. In general, you must use one or more spaces or tabs to separate items in a command string. Spaces or tabs preceding a qualifier are optional.

The formats of parameters, remarks, and qualifiers are described in Sections *Section 10.2, "Command Parameters"*, *Section 10.2.3, "Remarks"*, and *Section 10.4, "Command Qualifiers"*, respectively.

A CMS command string can consist of 1024 characters if you use hyphen continuation characters (-). The command can contain any printing characters, spaces, and tabs.

CMS compresses multiple spaces and tabs to a single space (except in quoted strings). You can enter CMS commands in either lowercase or uppercase characters. CMS changes lowercase characters to uppercase (except in quoted strings). As a result, all commands recorded in the library history are in uppercase characters.

If you enter a command that requires a parameter and you do not specify one, CMS prompts you for one. Note, however, that if you use CMS in batch mode or in a command procedure, CMS does not prompt for missing items.

Some commands might require confirmation after you enter the command. In these cases, you are prompted for a YES or NO answer. In some cases, you can also supply one of the following responses:

Positive Response	Negative Response
1	0
TRUE	FALSE
ALL	QUIT or Ctrl/Z

Typing ALL indicates that CMS should perform the action (or actions) specified by the command without any confirmation (for example, after the INSERT GENERATION command). Typing QUIT or pressing Ctrl/Z indicates that CMS should not perform any actions specified by the command.

If you press Return, CMS uses the default, indicated in brackets ([]). Note that CMS checks only the first character of each confirmation response. Thus, typing YAHOO is equivalent to typing YES or Y. If you type any other characters, CMS continues to prompt you until you type an acceptable response.

To halt the execution of a CMS command, press Ctrl/C. Ctrl/C indicates that CMS should terminate the processing of that command. For more information on using Ctrl/C, see *Chapter 9, "Library Maintenance"*.

10.2. Command Parameters

This section describes the parameters that can be used with CMS commands:

- Directory specifications
- Remarks
- Element names
- Element expressions
- Element generations
- Element generation expressions
- Group names
- Group expressions
- Class names
- Class expressions

In addition, you can use wildcard expressions as parameters to certain CMS commands. Wildcard expressions are described in *Section 10.5, "Wildcard Expressions"*.

10.2.1. Directory Specifications

You use a directory specification to refer to a directory that contains (or will contain) a CMS library or reference copy directory. A directory specification is used as a parameter to the CREATE LIBRARY, SET LIBRARY, and SET NOLIBRARY commands, and as a qualifier value to the COPY ELEMENT command. In addition, it is a parameter to the DCL command CREATE/DIRECTORY, which is used to create a directory that will contain a CMS library (see *Chapter 3, "Libraries"*) or reference copy directory.

The format of a directory specification is as follows:

```
disk:[directory]
```

disk

Specifies one or more disks where the directory that contains your CMS library is located. If you omit the disk name, your current default disk is assumed.

directory

Specifies a directory that contains your CMS library. Directory names must be enclosed in square brackets ([]). Wildcards are not allowed.

For more information on how to specify disk and directory names, see the *VSI OpenVMS User's Manual*.

10.2.2. Example

```
$ CMS SET LIBRARY [SWIFT.CMSLIB]
```

This example specifies the subdirectory CMSLIB under the top-level directory [SWIFT] on the current default disk.

10.2.3. Remarks

A *remark* is a character string that you supply to describe a transaction. All CMS commands that modify the library or its contents allow you to enter a remark, which is recorded in the library history as part of the transaction record. Remarks are useful in tracking modifications to a library element. For example, in the remark given on the REPLACE command, you could indicate what changes were made to the element for which you are creating a new generation. For example:

```
CMS> REPLACE DATAFIG3.SDML "updated figure to show new merge routine"
```

For the purpose of command-line interpretation, remarks are defined as parameters; thus, you can enter qualifiers after the remark. However, the remark must be the last parameter entered on the command line. Because remarks are defined as parameters, CMS attempts to translate the remark if other parameters are missing or incorrectly placed. If, for example, you omit an element name from the syntax of a command, but you enter a remark, CMS assumes that the remark is intended as the name of an element.

