User's Introduction to
VAX DEC/CMS

Order No. AA-L371B-TE

November 1984

REVISION/UPDATE INFORMATION:	This document supersedes the *User's Introduction to DEC/CMS*. (Order No. AA-L371A-TE).
OPERATING SYSTEM AND VERSION:	VAX/VMS Version 4.0
SOFTWARE VERSION:	VAX DEC/CMS Version 2.0

digital equipment corporation · maynard, massachusetts

First Printing, May 1982
Revised, November 1984

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Printed in U.S.A.

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	PDT
DEC/CMS	EduSystem	RSTS
DEC/MMS	IAS	RSX
DECnet	MASSBUS	TOPS-20
DECsystem-10	MicroPDP-11	UNIBUS
DECSYSTEM-20	Micro/RSX	VAX
DECUS	MicroVMS	VMS
DECwriter	PDP	VT
		digital

ZK2334

HOW TO ORDER ADDITIONAL DOCUMENTATION

In Continental USA and Puerto Rico call 800-258-1710

In New Hampshire, Alaska, and Hawaii call 603-884-6660

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6146 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575)

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
940 Belfast Road
Ottawa, Ontario K1G 4C2
Attn: P&SG Business Manager

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
P&SG Business Manager
c/o Digital's local subsidiary or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

Contents

PREFACE

OBJECTIVE

This manual explains how to use VAX DEC/CMS (Code Management System). It is an introductory level book with examples that illustrate basic techniques.

INTENDED AUDIENCE

This manual is intended for software engineers, technical writers, and managers working on a wide range of software projects.

Version 2 of DEC/CMS runs only on the VAX/VMS operating system Version 4.0 or later.

STRUCTURE OF THIS DOCUMENT

The *User's Introduction to VAX DEC/CMS* is divided into seven chapters, one appendix, and a glossary.

- Chapter 1, CMS Concepts, describes how CMS facilitates the software development cycle and briefly explains how to use CMS.
- Chapter 2, Building a CMS Library, describes how to set up a CMS library and also how to load files into the library.
- Chapter 3, Using a CMS Library, describes the basic commands that you use to retrieve and replace library files.
- Chapter 4, CMS SHOW Commands, explains how you can use CMS commands to get information about the contents and the history of a CMS library.
- Chapter 5, The Evolving CMS Library, describes how to use CMS to track concurrent development.
- Chapter 6, Project Organization with CMS, describes how to use CMS to organize a software project.
- Chapter 7, Additional CMS Commands, describes commands that provide additional functions.

- Appendix A, Notes on Using CMS Libraries, contains general information that is useful in understanding how CMS works.
- The Glossary defines important terms.

ASSOCIATED DOCUMENTS

- The *VAX DEC/CMS Reference Manual* (Order No. AA-L372B-TE) provides detailed information about CMS concepts and commands.
- The *VAX DEC/CMS Callable Interface Manual* (Order No. AA-Z340A-TE) contains information about using the DEC/CMS callable routines.
- *Installing VAX DEC/CMS* (Order No. AA-Z338A-TE) supplies the instructions for installing CMS on a VAX/VMS system.
- The *VAX DEC/CMS Pocket Guide* (Order No. AV-L374A-TE) provides concise information about CMS commands and callable routines.

CONVENTIONS USED IN THIS DOCUMENT

Convention	Meaning
[ ]	Square brackets indicate that the enclosed item is optional.
...	A horizontal ellipsis indicates that the preceding item(s) can be repeated one or more times.
`$ CMS SHOW VERSION`	Interactive examples show system output (prompt characters and output lines) in black type. All lines that you enter are printed in red type.
element	A term that appears in italics is defined in the Glossary.

Unless otherwise noted:

- All numeric values are represented in decimal notation.
- You terminate a command by pressing the RETURN key.

Chapter 1

CMS Concepts

DEC/CMS (Code Management System) is a program library system for software development and maintenance. DEC/CMS (also referred to as CMS) stores source files in a library, keeps track of changes made to these files, and records user access to the files in the libraries. In addition, CMS supplies two ways to organize files within a library; these formal mechanisms for library organization provide a focus for system design.

This chapter describes the basic CMS functions. The commands that you use to build and then use a CMS library are described in subsequent chapters. This chapter also describes a hypothetical project that uses CMS. (Examples throughout this manual refer to this project.)

1.1 Overview of DEC/CMS

CMS operates as an online librarian for a project. While you perform the normal functions of program development, modification, and testing, CMS keeps track of your files. After you place your programs in a CMS library, you can retrieve the programs to modify and test them in your own directory. You then replace the modified programs in the library for safekeeping. Also, you can look up the history of every program in the library, because CMS maintains its own records.

CMS is available at the system command level; commands to CMS are similar to DCL commands (such as EDIT, COPY, and SET). You can also use CMS at the subsystem level; once you invoke the CMS image, you can execute any number of CMS commands while remaining within the CMS subsystem.

CMS does its job without imposing unnecessary restrictions on project members. You can do all program modification, editing, testing, and development in the familiar VAX/VMS environment. You use CMS to perform the following operations:

- Place a program in the library (Chapter 2).
- Copy a program from the library to your current default directory and mark the library copy as reserved by you (Chapter 3).
- Return a modified program back to the library (Chapter 3).
- Retrieve a copy of a program from the library for read-only purposes (Chapter 3).

- Display a list of library programs (Chapter 4).
- List all versions of a library program (Chapter 4).
- Display the library status or history (Chapter 4).
- Control concurrent modifications to the same program (Chapter 5).
- Create specific sets of programs within a library (Chapter 6).
- Compare two ASCII files in your default directory (Chapter 7).
- Compare two versions of a program within a library (Chapter 7).

A CMS library can store an ASCII file that was created by a text editor such as EDT or SOS. However, the files you store in the library should be ASCII files that are unlikely to change completely in each iteration. (See Appendix A for a discussion of library storage.) Usually, text and source files are the project elements that require tracking.

NOTE

Use only CMS commands to access CMS libraries. You are responsible for the types of files in the library; a file must be sequential and cannot have the same characteristics as a binary file. You are also responsible for the method of putting files in the library and the method of manipulating them in the library; files that have been put in the library using non-CMS commands may eventually be deleted by CMS. In general, the integrity of a library is not assured if it has been changed using means other than CMS.

1.1.1 Library Elements

An *element* is the basic structural unit in a CMS library. An element consists of one ASCII file. You create and name an element when you transfer a file from your current default directory to a CMS library.

1.1.2 Making Changes to a Library Element

After you have created an element in a CMS library, you *reserve* the element in order to make changes to it. When you reserve an element, CMS places a copy of the element file in your current default directory. You can then edit, compile, test, and debug as usual. When you are ready, you *replace* the element in the library, and CMS stores the changes that have been made.

When you create an element and place it in a CMS library for the first time, CMS creates generation 1 of that element. An *element generation* represents a phase in the development of that element. Each time you reserve and replace an element in the library, CMS creates a new generation. You can recreate any phase in the development of an element by retrieving the appropriate generation.

CMS numbers the new generations in succession. A CMS generation number is not to be confused with the version number that VAX/VMS assigns to directory files; file version numbers have no significance to CMS. CMS ignores the number of work sessions done outside the library, and assigns a new generation number only when the element is returned to the library.

1.1.3 Reserving and Replacing an Element

Example 1-1 shows how you can use CMS in the course of program development.

```
$ CMS RESERVE TEST1.FOR
_Remark: check for non-printing chars
%CMS-S-RESERVED, generation 1 of element TEST1.FOR reserved
$ EDIT TEST1.FOR
    .
    .
    .
$ FORTRAN TEST1/DEBUG
$ LINK/DEBUG TEST1
$ RUN TEST1
    .
    .
    .
$ CMS REPLACE TEST1.FOR
_Remark:
%CMS-S-GENCREATED, generation 2 of element TEST1.FOR created
$ CMS SHOW HISTORY TEST1.FOR

History of DEC/CMS Library _DRA3:[PROGRAMS.LIBRARY]
    .
    .
    .
 28-MAY-1984 15:04:37 SHAW CREATE ELEMENT TEST1.FOR "parameter list test"
 29-MAY-1984 12:24:52 SHAW RESERVE TEST1.FOR(1) "check for non-printing chars"
 30-MAY-1984 18:05:42 SHAW REPLACE TEST1.FOR(2) "check for non-printing chars"
    .
    .
    .
```

Example 1-1: Sample CMS Terminal Session

The first command to CMS (CMS RESERVE TEST1.FOR) directs CMS to retrieve the element TEST1.FOR from the library. CMS responds with the prompt "_Remark:". This allows you to label the transaction with an appropriate remark. After the remark is entered, CMS executes the command and places a copy of the file associated with the element TEST1.FOR in your current default directory.

The series of commands that follow the first library transaction represent commands a programmer might use to modify and test the file TEST1.FOR. The next command to CMS (CMS REPLACE TEST1.FOR) also allows the programmer to enter a remark for the transaction. In this case, no remark is entered. CMS returns the file to the library. The last command to CMS (CMS SHOW HISTORY TEST1.FOR) displays information about the

element TEST1.FOR on the terminal. Since no remark was entered for the replacement transaction, the remark entered for the CMS RESERVE transaction is also logged for the CMS REPLACE transaction.

1.2 Interactive Help

Information about CMS is available at the terminal. The HELP CMS command provides online documentation in summary form for setting up a CMS library and for using specific CMS commands. For example, the following command displays information about the CMS RESERVE command:

```
$ HELP CMS RESERVE
```

You can also get help at the subsystem level:

```
CMS>HELP RESERVE
```

Since VAX/VMS allows abbreviations you need type only enough of a CMS command to distinguish it from any other CMS command.