Quotation marks ("") are required to enclose the remark if you enter it on the same line as the command and the remark contains any spaces. For example, a one-word remark entered on the command line does not require enclosing quotation marks. The text can consist of any printing characters, spaces, and tabs. If you press Ctrl/Z as part of a remark, it terminates the command input at that point, and CMS executes the command. If you press Ctrl/C as part of a remark, CMS cancels the command. To insert a quotation mark (") within a remark, type it twice (""). If a remark consists only of two consecutive quotation marks (""), the remark text is null.

If you omit a remark on the command line of a command that requires a remark, CMS prompts you for the text of the remark on the next line. For example:

```
CMS> REPLACE DATAFIG3.SDML
_Remark: updated figure to show new merge routine
```

Type the text of the remark immediately following the prompt. In this case, you need not enter quotation marks unless you want them to be included in the text of the remark. If you press Return in response to the prompt, you are not prompted again, and the remark text is entered as null.

When you start the remark on the same line as the CMS command, the total length of the remark (including quotation marks), added to the character count for the rest of the command, cannot exceed 256 characters. When you enter the remark in response to the prompt, the length of the remark cannot exceed 254 characters.

You cannot use the hyphen continuation character (-) to continue a remark. If you type a hyphen within a remark and then press Return, the hyphen becomes the last character in the logged remark. The closing quotation marks are assumed. To continue a remark, type the remark until the text wraps to the next line.

Examples

1.
   ```
   CMS> REPLACE SYNTAX.PAS "RECORD declaration implemented"
   ```

   Note that a blank must precede the first quotation mark in a remark. The remark, including the quotation marks, is recorded as part of the record of the REPLACE transaction in the project history.

2.
   ```
   CMS> FETCH SEMANTICS.PAS
   ```

```
_Remark: Get copy for code review
```

If you press Return before you enter a remark, CMS prompts for the remark. The remark is recorded in the project history. It looks the same as if the remark had been entered on the same line as the rest of the command (CMS encloses the remark in quotation marks).

3.
```
CMS> FETCH LEXICAL.PAS "check alternate two-character graphic
implementation for demo version of front end"
```

You cannot use the DCL continuation character (-) to continue the remark; you must continue typing until the text wraps to the next line.

10.2.4. Element Names

You name an element by specifying it as the parameter to the CREATE ELEMENT command.

The format of an element name is as follows:

```
filename.type
```

filename

Specifies the file-name component of an OpenVMS file specification. The filename can be 0 to 39 characters, and must begin with an alphanumeric character. For a list of the characters that you can use in a file name, see the *VSI OpenVMS User's Manual*. Note that systems running versions of OpenVMS that support extended file names no longer have these and other filename restrictions on ODS-5 volumes. See your system manager for details on the /EXTENDED_FILENAMES qualifier if this applies to your environment.

type

Specifies the file-type component of an OpenVMS file specification. The file type can be 0 to 39 characters. For a list of the characters you can use in a file type, see the *VSI OpenVMS User's Manual*.

Separate the file name and the file type with a period (.). An element name must contain a single period even if the file type or file name is null. Spaces and tabs are not legal element name characters. Note that systems running versions of OpenVMS that support extended file names no longer have these and other filename restrictions on ODS-5 volumes. See your system manager for details on the / EXTENDED_FILENAMES qualifier if this applies to your environment.

Note

Within a library, all element names must be unique. The file-name component cannot be 00CMS because that name is reserved for CMS.

The following are examples of valid element names on most systems:

```
TEST.BAS
     SAMPLE.SDML
     ARGCHK.COM
     MOD5.S
```

The following is an example of a valid element name for an OpenVMS systems that supports extended file names:

```
SAMPLE^ ORIGIN.TXT
```

Any file listing requests for this library would display the previous file name as SAMPLE^_ORIGIN.TXT (RMS automatically replaces the space with an underscore). Other characters must also be preceded by the circumflex character, but the space is the only character replaced like this.

10.2.5. Element Expressions

An element expression lets you name multiple instances of an element in a single parameter field.

An element expression is composed of one or more of the following:

- An element name
- A group expression
- A wildcard expression (a wildcard character, or a wildcard character used in combination with a name or partial name)
- A list of the preceding items, with the items separated by commas

If you specify an element name, CMS manipulates a single element. If you specify more than one element name separated by commas or if you specify a group, a wildcard expression, or a combination of these, CMS operates on one or more elements. For example:

```
CMS> SET LIBRARY [JONES.CMSLIB]
CMS> CREATE ELEMENT ELE.SDML,*.LIS,DATASAM.PAS "element list"
```

This command sets the current CMS library to [JONES.CMSLIB] and creates the element ELE.SDML, all elements with a file type of .LIS, and the element DATASAM.PAS. These elements are created from files in your default disk and directory.

You must include a period (.) in the element expression to select one or more elements from the complete list of elements in the library. If you do not include a period, CMS interprets the parameter as a group name and selects elements based on the list of groups established in the library.

10.2.6. Element Generations and Expressions

An element generation is a specific version of an element. Each time you reserve and replace a version of an element in the library, CMS creates a new generation of that element. The first generation of an element is generation 1. Each element generation is assigned a unique generation number; by default, subsequent generations are numbered sequentially by adding 1 to the predecessor generation number.

You can create a variant generation number from an existing generation number by appending a variant letter to the existing generation number and starting a new level number sequence beginning at 1. For example, the generation 7A1 could be a variant generation of generation 7.

The syntax of a generation number is as follows:

```
level-number [variant-letter level-number]...
```

In this syntax, the level-number is a positive integer, and leading zeros are not allowed. The variant-letter is a single alphabetic character (a through z, A through Z).

An element generation expression enables you to specify a particular generation of an element. You can specify a generation indirectly by using a class name, the plus operator, the semicolon, or relative generation offsets. These methods can be combined or used separately.

The format of a generation expression is as follows:

```
{generation-number [+]}
{class-name [+]       }[ [;] relative-generation-offset ]
```

generation-number

Specifies a unique element generation.

class-name

Specifies a CMS class name according to the syntax rules in *Section 10.2.9, "Class Names"*. If a class name value is given, the generation specification refers to the generation in the specified class.

+

Indicates the plus operator. CMS locates the latest generation on the same line of descent as the generation specified by the generation number or class name.

;

Required to separate the relative generation offset from the generation specification. The semicolon is not allowed in cases where a generation number or class name has been omitted and CMS supplies a default value.

relative-generation-offset

Specifies an integer that directs CMS to locate an ancestor or direct descendant of the specified generation. If the relative generation number is negative, CMS locates an ancestor generation. If the relative generation number is positive, CMS locates a direct descendant. The absolute value of the relative generation number indicates how many steps should be taken to the next existing ancestor or descendant generation. A relative generation offset of zero has no effect.

If generations have been deleted, CMS selects the third existing generation prior to the generation you specified. For example, assume the current generation of SAMPLE.PAS in class VERSION1 is generation 7, and generations 5 and 6 have been deleted on the main line of descent for SAMPLE.PAS (thus, the line of descent appears as 1, 2, 3, 4, 7).

Examples

Assume the element SEMANTICS.PAS has six generations on the main line of descent. In addition, a variant line consists of generations 3C1 and 3C2. Generation 5 belongs to the class VERSION1. The following examples show valid forms of the /GENERATION qualifier for the element SEMANTICS.PAS.

1. **`SEMANTICS.PAS/GENERATION=4`**

   This reference selects generation 4 of SEMANTICS.PAS.

2. **`SEMANTICS.PAS/GENERATION=3C1+`**

   This reference selects the latest generation (generation 3C2) on variant line C that extends from generation 3 on the main line of descent. You can use this form if you know a variant line exists and want the most recent generation, but do not know how many generations are on that line.

3. **`SEMANTICS.PAS/GENERATION=VERSION1`**

This reference selects the generation of SEMANTICS.PAS (generation 5) that belongs to the class VERSION1.