1.3 Project Planning and Development

CMS provides two mechanisms for organizing library elements–*groups* and *classes*; by using groups and classes, you can plan and use your CMS libraries more effectively. Each mechanism imposes its own kind of structure on the library, yet they can be used in the same library without conflict. The following sections summarize these methods of project organization. See Chapter 6 for more information.

1.3.1 Using Groups for Structural Organization

Individual modules of code or text exist in a CMS library as elements. CMS allows you to combine related elements into a group that you can then manipulate as a unit. For example, you might create a group that contains all modules that process error messages.

1.3.2 Using Classes for Organization by Milestone

The second method of organization is to create classes. You use classes to organize your library by milestones.

One approach is to integrate the modules into working versions or "base levels" that represent progressive stages in the development of the entire system. A different approach might be to establish classes for more localized stages–one class for each stage of coding, testing, and integration to be performed for each generation of the element. The way in which you use classes is dependent on the structure of your library and the methodology that you use for software development.

1.4 A Typical CMS Application

The following story summarizes how the programmers on one project used CMS. Examples throughout this manual refer to the experiences that these programmers had using CMS.

The three-programmer team was developing an analog data sampling program. The team members agreed to use CMS, rather than keeping track of project files manually. During the design phase, the project members planned the structure of the product and determined the basic CMS mechanisms they would use to reflect that structure. The lead programmer, Rick Sanchez, created a library directory for CMS access. As they wrote the design documentation, the project members put the files for these documents in the CMS library.

When the design phase was finished, Rick put some project files from his current default directory into the project library. Then Rick told Glenn Lewis and Sue Kelley the project library directory name, and Glenn and Sue added their own working programs to the project library. Now the project files (SYNCHRON.BAS, SAMPLE.BAS, SPEC.RNO, and several others) resided in the library as elements.

To organize the library elements, the project members created groups to contain selected elements. CMS maintained the list of elements contained in each group. After establishing the groups, project members were able to manipulate each group of elements as a single unit.

Once the project library was set up, it was used as a repository for the project programs. Each developer had access to all of the library elements, and any work done on a program would be recorded and tracked automatically by CMS. When progress on a program branched out to an alternate development path, CMS tracked both the original development path and the alternate path. CMS maintained a complete program history of all modifications to the original set of programs.

Rick and Glenn worked on the same program from time to time. When Glenn requested SAMPLE.BAS while Rick was working on it, CMS notified him that Rick had it reserved. Occasionally Glenn had to ignore the warning and use the same program. When Rick replaced the program, CMS reported that Glenn had worked on the program while he was updating it. CMS stored these concurrent changes to the program. Any one of the team members could use CMS to merge their changes and flag any conflicting modifications (changes that affected the same line of the code).

Although all three programmers used the command file, TIMTST.COM, to test modifications to their own programs, TIMTST.COM itself rarely changed. CMS allowed the project members to get copies of the command file while protecting the command file from inadvertent modification.

When the team was ready to build its first prerelease version of the data sampler, CMS aided in specifying the programs in that version (or "base level"). Rick established a CMS *class* to combine specific versions of these programs together and named the class BASELEVEL1. By this time, some programs had been modified more than 20 times. Rick inserted specific *generations* (program versions) of each project program in that class so there would be no confusion over which generation of a program belonged in the base level.

Throughout the project the programmers found that CMS helped to define project development and direction more clearly. Each project member could review all library transactions and associated remarks. In addition, the programmers could selectively review the transactions related to a specific aspect of project development. They also discovered that CMS functions simplified their work. For example, when Sue was on vacation, Glenn made some changes to an element that Sue usually worked on. Later, Sue was able to get line-by-line information on the element, indicating which of her planned modifications had already been done. Rick found that the CMS history and its remarks could be used as reminders of project accomplishments when he wrote his status reports.

Later Mike Cohen joined the team. His first assignment was to evaluate problem reports from field test sites. He found that the CMS annotated listings of the program modifications helped him locate recently changed code more easily. When he wanted to know the reasons for the changes, Mike referred to the programmer's remarks that CMS recorded when each modification was performed.

Now the original developers have moved on to new projects. Mike, however, has chosen to stay with the data sampling project, and he has a new programmer to help him with project support and enhancements. They still use the project's CMS library, which now contains every line of code written for the data sampler, along with a complete historical record of the project.

Chapter 2

Building a CMS Library

This chapter describes the procedures and CMS commands that you use to set up a CMS library. When you establish a CMS library, you create a directory that is to be used only as a CMS library. You then define the protection that will allow the appropriate people to access the library. Once the directory is established, you can direct CMS to initialize the library and then place project files in the library. If you do not have to create a library or create elements in a library, you can skip this chapter and go to Chapter 3.

You set up the library in four steps:

1. Create a directory to contain the CMS library.
2. Establish the protection scheme for the library.
3. Initialize the directory as a CMS library.
4. Store files in the library as elements.

Once you have created the library, you should use only CMS to access it.

2.1 Creating a Library Directory

Use the DCL command CREATE/DIRECTORY to establish a directory to contain a CMS library. For example, at the beginning of the data sampler project, Rick set up a CMS library subdirectory with the command:

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

This command created the subdirectory [PROJECT.ADSLIB] for use as the project library directory on Rick's default disk.

2.2 Protecting a CMS Library

CMS does not define any protection scheme for a library directory or any of the files contained within the library. Thus, you must explicitly define a UIC-based or ACL-based protection scheme for your libraries.

Rick wanted Glenn and Sue, who also are in his user identification code (UIC) group, to have access to the library files at will. After he created the subdirectory [PROJECT.ADSLIB] for use as the CMS library, he used the following commands to set the required protection:

```
$ SET PROTECTION=(S:RWE,O:RWE,G:RWE) [PROJECT]ADSLIB.DIR
$ SET FILE/ACL=(DEFAULT_PROTECTION,S:RWD,O:RWD,G:RWD) [PROJECT]ADSLIB.DIR
```

The DCL command SET PROTECTION specifies that system, owner, and group members have the standard read, write, and execute access to the [PROJECT.ADSLIB] directory. The DCL command SET FILE/ACL specifies that system, owner, and group members are allowed read, write, and delete access to all new files created in the library directory.

See the *VAX DEC/CMS Reference Manual* for more information about protecting CMS libraries.

2.3 Creating a CMS Library

The CMS CREATE LIBRARY command creates CMS control files in a directory. Once you initialize a directory with CMS data structures, CMS uses that directory to locate and store project files. All commands to the CMS system automatically refer to this directory until the end of the terminal session, or until you specify a different library with the CMS SET LIBRARY command (see Section 3.1).

For example, Rick initialized the subdirectory, ADSLIB, with the following command:

```
$ CMS CREATE LIBRARY [PROJECT.ADSLIB]
_Remark: a/d data sampling library
%CMS-S-CREATED, DEC/CMS library DRA3:[PROJECT.ADSLIB] created
```

Then, with the library created, protected, and initialized, he was ready to use CMS and place project files in the library.

2.4 Placing Project Files in the CMS Library

The CMS CREATE ELEMENT command instructs CMS to move a file from your current default directory into the library. (When CMS places a file in the library, all copies of that file are deleted from your current default directory. You can use the /INPUT qualifier to direct CMS to use a file from a directory other than your current default, in which case CMS deletes the file from the specified area.) An element name is the file name and type of the file specified in the CMS CREATE ELEMENT command. Every element in the CMS library must have a unique name.

On the data sampler project, Rick moved five files into the library. He gave CMS the following CMS CREATE ELEMENT commands:

```
$ CMS CREATE ELEMENT SPEC.RNO
_Remark: ad sampling functional specification
%CMS-S-CREATED, element SPEC.RNO created
$ CMS CREATE ELEMENT ADCONVERT.BAS
_Remark: analog to digital conversion routines
%CMS-S-CREATED, element ADCONVERT.BAS created
$ CMS CREATE ELEMENT SAMPLE.BAS
_Remark: Sampling module
%CMS-S-CREATED, element SAMPLE.BAS created
$ CMS CREATE ELEMENT SYNCHRON.BAS
_Remark: Synchronization routines
%CMS-S-CREATED, element SYNCHRON.BAS created
$ CMS CREATE ELEMENT TIMTST.COM
_Remark: Command procedure for tests
%CMS-S-CREATED, element TIMTST.COM created
```

Figure 2-1 shows the effects of each action in the building procedure. When Sue and Glenn put their files into the library, they used the CMS CREATE ELEMENT command with their own file names.