4. **`SEMANTICS.PAS/GENERATION=VERSION1;-3`**

   This reference uses a relative generation offset of –3 to select the third generation of SEMANTICS.PAS before the generation that is in class VERSION1. In this example, CMS locates generation 2 of SAMPLE.PAS.

10.2.7. Group Names

You name a group by specifying it as the parameter to the CREATE GROUP command. A group name can be up to 39 characters long, and can contain any of the following characters:

- Letters and digits (a through z, A through Z, and 0 through 9)
- Dollar signs ($)
- Hyphens (-)
- Underscores (_)

A group name must begin with an alphabetic character. Group names cannot contain a period (.) because CMS interprets a group name containing a period as an element name. You cannot use the same name for both a group and a class in the same library. The following are examples of valid group names:

```
GRAPHICS
DATA_IN
DATA$OUT
CREATE-MODULES
```

10.2.8. Group Expressions

A group expression lets you name one or more multiple instances of a group in a single parameter field. A group expression is composed of one or more of the following:

- A group name
- A wildcard expression (a wildcard character, or a wildcard character used in combination with a name or partial name)
- A list of the preceding items, with the items separated by commas

If you specify a group name, CMS operates on a single group. If you specify more than one group name separated by commas or a wildcard expression, CMS operates on one or more groups. The following are examples of valid group expressions:

```
GROUPA
*88
MAIN$MODULES
PHASE_*_DOCS
```

10.2.9. Class Names

You name a class by specifying it as the parameter to the CREATE CLASS command. A class name can be up to 39 characters long, and can contain any of the following characters:

- Letters and digits (a through z, A through Z, and 0 through 9)
- Periods (.)
- Underscores (_)
- Dollar signs ($)
- Hyphens (-)

A class name must begin with an alphabetic character. You cannot use the same name for both a class and a group in the same library. The following are examples of valid class names:

```
BASE_LEVEL3
DEMO.1
VERSION$A
FIELD-TEST
```

10.2.10. Class Expressions

A class expression lets you name multiple classes in a single parameter field. A class expression is composed of one or more of the following:

- A class name
- A wildcard expression (a wildcard character used in combination with a name or partial name)
- A list of the preceding items, with the items separated by commas

If you specify a class name, CMS operates on a single class. If you specify more than one class name separated by commas or a wildcard expression, CMS operates on one or more classes. The following are examples of valid class expressions:

```
VERSION1
BASELINE*
FIELD_TEST
DEMO.%
```

10.3. Comma Lists

Where a comma list is valid, you can specify more than one value for a parameter, separated by commas, on the command line. For example:

```
CMS> DELETE GROUP USER_VIEW,USER_INTFACE,TESTGRP
_Remark: groups no longer necessary-superseded by field test
%CMS-I-DELETED, group DISKX:[PROJECT.CMSLIB]USER_VIEW deleted
%CMS-I-DELETED, group DISKX:[PROJECT.CMSLIB]USER_INTFACE deleted
%CMS-I-DELETED, group DISKX:[PROJECT.CMSLIB]TESTGRP deleted
%CMS-S-DELETIONS, 3 deletions completed
```

This command deletes the three groups USER_VIEW, USER_INTFACE, and TESTGRP. The same remark is logged in the history for each of these groups.

To cancel a comma list transaction before it has completed, press Ctrl/C. If you press Ctrl/C during a transaction using a comma list, CMS finishes the immediate transaction, but does not continue. For example, if you are replacing several elements and you press Ctrl/C during the replacement of the first element, CMS finishes that replacement transaction but does not continue with the others.

When you enter a command using a comma list from DCL command level and then press Ctrl/C during execution of the command, CMS returns control to DCL. If you enter the command from the CMS subsystem prompt level, control is returned to CMS.

10.4. Command Qualifiers

Command qualifiers always start with a slash character (/) and might require a value. A command qualifier, if used, must follow the command (and the keyword, if any). Qualifiers can appear before or after any parameters specified on the command line, except when you use the /GENERATION qualifier with the DIFFERENCES command (see the description of the DIFFERENCES command in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual). You can enter qualifiers after remarks. A command qualifier has the same meaning whether it follows the command name or a command parameter.