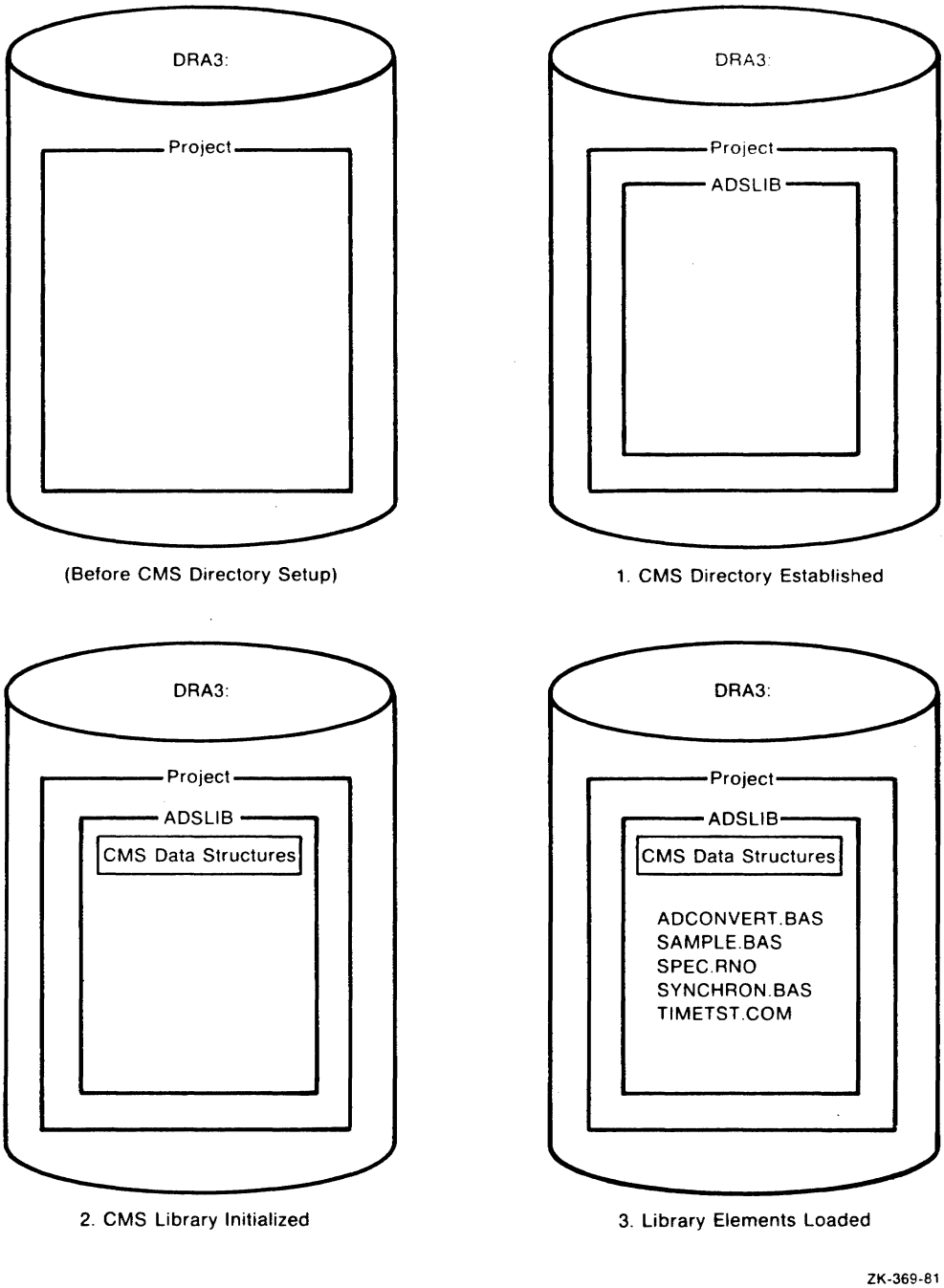


Figure 2-1: Building a CMS Library

When determining which files should be in a library, remember that CMS elements must be ASCII files. The ASCII files must have been created or modified by a text editor such as EDT. If you are in doubt about a file's suitability as a CMS library element, check the file characteristics in your own directory with the following command.

```
$ DIRECTORY /FULL filename.type
```

A file must be sequential and cannot have the characteristics of a binary file. If these conditions are not met, CMS cannot handle the file. See the *VAX DEC/CMS Reference Manual* for more information about file characteristics.

Once you create elements in a library, you can impose a structure on your library by creating groups of elements. A group consists of one or more elements that CMS manipulates as a unit. See Chapter 6 for more information about groups.

Chapter 3

Accessing Files in a CMS Library

This chapter describes the CMS commands that you use to access files in a CMS library. These commands perform the following functions:

- The CMS SET LIBRARY command gives you access to a particular library.
- The CMS RESERVE command copies a generation of an element file to your current default directory. You perform all program modifications and testing in your own directory with an editor, compiler, and utility. CMS records the reservation transaction and marks the element as reserved.
- The CMS UNRESERVE command cancels your reservation and leaves the element copy in your directory. CMS marks the library element as available for other reservations and records the transaction. Since there are no modifications to the library element, CMS does not update the library element.
- The CMS REPLACE command copies the latest version of the element file from your current default directory to the CMS library, creating a new generation of the element. CMS deletes the element file from your directory, allows you to enter a remark to document the changes, records the transaction, and concludes your reservation.
- The CMS FETCH command copies an element into your current default directory. The copy is treated as a read-only version for information, not for modification. An element is not reserved when it is fetched, therefore fetched copies cannot be returned to the library with the CMS REPLACE command.

'he sections that follow illustrate how each command can be used in various project ituations.

.1 Getting Started – CMS SET LIBRARY

efore you can use CMS commands on an existing CMS library, you must use the CMS ET LIBRARY command to establish access to that library. To set your library, type CMS ET LIBRARY followed by the CMS library directory name. Each user must supply the ame of the library directory when using the command:

```
CMS SET LIBRARY directory
```

fter Rick built the CMS library, he told Glenn and Sue the library directory name,

[PROJECT.ADSLIB]. Glenn included the SET LIBRARY command in his LOGIN.COM file so that each time he logged on to the system, he automatically had access to the library elements. He entered the following command in his LOGIN.COM file:

```
$ CMS SET LIBRARY [PROJECT.ADSLIB]
```

Rick put the same command in his own LOGIN.COM file, but Sue decided to issue the CMS SET LIBRARY command each time she wanted to work on the data sampler.

After you have established access to a library, you can display the name of your CMS library directory by using the CMS SHOW LIBRARY command:

```
$ CMS SHOW LIBRARY
```

When you have access to an existing CMS library, CMS responds with the name of your CMS library directory. If you do not have access to a library, CMS returns the following message:

```
%CMS-E-NOREF, error referencing CMS$LIB:
```

3.2 Reserving a Library Element – CMS RESERVE

Most of your dealings with the CMS library for your project consist of reserving library elements for work and replacing modified elements back into the library. If you have established access to a library (see Section 3.1) you can select an element and reserve it.

Use the CMS RESERVE command to reserve a library element. CMS always expects you to enter a remark when you reserve an element. The remark that you enter should explain why you are reserving the element. CMS then accepts your CMS RESERVE command and copies the element into your current default directory. CMS marks the element with your reservation and records the transaction in the library transaction history. While you have an element reserved, anyone reserving any generation of the same element receives a CMS message informing them that you have the element reserved.

At one point during the data sampling project, Sue Kelley reserved the element SYNCHRON.BAS to debug a problem about lost data from one of the input channels. The reservation transaction follows:

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

CMS places a copy of the element in Sue's current default directory and displays a message, indicating that generation 2 of element SYNCHRON.BAS is reserved.

By default, CMS retrieved the most recent generation of the element SYNCHRON.BAS (generation 2). If Sue wanted to reserve an earlier generation to see if the same problem existed, she could have specified the earlier generation with the /GENERATION qualifier. That reservation transaction would have been as follows:

```
$ CMS RESERVE SYNCHRON.BAS /GENERATION=1
_Remark: Checking data line sampling
%CMS-S-RESERVED, generation 1 of element SYNCHRON.BAS reserved
```

Sue would have received a copy of the first generation of the element in her current default directory.

A remark can be entered in quotation marks in the same line with any CMS command. Sue could have typed:

```
$ CMS RESERVE SYNCHRON.BAS "losing sample from one data line"
```

If any generation of an element is already reserved by another person, CMS responds to your request by issuing a message about the reservation already in effect. You then have the option to proceed with your reservation or to quit. Usually two people choose not to work on the same element concurrently. However, should the need arise, CMS provides the mechanism to handle concurrent development. See Chapter 5 for information about concurrent reservations.

3.3 Canceling a CMS Reservation – CMS UNRESERVE

When you reserve a library element with the CMS RESERVE command and then decide not to modify that element, you can cancel your reservation with the CMS UNRESERVE command. CMS accepts your UNRESERVE command and the remark you enter with it, marks the element as available, and records the cancellation in the library history. CMS does not update the element generation number when you cancel a reservation.

The CMS UNRESERVE command is useful when you accidentally reserve the wrong element, or when you want to make an unmodified element available for another user, or when you do not want your modifications to become part of an element. When you use the CMS UNRESERVE command, the element in the library is not modified; it remains the same as it was when you reserved it.

For instance, Glenn Lewis reserved a project element to diagnose a problem. A quick view of the file showed him that he had selected an element that was not related to the problem. He used the CMS UNRESERVE command to cancel his reservation for that element.

```
$ CMS UNRESERVE SYNCHRON.BAS
_Remark: PR 4 not applicable - wrong file
%CMS-S-UNRESERVED, element SYNCHRON.BAS unreserved
```

CMS canceled Glenn's reservation, leaving SYNCHRON.BAS available for other users. CMS left the copies in Glenn's directory and recorded the transaction.

3.4 Replacing a Modified Element in the Library

After modifying a reserved element, you put the modified element back in the project library with the CMS REPLACE command. This command brings the library up to date on element modifications and documents them.

Use the CMS REPLACE command when you complete work on an element. The CMS REPLACE command copies the latest version of the element file from your current default directory into the project library. CMS then deletes all copies of that element from your directory, assigns a new CMS generation number to the stored element generation, and ends your reservation of that element. CMS also records the transaction.

For example, Rick reserved the current generation (generation 3) of the data sampler's functional specification to correct a typographical error. When he finished correcting that element, he replaced it.

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

CMS ended Rick's reservation of SPEC.RNO, deleted all copies from Rick's current default directory, and recorded the transaction.

You can use the CMS REPLACE command with its /RESERVE qualifier when you wish to update the library while retaining the element for additional modifications. This form of the command continues your reservation of the element. The /RESERVE qualifier also prevents CMS from deleting any copies of the element from your directory. CMS assigns a new generation number to the stored element generation and records the transaction along with your remark.