For example, the following two commands are equivalent:

```
$ CMS CREATE ELEMENT/KEEP CODEGEN.PAS ""
$ CMS CREATE ELEMENT CODEGEN.PAS/KEEP ""
```

The /KEEP qualifier specifies that the file CODEGEN.PAS is not to be deleted from the user's directory.

Many qualifiers on CMS commands have both a positive and a negative form. For example, /APPEND and /NOAPPEND are the positive and negative forms of the same qualifier.

If you specify the same qualifier more than once on a command or specify both the positive and negative form of the same qualifier, CMS uses only the last specification. For example:

```
$ CMS FETCH INIT.FOR/OUTPUT=TEST.FOR/OUTPUT=INITEST.FOR
```

If you enter this command, CMS uses the second output file specification (INITEST.FOR).

10.4.1. Qualifier Values

Various CMS command qualifiers require quoted strings, file specifications, directory specifications, numeric values, alphabetic values, times, or generation expressions as qualifier values.

You must separate a qualifier and its value with either an equal sign (=) or a colon (:). Zero, one, or more spaces and tabs can appear between the qualifier and the separator, and between the separator and the value. For example, the following two specifications are equivalent:

```
/OUTPUT = TESTFE.COM
/OUTPUT: TESTFE.COM
```

The following sections describe file specifications and the format for entering dates.

10.4.1.1. File Specifications

Many CMS commands allow you to specify input or output files. These commands accept full OpenVMS file specifications as qualifier values. If you do not enter a full file specification, CMS uses the current directory, device, or node. For a complete description of a file specification, see the *VSI OpenVMS User's Manual*. Note that systems running versions of OpenVMS that support extended file names no longer have these and other filename restrictions on ODS-5 volumes. See your system manager for details on the /EXTENDED_FILENAMES qualifier if this applies to your environment.

10.4.1.2. File Structures

CMS only handles valid RMS files. If the input file is generated from anon-valid (non-VSI) RMS file, you must use the OpenVMS ANALYZE/RMS facility to analyze the legality of the file before you store it in the CMS library.

10.4.1.3. Time Values

Several commands allow you to specify time values with the /BEFORE and /SINCE qualifiers. Each of these qualifiers accepts an absolute, delta, or combination time value. You can also specify one of the following keywords: YESTERDAY, TODAY, or TOMORROW.

An absolute time is a specific date or time of day, or both. A delta time value is the difference between the current time and a future time. A combination time consists of an absolute time value plus or minus a delta time value. For detailed information about time values, see the *VSI OpenVMS User's Manual*.

10.4.2. Qualifier Defaults

Each command description in the online help or the VSI DECset for OpenVMS Code Management System Reference Manual contains a list of qualifiers and qualifier defaults. The default indicates the action taken when you omit the qualifier.

Qualifiers with simple positive and negative forms (those that do not take qualifier values) are listed in the command format sections with their defaults. For example:

```
/[NO]APPEND
/NOAPPEND
```

On the left, the qualifier is listed with brackets ([]) around the optional part of the qualifier (NO). On the right, the default is listed.

Some qualifiers have a positive form that allows a qualifier value, and a negative form that does not allow the value. These qualifiers are shown with their defaults. For example:

```
/MERGE=generation-exp      /NOMERGE
/NOMERGE
```

If you use the positive form, the generation expression is required. If you use the negative form, which is the default, the generation expression is not allowed.

The defaults (if any) for qualifier values are explained in the qualifier descriptions and are also indicated by the letter D next to the qualifier name.

10.5. Wildcard Expressions

You can use DCL wildcard expressions in the parameters for many CMS commands. The wildcard characters are the percent sign (%) for single-character substitution and the asterisk (*) for partial- or full-field substitution. By using these wildcards, you can direct CMS to operate on more than one element, group, or class at a time. In addition, you can use wildcards in input and output file specifications, and the directory-searching wildcards (the ellipsis (…) and the minus sign (–)) in input file specifications.

For elements and generations, wildcards can apply to either the file-name field or the file-type field, according to the position they occupy.

The following sections describe general rules for using wildcards.

10.5.1. Single-Character Wildcards

The percent sign (%) is the single-character wildcard indicator. When you use the percent sign in a command parameter, CMS selects elements, groups, or classes by substituting any single, allowable character for the percent sign.

For example, the wildcard expression DATA%.FOR might result in the following list of elements:

```
DATA1.FOR
DATA2.FOR
```

10.5.2. Partial-Field and Full-Field Wildcards

The asterisk (*) is the partial- and full-field wildcard indicator. When you use an asterisk in a command parameter, CMS selects objects whose names contain the character patterns given in the wildcard expression. CMS replaces the asterisk with any number of allowable characters (within the range of zero to the maximum size of the field).

For example, the element expression DATA*.FOR might result in the following list of elements:

```
DATA.FOR
DATA1.FOR
DATA2.FOR
DATA_IN.FOR
DATA_OUT.FOR
```

10.5.3. Canceling Wildcard Transactions

To cancel a wildcard transaction before it has completed, press Ctrl/C. If you press Ctrl/C during a wildcard transaction that updates the library, CMS finishes the immediate transaction, but does not continue. For example, if you are replacing several elements and you press Ctrl/C during there placement of the first element, CMS finishes that replacement transaction but does not continue with the others.

When you enter a wildcard command from DCL command level and then press Ctrl/C during execution of the command, CMS returns control to DCL. If you enter the command from the CMS subsystem prompt level, control is returned to CMS.

10.6. Command Abbreviations

You can abbreviate command, keyword, and qualifier names by eliminating characters from the end of the specified command, keyword, or name. You cannot truncate the string “CMS” when entering a CMS command at DCL level. All commands and qualifiers are unique when truncated to their first four characters. You can truncate these names to fewer than four characters as long as the result is unique.

For example, VERIFY is the only CMS command that begins with the character V. Therefore, the VERIFY command can be truncated to CMS V at DCL level and V at CMS subsystem level.

You do not count the slash character (/) or the prefix NO on negative qualifiers when you count characters to determine the shortest allowable form of a qualifier. However, you must count the underscore (_) character.

Appendix A. Summary of CMS Interface Functional Mappings

This table displays how each of the CMS interfaces are functionally mapped into each other.

DCL Command	Callable Routine	DECwindows Motif Menu Item
ACCEPT GENERATION	CMS$REVIEW_GENERATION	Maintenance.Review.Accept
ANNOTATE	CMS$ANNOTATE	Data.Annotate
CANCEL REVIEW	CMS$REVIEW_GENERATION	Maintenance.Review.Cancel
CONVERT LIBRARY	N/A	N/A
COPY CLASS	CMS$COPY_CLASS	Maintenance.Copy.Class
COPY ELEMENT	CMS$COPY_ELEMENT	Maintenance.Copy.Element
COPY GROUP	CMS$COPY_GROUP	Maintenance.Copy.Group
CREATE CLASS	CMS$CREATE_CLASS	Maintenance.Create.Class
CREATE ELEMENT	CMS$CREATE_ELEMENT	Data.Create.Element
CREATE GROUP	CMS$CREATE_GROUP	Maintenance.Create.Group
CREATE LIBRARY	CMS$CREATE_LIBRARY	Library.Create
DELETE CLASS	CMS$DELETE_CLASS	Maintenance.Delete.Class
DELETE ELEMENT	CMS$DELETE_ELEMENT	Maintenance.Delete.Element
DELETE GENERATION	CMS$DELETE_GENERATION	Maintenance.Delete.Generation
DELETE GROUP	CMS$DELETE_GROUP	Maintenance.Delete.Group
DELETE HISTORY	CMS$DELETE_HISTORY	Maintenance.Delete.History
DIFFERENCES	CMS$DIFFERENCES	Data.Differences.File
DIFFERENCES/CLASS	CMS$DIFFERENCES_CLASS	Data.Differences.Class
FETCH	CMS$FETCH	Data.Fetch
INSERT ELEMENT	CMS$INSERT_ELEMENT	Maintenance.Insert.Element
INSERT GENERATION	CMS$INSERT_GENERATION	Maintenance.Insert.Generation
INSERT GROUP	CMS$INSERT_GROUP	Maintenance.Insert.Group
MARK GENERATION	CMS$REVIEW_GENERATION	Maintenance.Review.Mark
MODIFY CLASS	CMS$MODIFY_CLASS	Maintenance.Modify.Class
MODIFY ELEMENT	CMS$MODIFY_ELEMENT	Maintenance.Modify.Element
MODIFY GENERATION	CMS$MODIFY_GENERATION	Maintenance.Modify.Generation
MODIFY GROUP	CMS$MODIFY_GROUP	Maintenance.Modify.Group
MODIFY LIBRARY	CMS$MODIFY_LIBRARY	Maintenance.Modify.Library
MODIFY RESERVATION	CMS $MODIFY_RESERVATION	Maintenance.Modify.Reservation
REJECT GENERATION	CMS$REVIEW_GENERATION	Maintenance.Review.Reject
REMARK	CMS$REMARK	Maintenance.Remark
REMOVE ELEMENT	CMS$REMOVE_ELEMENT	Maintenance.Remove.Element