In the sample project, Glenn had three separate problem reports to check against a single element. As he solved each problem, he used the CMS REPLACE command with the /RESERVE qualifier to document his modifications. The first two transactions follow:

```
$ CMS REPLACE SAMPLE.BAS /RESERVE
_Remark: PR 4 - doc. error fixed - doing 5 next
%CMS-I-GENCREATED, generation 2 of element SAMPLE.BAS created
-CMS-S-RESERVED, generation 2 of element SAMPLE.BAS reserved
    .
    .
    .
$ CMS REPLACE SAMPLE.BAS /RESERVE
_Remark: PR 5 - double precision added
%CMS-S-GENCREATED, generation 3 of element SAMPLE.BAS created and reserved
```

CMS updated the library with two new generations of the element, but Glenn still had his copy for the next modification, and the element was still reserved for him. Each of the generations was documented with Glenn's comments in the library history.

You should replace a modified element whenever you reach a logical stopping point in development. This gives you a record of the specific reason for the element modification. For example, when you modify a reserved element by adding a single routine and have no further work to do on that element, use the CMS REPLACE command to document that addition and return the element to the library. When you want to make several different modifications to a reserved element, use the CMS REPLACE /RESERVE command after all but the last modification. This allows you to document each modification but retain the element for further work.

If you replace an element that was concurrently reserved and then replaced by another user, you must create an alternate development path. (Refer to Chapter 5 for information on alternate development paths.)

3.5 Getting a Read-only Copy

The CMS FETCH command copies an element file into your current default directory. Unlike the CMS RESERVE command, CMS FETCH does not mark an element reserved, and CMS does not allow you to replace a fetched copy in the library. You can fetch a copy of an element whether the element is reserved or not. Use the /GENERATION qualifier to fetch an earlier generation of the element.

Whenever Rick needed to check his modifications to the data sampling software he used the CMS FETCH command to retrieve a copy of the testing command file, TIMTST.COM. For example:

```
$ CMS FETCH TIMTST.COM
_Remark: Testing storage blocks
%CMS-S-FETCHED, Element TIMTST.COM, generation 2 fetched
```

CMS delivered a copy of the file to Rick's directory, and he used it to test his modifications. When he finished testing his modifications, he simply deleted TIMTST.COM from his directory. This transaction did not affect the library copy of the file.

A situation may arise in which you need to fetch an element that you have previously reserved. If, for example, you wish to abandon a series of edits and start over, fetch the element generation that you have previously reserved. If your current default directory already contains a copy of the file you are fetching, CMS notifies you and continues the transaction. In this case, CMS assigns the next higher file version number to the new file and displays the following messages:

```
%CMS-I-FILEXISTS, file already exists, DRA1:[TESTS]TEST.FOR;2 created
%CMS-S-FETCHED, generation 3 of element TEST.FOR fetched
```

When you issue the CMS REPLACE command, CMS always moves the highest-numbered version of an element file from your current default directory to the library. Thus, if you most recently worked on the fetched version, the changes that correspond to the fetched version will be stored in the library.

Chapter 4

CMS Show Commands

As you begin to work with an existing CMS library, you might need additional information about it. The CMS SHOW commands provide information about the library structure, history, and current status. This chapter contains information about several of the CMS SHOW commands. The following list indicates the kind of information that you can get by using each of these commands.

- What files are in the library? – CMS SHOW ELEMENT (Section 4.1)
- What files are being worked on? – CMS SHOW RESERVATIONS (Section 4.2)
- What transactions have occurred? – CMS SHOW HISTORY (Section 4.3)
- What is generation *n* of element X? – CMS SHOW GENERATION (Section 4.4)
- How did element X evolve? – CMS SHOW GENERATION/ANCESTORS (Section 4.5)
- What evolved from element X? – CMS SHOW GENERATION/DESCENDANTS (Section 4.6)

4.1 What Files Are in The Library? – CMS SHOW ELEMENT

The CMS SHOW ELEMENT command displays an alphabetical list of the elements that are in your library. For instance, in the sample project Rick used the CMS SHOW ELEMENT command to see which elements were in the project library. CMS reported six elements in the library.

```
$ CMS SHOW ELEMENT
Elements in DEC/CMS Library DRA3:[PROJECT.ADSLIB]
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"
```

4.2 What Files are Being Worked On? – CMS SHOW RESERVATIONS

The CMS SHOW RESERVATIONS command lists those reservations currently in effect. In the sample project, Rick used this command to see which elements were being worked on:

```
$ CMS SHOW RESERVATIONS
```

CMS reported which elements were reserved, who reserved the element, the generation that was reserved, when, and why. Rick used this command to determine whether or not any files from his default directory should be returned to the library.

```
Reservations in DEC/CMS Library DRA3:[PROJECT.ADSLIB]

SAMPLE.BAS
    LEWIS     1       30-JUN-1984 11:19:29 "add code for more data lines"
SYNCHRON.BAS
    KELLEY    3       18-JUN-1984 09:42:03 "integrate a/d conversion"
```

4.3 What Transactions Have Occurred? – CMS SHOW HISTORY

Whenever you create, retrieve, or replace a library element, CMS stores information about the transaction in its chronological history file. Various CMS SHOW commands display historical information about the library. For example, the CMS SHOW HISTORY command allows you to review a chronological list of all library transactions. Early in the sample project, Sue used this command to request a record of library transactions:

```
$ CMS SHOW HISTORY
```

CMS reported the following:

```
History of DEC/CMS Library DRA3:[PROJECT.ADSLIB]

 1-MAY-1984 14:22:16 SANCHEZ CREATE LIBRARY DRA3:[PROJECT.ADSLIB] "a/d data
       sampling library"
 1-MAY-1984 14:26:47 SANCHEZ CREATE ELEMENT SPEC.RNO "ADS functional
       specification"
 1-JUN-1984 12:09:02 SANCHEZ CREATE ELEMENT ADCONVERT.BAS "analog to digital
       conversion routines"
 1-JUN-1984 12:25:41 SANCHEZ CREATE ELEMENT SAMPLE.BAS "Sampling module"
 1-JUN-1984 12:29:24 SANCHEZ CREATE ELEMENT SYNCHRON.BAS "Synchronization
       routines"
 1-JUN-1984 14:01:36 SANCHEZ CREATE ELEMENT TIMTST.COM "Command procedure for
       tests"
 5-JUN-1984 14:47:40 KELLEY RESERVE SYNCHRON.BAS(1) "losing sample from one
       data line"
```

Note that CMS does not record transactions that do not alter the library. (These commands include CMS ANNOTATE, CMS DIFFERENCES, CMS SET LIBRARY, and CMS SHOW commands. CMS logs fetch transactions only if you supply a remark.)

Once your library has a large number of transactions, you can limit the CMS SHOW HISTORY display with the /SINCE = date qualifier. If Sue had used CMS SHOW HISTORY /SINCE = 15-MAY-1984, CMS would have reported only the transactions since that date.

The CMS SHOW HISTORY command can be used with the qualifier /UNUSUAL to report any abnormal library transactions that occurred, for example, two reservations in effect for the same element at the same time.

4.4 What Is Generation n of Element X? – CMS SHOW GENERATION

When you need information about a specific generation of an element, use the CMS SHOW GENERATION command to list the transaction that created the generation. When you include a generation number, it must be specified with the /GENERATION qualifier. If you omit a generation number, CMS assumes the most recent generation.

Rick investigated the creation of generation 3 with the following command:

```
$ CMS SHOW GENERATION SYNCHRON.BAS /GENERATION=3
```

CMS replied:

```
SYNCHRON.BAS      3      26-JUN-1984 09:44:12 KELLEY "a/d conversion integrated"
```

4.5 How Did Element X Evolve? – CMS SHOW GENERATION /ANCESTORS

The CMS SHOW GENERATION /ANCESTORS command lists the transactions that created each prior generation of the element. CMS produces a list in reverse chronological order with one line about each generation. Each line includes the element name, generation number, user name, date, time, and remark of the transaction that created the generation.

You can limit the listing by using the /GENERATION qualifier to specify a generation number; in this case, the generation listing begins with the specified generation. If you omit a generation number, CMS provides a complete ancestor list from the most recent library version back to generation 1.

During the data sampler project, Glenn Lewis began modifications to the element SYNCHRON.BAS sometime after Rick and Sue had both worked on it. To review the history of the element, he used the CMS SHOW GENERATION /ANCESTORS command as follows:

```
$ CMS SHOW GENERATION /ANCESTORS SYNCHRON.BAS
```

CMS listed the following transactions:

```
SYNCHRON.BAS
  3      26-JUN-1984 09:44:12 KELLEY "a/d conversion integrated"
  2      10-JUN-1984 13:10:12 KELLEY "last data line included"
  1      1-JUN-1984 12.29.24 SANCHEZ "Synchronization routines"
```

CMS SHOW GENERATION /ANCESTORS reported that the current version of SYNCHRON.BAS is generation 3, which was replaced by Sue Kelley at 9:44 A.M. on June 26 after she integrated the conversion code. The previous generation (also created by Sue) involved modifications to correct errors. Rick Sanchez placed the original element in the library on June 1.

4.6 What Evolved from Program X? – CMS SHOW GENERATION /DESCENDANTS

To get a report of all subsequent generations of an element, use the CMS SHOW GENERATION /DESCENDANTS command. It allows you to specify a generation number with the /GENERATION qualifier; generation 1 is the default generation. As in the display of ancestors, CMS SHOW GENERATION /DESCENDANTS lists the generations in reverse chronological order.

For example, Rick knew that generation 2 had been debugged, so he requested only its descendants:

```
$ CMS SHOW GENERATION /DESCENDANTS SYNCHRON.BAS /GENERATION=2
```

CMS would list the following transactions:

```
SYNCHRON.BAS
  3      26-JUN-1984 09:44:12 KELLEY "a/d conversion integrated"
  2      10-JUN-1984 13:10:12 KELLEY "last data line included"
```

The CMS SHOW GENERATION /DESCENDANTS command reported that the problem was resolved in generation 2 and that generation 3 includes new conversion code. This command is also useful if you do not know if any variant lines of descent have been created (see Chapter 5).

Chapter 5

The Evolving CMS Library

Chapters 1, 2, and 3 describe how to use CMS commands to store and update a file in a CMS library. The examples in these chapters show how to use CMS to build the main development path of an element, called the main line of descent.

Some circumstances require alternate development paths for project programs: for example, a change in scope of an existing program, trial development of a slightly different internal program structure, or the discovery of an error in an earlier generation of an existing program. To handle these circumstances, CMS allows you to establish a path that is a variant of the main line of development – a branching in the evolution of the element. CMS maintains a complete history in support of alternate development paths.

This chapter describes the use of the /VARIANT qualifier on the CMS REPLACE command for creating alternate development paths. It describes the /MERGE qualifier for the CMS RESERVE and CMS FETCH commands. The /MERGE qualifier allows you to combine two paths of development. This chapter also describes concurrent reservations. Program evolution is illustrated in diagrams that show successive states of a library.

5.1 Alternate Development Paths

An alternate development path in CMS is known as a variant line of descent. You establish a variant line by using the /VARIANT = x qualifier on the CMS REPLACE command. This creates a variant generation that CMS can distinguish from the main line of descent. The parameter *x*, called the variant letter, is any single alphabetic character. The format of the command is as follows:

```
CMS REPLACE element-name /VARIANT=x
```

CMS copies the element from your default directory into the library and labels the variant generation by appending the variant letter and the numeral 1 to the generation number. For example, if you had reserved generation 7 of an element named TEST1.FOR, you could create a variant as follows:

```
 CMS REPLACE TEST1.FOR /VARIANT=A
Remark: Routine added for multi-user system
CMS-S-GENCREATED, Generation 7A1 of element TEST1.FOR created
```

The number after the letter A identifies successive generations on that variant branch. If you reserve and replace generation 7A1 of TEST1.FOR, generation 7A2 will be created. For added meaning, you can choose a variant letter that indicates the purpose of the variant line – "A" for alternate, "B" for bug fixes, "E" to indicate enhancements, and so forth. Each variant can have variants of its own using the same /VARIANT method; for example, a variant to 7A1 could be replaced with /VARIANT = E to become 7A1E1.

During the data sampler project, Glenn knew that the program SAMPLE.BAS would have to be tested in two separate environments, and that some differences would exist in the code for each environment. He decided to establish a variant to the current main line for the second system. Glenn reserved the most recent main line element and then replaced it as a variant, starting a variant line of development. Figure 5-1 shows the main line development for SAMPLE.BAS prior to the variant path. Figure 5-2 shows the parallel line of development after the program was replaced as a variant. Glenn's procedures to establish the variant line of descent were as follows:

```
$ CMS RESERVE SAMPLE.BAS
_Remark: handle different input for second system
%CMS-S-RESERVED, element SAMPLE.BAS, generation 3 reserved
    .
    .
    .
(modification and test)
    .
    .
    .
$ CMS REPLACE SAMPLE.BAS /VARIANT=A
_Remark: Modified for second system
%CMS-S-GENCREATED, generation 3A1 of element SAMPLE.BAS created
```

Now Glenn could test the variant in a different environment, keeping the main line generation for his first target system.

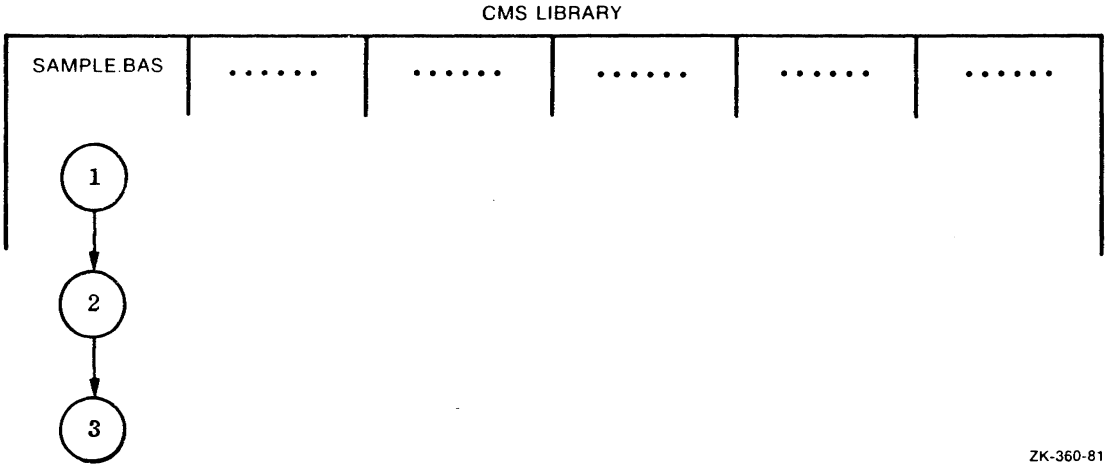


Figure 5-1: Main Line Evolution

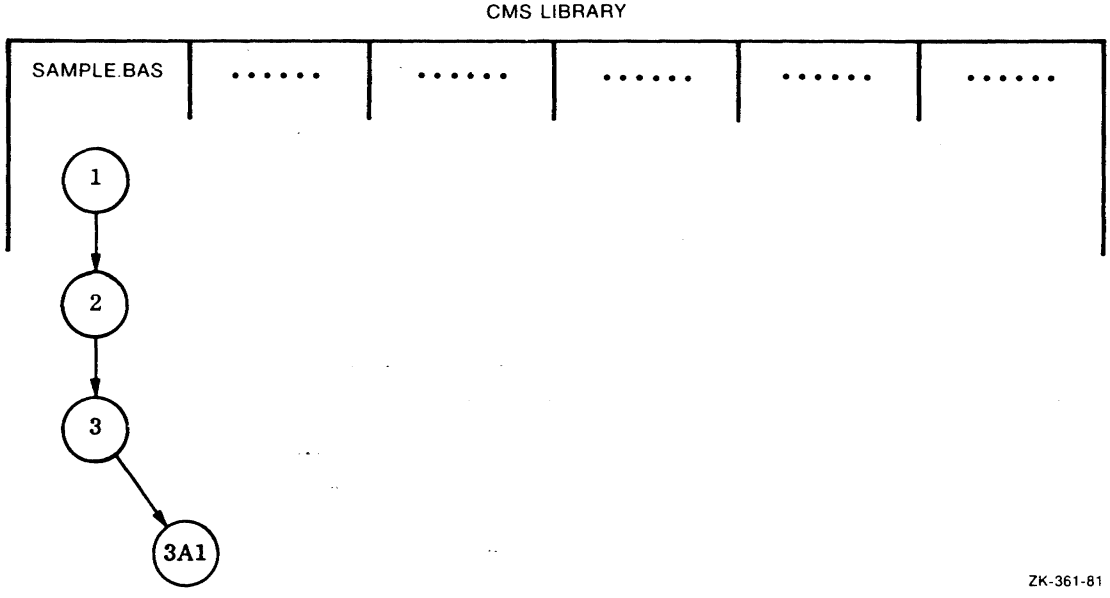


Figure 5-2: Variant Line Evolution

5.2 Merging Alternate Development Paths

At some point in development, you may want to merge the changes made to an element in a variant line back into the main line of descent. When you reserve an element from the library, you can merge any two generations of the element that are not on the same line of descent. Use the /MERGE = y qualifier on the CMS RESERVE command; *y* is the generation that is merged into the reserved generation.

If you are merging a generation on the main line into a variant generation, you reserve the variant, and specify the main line generation number as the value for the /MERGE qualifier. (The /MERGE = y qualifier can also be used with the FETCH command if you wish to only see merge results, without updating the library.)

The /MERGE qualifier directs CMS to identify the common ancestor, the most recent generation that is common to both lines of descent. Internally, the subsequent changes in both lines of descent are integrated with the common ancestor.

When you merge two generations of an element, CMS creates a file in your current default directory. This file contains the results of the merge transaction. Because merging is only a mechanical process, you must always check to be sure that the result is what you intend. If there are conflicting modifications, CMS flags these lines in the file. As you are checking the program for accuracy, you must resolve these conflicts.

To continue with the earlier example, Glenn decided after producing two generations of SAMPLE.BAS on each development path, that the two paths should be combined. The combined generation of SAMPLE.BAS would have code from each development path.