DCL Command	Callable Routine	DECwindows Motif Menu Item
REMOVE GENERATION	CMS $REMOVE_GENERATION	Maintenance.Remove.Generation
REMOVE GROUP	CMS$REMOVE_GROUP	Maintenance.Remove.Group
REPLACE	CMS$REPLACE	Data.Replace
RESERVE	CMS$FETCH	Data.Reserve
RETRIEVE ARCHIVE	CMS$RETRIEVE_ARCHIVE	N/A
REVIEW GENERATION	CMS$REVIEW_GENERATION	Maintenance.Review.Comment
SET ACL	CMS$SET_ACL	Maintenance.Set.ACL
SET LIBRARY	CMS$SET_LIBRARY	Library.Open
SET NOLIBRARY	CMS$SET_NOLIBRARY	Library.Close
SHOW ACL	CMS$SHOW_ACL	View.Expand.ACLs
SHOW ARCHIVE	CMS$SHOW_ARCHIVE	N/A
SHOW CLASS	CMS$SHOW_CLASS	View.Class
SHOW ELEMENT	CMS$SHOW_ELEMENT	View.Element
SHOW GENERATION	CMS$SHOW_GENERATION	View.Expand.Children
SHOW GROUP	CMS$SHOW_GROUP	View.Group
SHOW HISTORY	CMS$SHOW_HISTORY	View.History
SHOW LIBRARY	CMS$SHOW_LIBRARY	N/A
SHOW RESERVATIONS	CMS$SHOW_RESERVATIONS	View.Reservation
SHOW REVIEWS_PENDING	CMS $SHOW_REVIEWS_PENDING	View.Review
SHOW VERSION	CMS$SHOW_VERSION	Help.About
UNRESERVE	CMS$UNRESERVE	Data.Unreserve
VERIFY	CMS$VERIFY	Library.Verify

Appendix B. CMS Library Storage Method

This appendix contains information that might be useful to you as you build and use your CMS library. In general, project planning has the greatest impact on how you can best use CMS. Each project has its own characteristics that determine how you should organize your library.

CMS stores the entire text of the first generation of an element. This file is called a delta file. Each time you replace an element, CMS determines what has been changed in the element files, and, to save storage space, stores only the new and changed lines of successive generations. (See *Chapter 4, "Elements and Generations"* for more information.) To estimate the required storage for the library, you should allow three times the amount of disk space that you would normally allow for one copy of all project files.