Glenn used the CMS RESERVE command with its /MERGE qualifier to combine the two generations and produce a single main line generation. Figure 5-3 illustrates the combining of the two generations into a merged copy in his default directory. Glenn used the following CMS command to merge the two generations:

```
$ CMS RESERVE SAMPLE.BAS /MERGE=3A2
_Remark: Combining sys2 version into dual-support version
%CMS-I-MERGECOUNT, 4 changes successfully merged with no conflicts
%CMS-S-RESERVED, generation 5 of element SAMPLE.BAS reserved and merged
 with generation 3A2
```

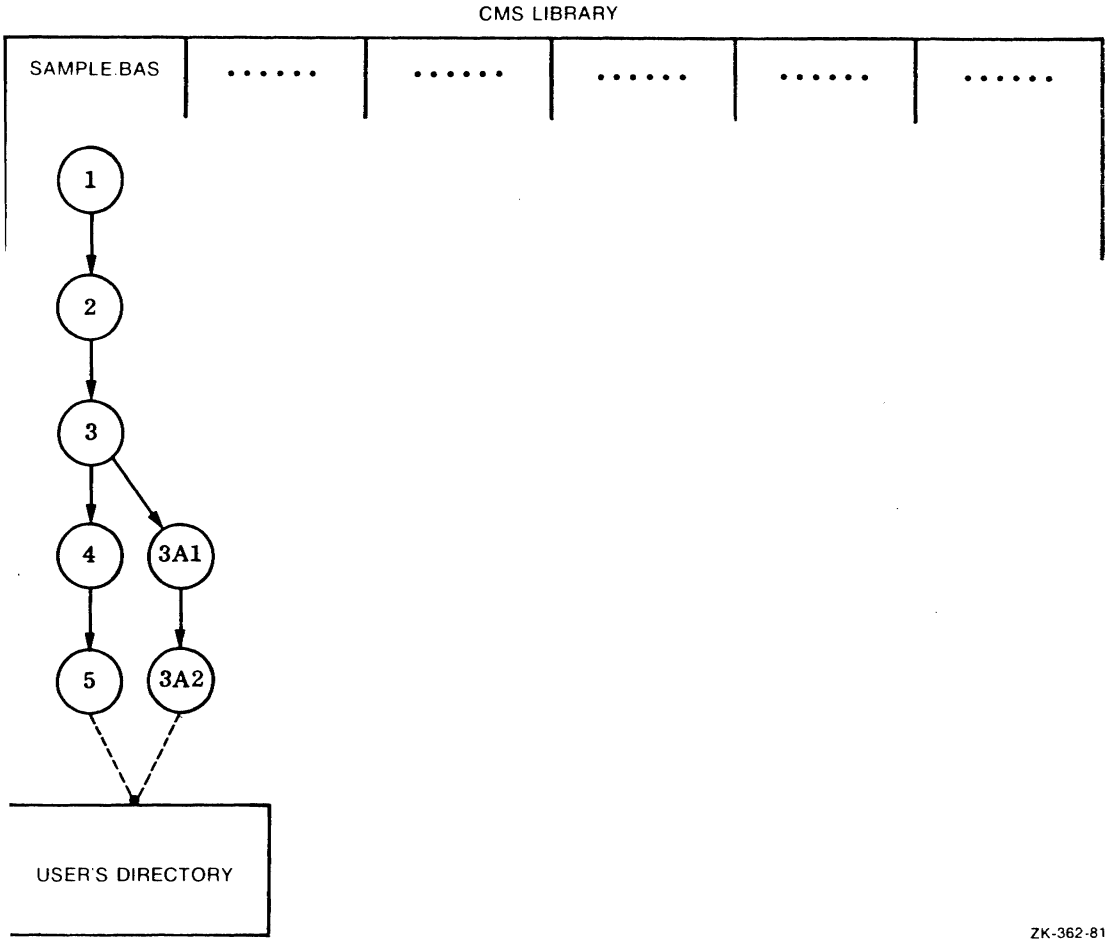


Figure 5-3: Merging Element Generations

The CMS RESERVE command with the /MERGE qualifier did the clerical work of combining the changes from the two lines of development. CMS reported that four areas of changes in generations 5 and 3A2 did not conflict and were incorporated successfully into the default directory copy. After compiling and testing the program, Glenn used the CMS REPLACE command to create generation 6 on the main line.

5.2.1 Replacing a Merged Copy

Glenn used the following command to return the merged default directory copy of SAMPLE.BAS to the library as generation 6 on the main line:

```
$ CMS REPLACE SAMPLE.BAS
_Remark: 3A2 and 5 merged for dual-support
%CMS-S-GENCREATED, generation 6 of element SAMPLE.BAS created
```

Figure 5-4 shows the library after the merged version of the element was replaced.

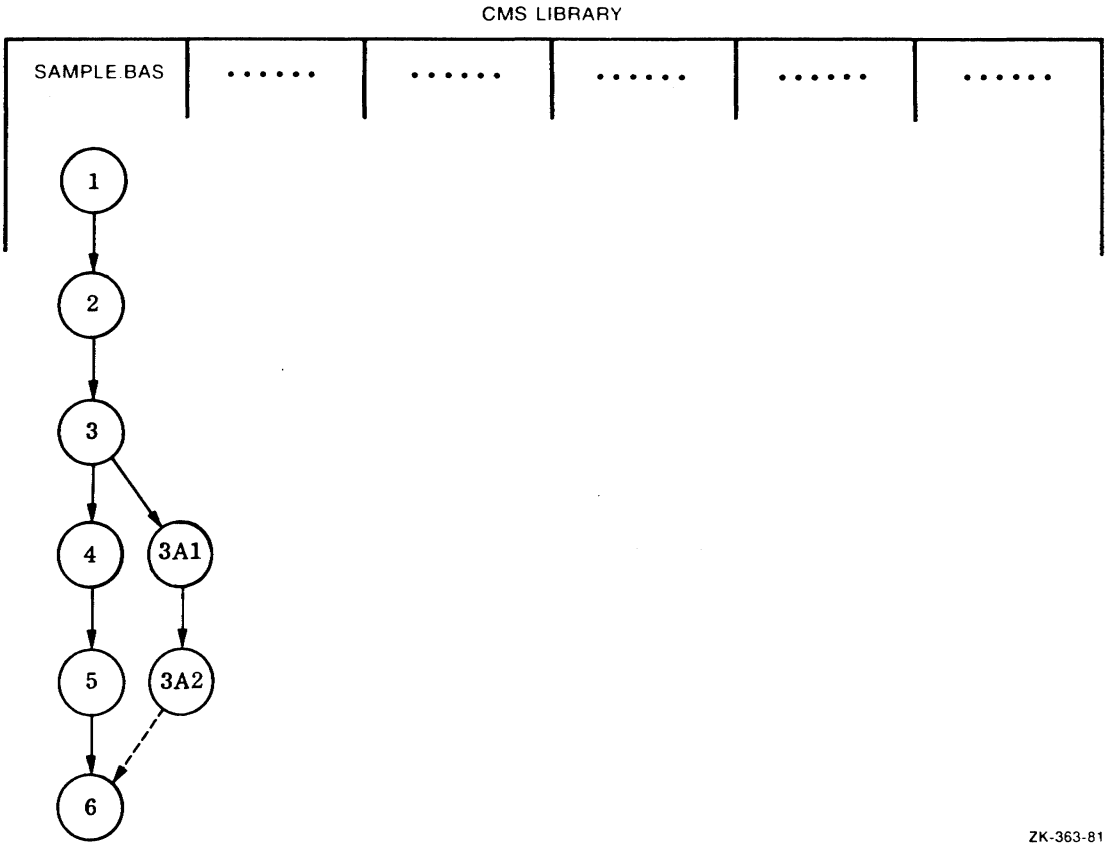


Figure 5-4: A Merged Element

5.2.2 Resolving Merge Conflicts

When you merge two generations with either the CMS RESERVE command or the CMS FETCH command, CMS checks for conflicting changes at the same line of the code. CMS notifies you of any conflicts and flags the conflicting lines so that you can resolve the problem. Example 5-1 shows a flagged conflict in a source file.

```
15      OPTION TYPE = EXPLICIT
20      DECLARE STRING DELTA_TIME
*************** Conflict 1      ***********************************************
        DECLARE STRING ASC_TIME
*****************************************************************************
        MAP (STRING_LEN) STRING ASC_TIME = 80
******** End of Conflict 1      ***********************************************
        DECLARE LONG RETCODE
50      DIM LONG BINARY_DELTA(1)
        DIM LONG NOW(1)
        DIM LONG BINARY_CVT_TIME(1)

          .
          .
          .
```

Example 5-1: Merge Conflicts Flagged

Each conflict is flagged with the word "Conflict" and a sequential conflict number in a line of asterisks. Following the asterisks, CMS displays the conflicting segments of text.

When you resolve the conflicts, use a text editor (for example, EDT) to delete the unwanted lines and the asterisks from the file, and return the copy to the library with the CMS REPLACE command.

5.3 Working Simultaneously on the Same Element

Situations can arise in which two people must work on the same element at the same time. For example, one person has already reserved an element when a second person enters a CMS RESERVE command for the same element. CMS informs the second person that the element is reserved. At this point, the transaction can be terminated, or, the second reserver can disregard the CMS message, reserve the element, and modify one of its generations. When the first reserver replaces the element, CMS reports that a concurrent reservation was made and tells who the second reserver was. Even if the second reserver has already replaced the element, CMS reports the replacement transaction to the first person. For example, if you reserve element TEST.BAS for modification after another person has reserved it, you cause a concurrent reservation for element TEST.BAS. CMS always notifies you if an element is already reserved.

If you cannot avoid a concurrent reservation, be aware that some additional effort is involved when replacing concurrently-reserved elements. The following sections show the procedures for reserving an already-reserved element and for replacing such an element into the library.

5.3.1 Reserving an Element That Is Already Reserved

If the library element you wish to modify is already reserved by another person, CMS accepts your CMS RESERVE command and the remark you enter with it. Then CMS reports that the element is currently reserved and by whom. You may not reserve an element you have already reserved. If you use the CMS RESERVE command on an element you have already reserved, CMS reminds you with the following message.

```
%CMS-E-NORESERVATION, error reserving element TEST1.FOR
-CMS-E-RESERVEDBYYOU, element TEST1.FOR is already reserved by you
```

Once CMS tells you about a previous reservation, it still gives you the option to reserve the element at that time. If you choose to proceed, CMS delivers a copy of the element to your default directory, marks the element as concurrently reserved by you, and notes the reservation transaction in the library history. Because concurrent reservations are not normal for the library, CMS notes the concurrent reservation as an unusual condition. Unusual conditions are displayed with the CMS SHOW HISTORY /UNUSUAL command.

In the sample project, when Glenn Lewis reserved the element SAMPLE.BAS, CMS reported that Sue was updating the element by adding a new search routine. Glenn chose not to work on the element at the same time. That reservation interaction was as follows:

```
$ CMS RESERVE SAMPLE.BAS
_Remark: Change conversion input
%CMS-I-RESBY, Currently Reserved by
     KELLEY Generation 7 31-SEP-1984 08:15:47 "extending conversion"
Proceed? [Y/N] (N): no
```

Glenn decided not to reserve SAMPLE.BAS at the same time Sue was updating it. He waited until she replaced the modified element into the project library before making his modifications.

Another time Glenn had no choice but to reserve an element concurrently. Rick had reserved the element SYNCHRON.BAS for modifications, and he was out of the office when Glenn needed to change the element to diagnose a problem at a test site. Glenn reserved the element as follows:

```
$ CMS RESERVE SYNCHRON.BAS
_Remark: Remote site down - 7th phone call!
%CMS-I-RESBY, Currently Reserved by
     Sanchez Generation 4 10-SEP-1982 10:36:24 "Changing FLDEXP"
Proceed? [Y/N] (N): yes
%CMS-S-RESERVED, Element SYNCHRON.BAS, generation 4 reserved
```

Once he had reserved SYNCHRON.BAS, Glenn made the modifications to solve the test site's problem. He then replaced SYNCHRON.BAS, creating generation 5.

5.3.2 Replacing an Element That Has Already Been Replaced

When two users modify the same element at the same time, the first user can replace the modified generation with the normal CMS REPLACE command to produce the next generation. However, the second user must replace the modifications as a variant generation. One or the other can then choose to merge that variant back into the main line so that both sets of program modifications appear in one generation.

When Rick returned, he used the CMS SHOW RESERVATIONS command to see what was out of the library at the moment. CMS listed his reservation of SYNCHRON.BAS and showed that Glenn had reserved and replaced the element while Rick was away.

```
$ CMS SHOW RESERVATIONS
    .
    .
    .
SYNCHRON.BAS
     SANCHEZ        4      10-SEP-1982 10:36:24 "changing FLDEXP"
  Concurrent Replacements
     LEWIS          5      14-SEP-1982 15:04:52 "Added switch test"
    .
    .
    .
```

Now Rick had to resolve the differences between generation 4 of the element, which was reserved and in his directory, and generation 5, which Glenn had replaced in the library. His choices were:

- To cancel his reservation with the CMS UNRESERVE command. This would be the simplest solution if he had not done any work on the element. Rick would then reserve the generation Glenn had replaced.
- To replace the modified element in the library as a variant generation of the element using the CMS REPLACE /VARIANT command, and then make another reservation with the /MERGE qualifier. CMS would combine the changes, identifying any conflicts between the two generations. Rick and Glenn could resolve any conflicts and replace the merged element as generation 6.

Since Rick had already modified the program, he chose the second option as the most effective, and entered the following CMS commands:

```
$ CMS REPLACE SYNCHRON.BAS /VARIANT=A.
_Remark: Fixed but w/o change for remote site
%CMS-S-GENCREATED, generation 4A1 of element SYNCHRON.BAS created
$ CMS RESERVE SYNCHRON.BAS  /MERGE=4A1
_Remark: Merging 5 (remote fix) and 4A (table fix)
%CMS-S-MERGECOUNT, 3 changes successfully merged with no conflicts
%CMS-S-RESERVED, generation 5 of element SYNCHRON.BAS reserved and merged
 with generation 4A1
    .
    .
    .
(read and test)
    .
    .
    .
$ CMS REPLACE SYNCHRON.BAS
_Remark: Replacing merged element - no changes needed
%CMS-S-GENCREATED, generation 6 of element SYNCHRON.BAS created
```

There were no conflicting lines in the merged generations. Had any of the modifications been made to the same line in the program, CMS would have noted these as conflicts.

Chapter 6

Project Organization with CMS

Earlier chapters in this manual describe how CMS can be used for storing project files and for tracking the changes to these files during development. The commands that perform these functions can be used by any one library user without a great deal of interaction with other team members. Yet CMS is a project tool, and a project usually involves a number of people. This chapter describes CMS as an organization and communication tool for any number of people on a project. The commands described are basic commands; full command capabilities are described in the *VAX DEC/CMS Reference Manual.*

6.1 The Early CMS Library

Figure 6-1 shows the early data sampler project library. Although the various elements have been worked on by more than one project member, each element has evolved independently. After the first generation of each element was placed in the library, the programmers modified and tested the elements, thereby creating new generations each time they returned the elements to the library.

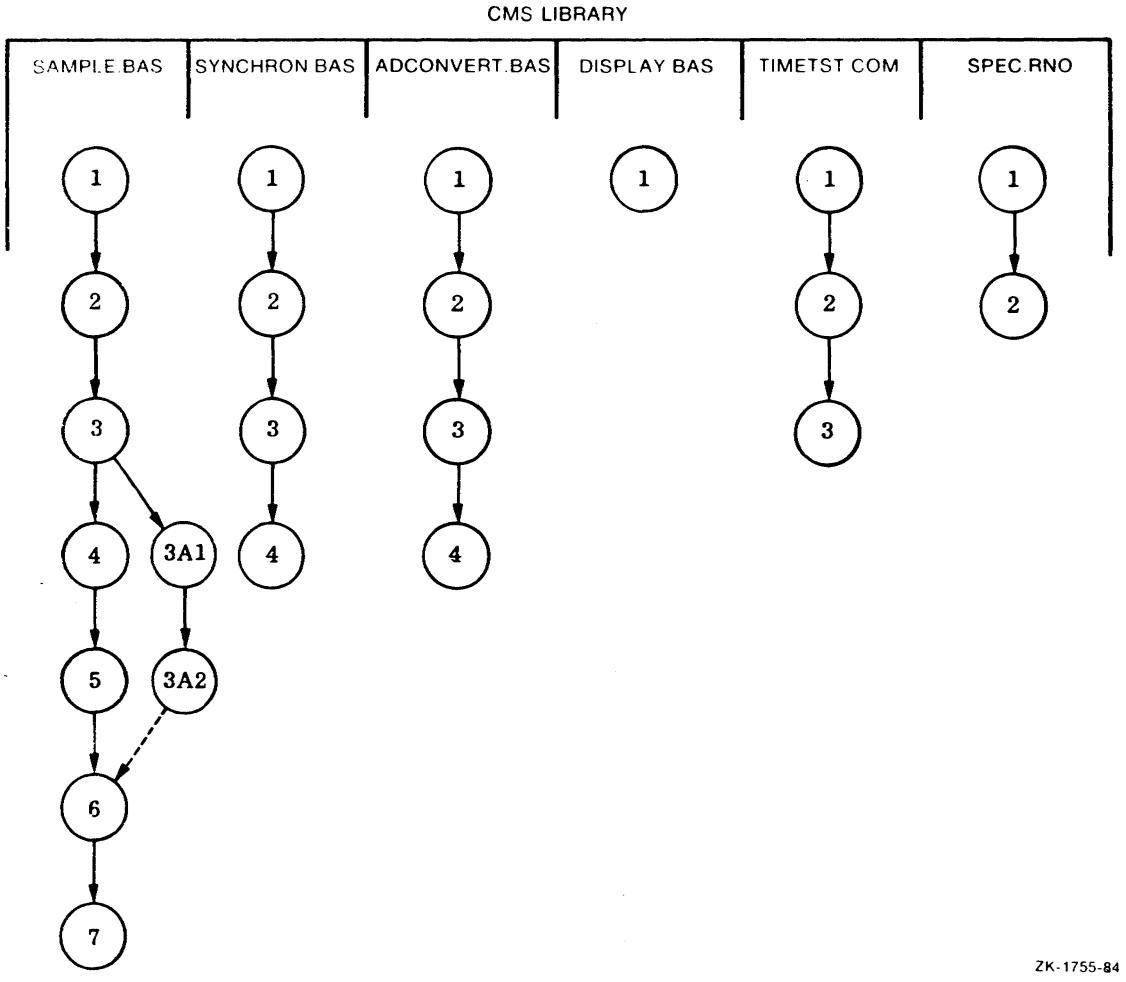


Figure 6-1: An Early CMS Library

For example, note that the element SAMPLE.BAS had evolved to its seventh generation when this "snapshot" of the library was taken; yet by the same date the functional specification, SPEC.RNO, had been modified only once. The rest of this chapter builds on this basic diagram to demonstrate how CMS aids interaction among project members. For clarity, only six of the project library elements are shown in the diagrams.

6.2 Using CMS Groups

CMS uses the element as the basic structural unit in a library. However, you can combine one or more elements into a group that you can manipulate as a unit. Also, you can use groups to reflect the general structure of the software project. The following sections describe how to build and manipulate CMS groups.

6.2.1 Building a CMS Group

To build a group, you create an empty group with the CMS CREATE GROUP command. Then you use CMS INSERT ELEMENT commands to associate elements with that group name.

The project team established a group to contain elements that handled data. They created the group and then inserted elements into that group with the following commands:

```
$ CMS CREATE GROUP DATA_ROUTINES
_Remark: routines for input & conversion
%CMS-S-CREATED, group DATA_ROUTINES created
$ CMS INSERT ELEMENT SAMPLE.BAS DATA_ROUTINES
_Remark: input sampling routines
%CMS-S-INSERTED, element SAMPLE.BAS inserted into group DATA_ROUTINES
$ CMS INSERT ELEMENT ADCONVERT.BAS DATA_ROUTINES
_Remark: analog-to-digital conversion
%CMS-S-INSERTED, element ADCONVERT.BAS inserted into group DATA_ROUTINES
```

By using the CMS INSERT ELEMENT command, you can insert individual elements into a group. You can also use the CMS INSERT GROUP command to associate a group of elements with another group. Because this is a bookkeeping function, the insertion of elements (or groups) into a specific group is only a logical connection. No physical merging of elements occurs, so you can access individual elements as usual. See the *VAX DEC/CMS Reference Manual* for more information about using these commands.