The following example shows a file in a CMS library:

```
    .
    .
    .
                APPLES
                BANANAS
                CHERRIES
*2D
                POOCHES
*2E
*2I
*3D
                PAUNCHES
*3E
                PEACHES
*2E
                ELDERBERRIES
    .
    .
    .
```

This example shows that each data item is numbered according to the element generations in which the item appears. The first generation elements are APPLES, BANANAS, CHERRIES, POOCHES, and ELDERBERRIES. In the second generation, POOCHES is deleted (*2D-*2E) but PAUNCHES and PEACHES are inserted (*2I-*2E). PAUNCHES is then deleted in the third generation (*3D-*3E).

CMS can provide a complete copy of any of the three generations whenever necessary. (It can also produce an annotated copy showing all changes in each generation and identify the users who made those changes.) However, the data requires only seven lines of storage space in the library. Conventional storage methods require 15 lines, five lines for each of the three copies of the data.

You do not need to save backup copies of CMS library files in your account. Normal system-backup procedures should be followed for a CMS library. CMS itself maintains a certain amount of backup information so it can recover from an incomplete transaction after a system failure.

Elements are stored most efficiently when modifications leave the majority of the file lines unchanged. CMS stores only one copy of an element; this copy includes all lines from the first generation plus all modifications to successive generations. Thus, the number of differences (relative to the number of original lines) affects system efficiency.

For example, the modifications to successive generations of a FORTRAN source program might typically change 15 to 20 percent of the lines during the development of that program. Because the bulk of the program does not change, this kind of element is ideal for a CMS library. However, because the same program's listing file would change greatly with each modification (due to the compiler's effect on line numbers, addresses, and so forth), each generation of the stored listing element would contain almost as many differences as original lines.

Appendix C. System Management Considerations

This appendix contains information about running CMS on an OpenVMS system.

C.1. Library Backup

You should use normal system-backup procedures to back up your CMS libraries. Although CMS is designed to recover from certain kinds of failures, there are some events that leave a library in a state that is impossible to recover or repair. If this is the case, you should have a recent backup copy of the library that you can use as a replacement.

If there is an error in any of the library data structures contained in the control file, you should use a backup copy of the entire library. In some cases, CMS might display an error message indicating that an element file has been corrupted. In this case, you can substitute a recent backup copy of the element file.

If you substitute a backup version of an element file for a corrupted element file, you should note the following effect: if the library control file indicates that additional generations were created after the backup element generation, the contents of those additional generations are the same as those of the backup element generation. Thus, if you have an element that has 10 generations, and you substitute a file that corresponds to generation 9, the element is still valid; however, generations 9 and 10 have the same contents.

When you use a backup copy of an element data file, follow these rules:

- Use full wildcards (file name, extension, and version number) when you copy the backup file so you maintain all the file header information.
- Remove the corrupted file from the library to avoid time conflicts between the earlier backup copy and the later, corrupted file.
- After restoring the element data file, use VERIFY/REPAIR to fix the checksum.

See *Chapter 9, "Library Maintenance"* for more information on library maintenance.

C.2. System Time Errors

Under certain circumstances, CMS might display the following messages:

```
%CMS-E-NOREF, error referencing 'directory'
-CMS-E-SYSTIMERR, system time has been set incorrectly
-CMS-E-SYSTIMDIF, last transaction executed at 'time'
```

CMS displays these messages if the current system time is before the time of the last transaction. One of three situations might have occurred:

- The system time was inadvertently set back.
- At some point, the system time was set ahead and then later corrected (any transactions that were executed during this time period might appear to have occurred in the future).
- There is a difference between the system time of two nodes on a cluster.

When CMS indicates a system time error, you should first check the system times and correct any error. If your system is part of a cluster, check all the nodes on the cluster. After you correct the system time error, compare the current system time with the time of the last transaction (reported in the SYSTIMDIF message). If the difference between the time of the last transaction and the current system time is small, wait until the current system time is later than the last transaction and execute the command again. If the time difference is large, use the command VERIFY/REPAIR.

C.3. Library Limits

CMS libraries are limited to the maximum number of objects as shown in the following table.

Library Objects	**Maximum Number Allowed per Library**
Elements	65,535
Classes	65,535
Groups	65,535

C.4. Quotas

See the Code Management System for OpenVMS Release Notes for the recommended minimum disk space and system quota settings for CMS.