6.2.2 Using the Group Designation

Once you define a group, CMS allows you to access all the elements in the group by specifying the group name. For example, Rick wanted to check the latest generations of each of the modules that process the input data. He used the following command to fetch all of the appropriate elements:

```
$ CMS FETCH DATA_ROUTINES "checking signal storage"
```

CMS retrieved the latest main line generation of each of the elements belonging to the group DATA_ROUTINES.

6.2.3 Removing Elements from Groups

The CMS REMOVE ELEMENT command allows you to eliminate an element from a group. This command only removes the association between an element and a group; it does not alter or delete the element.

The project team had created a group named DOCUMENTATION; for some time, this group contained only the element SPEC.RNO, the functional specification for the software. As the project progressed, they hired a writer, Paul Abbott. Paul created several elements in the library for the user's manual. When the documentation was ready for review, he decided to use the group DOCUMENTATION. To keep the functional specification separate from the user documentation, he removed SPEC.RNO with the following command:

```
$ CMS REMOVE ELEMENT SPEC.RNO DOCUMENTATION
_Remark: user's manual ready for first review
%CMS-S-REMOVED, element SPEC.RNO removed from group DOCUMENTATION
```

6.2.4 Displaying the Group Structure of a CMS Library

To find out what groups are defined in your library, use the CMS SHOW GROUP command. CMS lists the group names in alphabetical order with the remark that was entered when the group was created. To obtain a list of all elements in a specific group, use the CMS SHOW GROUP command with the /CONTENTS qualifier. For example, Rick wanted to check the contents of the group named DATA_ROUTINES:

```
$ CMS SHOW GROUP/CONTENTS DATA_ROUTINES
Groups in DEC/CMS Library DRA3:[PROJECT.ADSLIB]

DATA_ROUTINES "routines for input & conversion"
    ADCONVERT.BAS
    SAMPLE.BAS
```

6.3 Using CMS Classes

CMS allows you to define sets of element generations, called *classes*. CMS keeps track of which element generations you assign to a specific class. You can use classes to represent milestones for any or all phases of a project.

The purpose of a class is determined by the methodology used by project members. For example, you can establish classes that represent different stages of development for each element generation. Each generation represents a cycle that includes several steps: reserving the element, adding or changing code, testing, and then replacing the element. You can establish classes for different stages; one for implementation, another for testing, and a third class for the generations that have completed the first two stages. As each module progresses through each stage, you assign it to the appropriate class; thus, you can easily determine your progress by displaying the contents of the different classes.

You can also use classes for system integration. When all the modules reach a working stage in development, you integrate the modules into the first working version or "base level" of the product for testing. Each module that is used in the base level corresponds to an element generation in a CMS library. You can create a class that contains each element generation used in the base level.

Different base levels represent milestones in the development of a project. For example, test versions, internal release versions, and external release versions might be significant milestones. A project can also evolve through additional releases with enhancements and corrections. These stages vary from project to project, but the life of a project is essentially the same for a payroll application program as it is for a compiler project – from design and development to test and release, then to maintenance and enhancement. You can establish one or more classes to reflect each of the different stages of a project.

6.3.1 Building a CMS Class

To build a class, you create an empty class with the CMS CREATE CLASS command. Then you use CMS INSERT GENERATION commands to place specific element generations in that class. (CMS allows only one generation of an element per class.) The /GENERATION qualifier of the CMS INSERT GENERATION command specifies the generation to be inserted. When you omit the qualifier, CMS places the most recent main line generation in the class.

On the data sampler project, the team created a class called BASELEVEL1 and assigned specific generations of the library elements to that class. Rick did this with the following commands:

```
$ CMS CREATE CLASS BASELEVEL1
_Remark: Specifying all gens for first base level
%CMS-S-CREATED, class BASELEVEL1 created
$ CMS INSERT SAMPLE.BAS /GENERATION=6 BASELEVEL1
_Remark: Checked and OKd by SK
%CMS-S-GENINSERTED, generation 6 of element SAMPLE.BAS inserted into class
BASELEVEL1
$ CMS INSERT SYNCHRON.BAS /GENERATION=4 BASELEVEL1
_Remark: Checked and OKd by RS
%CMS-S-GENINSERTED, generation 4 of element SYNCHRON.BAS inserted into class
BASELEVEL1
$ CMS INSERT ADCONVERT.BAS BASELEVEL1
_Remark: Using latest, checked OK this am - GL
%CMS-S-GENINSERTED, generation 5 of element ADCONVERT.BAS inserted into class
BASELEVEL1
$ CMS INSERT DISPLAY.BAS BASELEVEL1
_Remark: Terminal display of sampled data
%CMS-S-GENINSERTED, generation 2 of element DISPLAY.BAS inserted into class
BASELEVEL1
```

Figure 6-2 indicates which element generations were inserted in the class BASELEVEL1. The class consists of only those element generations specified by the CMS INSERT GENERATION commands. Because this is a bookkeeping function, the insertion of generations into a specific class is only a logical connection. No physical merging of elements occurs, so you can access individual generations and elements as usual with other CMS commands.

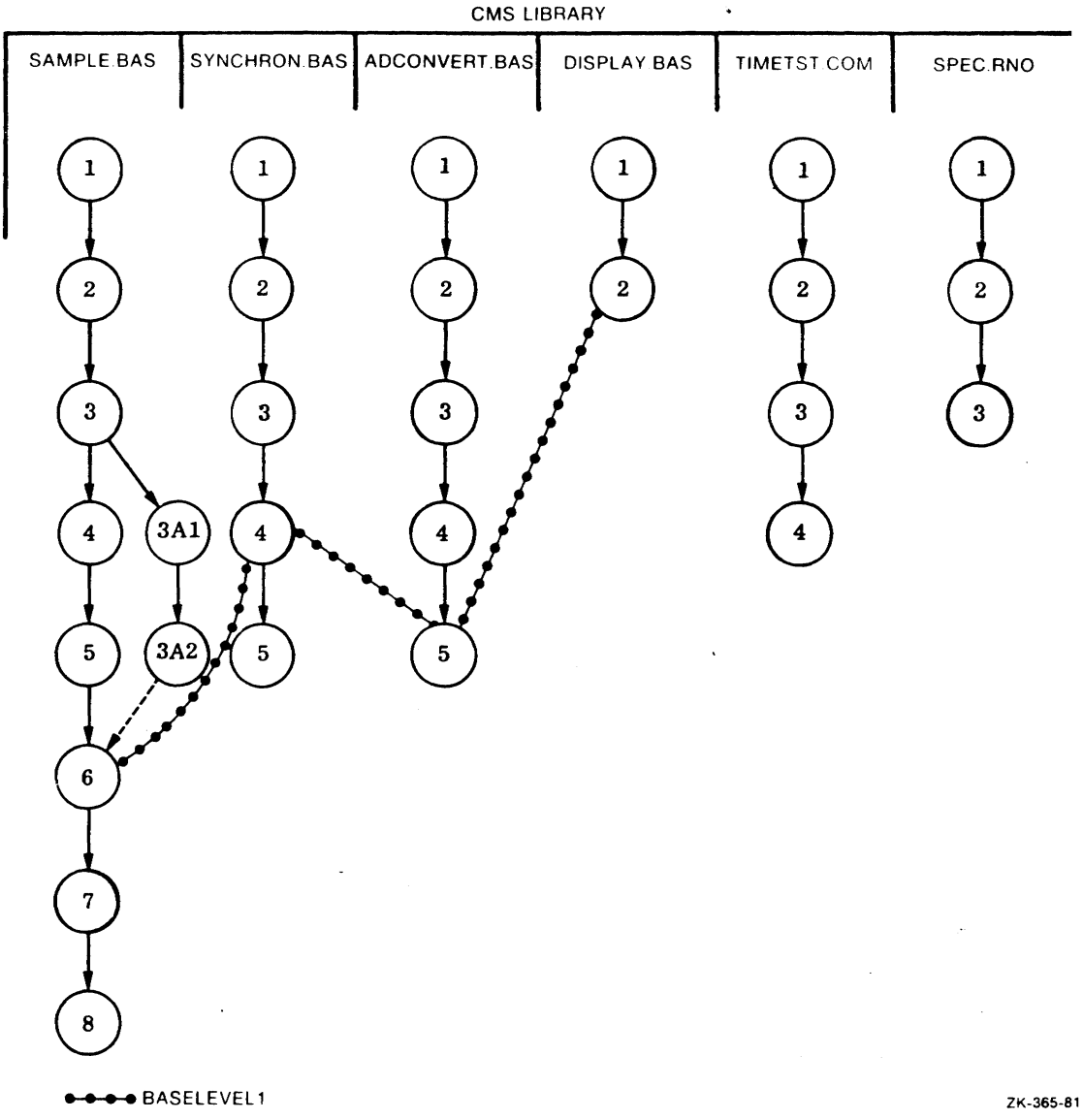


Figure 6-2: A CMS Library Showing a Defined Class

6.3.2 Using the Class Designation

Once you define a class, CMS allows you to access any element generation in that class by specifying the class name. You do not have to know the specific generation number of the element. For example, when a problem in the base level was reported to the project team, Rick had to test SAMPLE.BAS. To ensure that the proper generation of the element was tested, he used the following command:

```
$ CMS RESERVE SAMPLE.BAS /GENERATION=BASELEVEL1
```

CMS placed the proper element, generation 6, in the default directory, even though the latest generation was 7.

6.3.3 Displaying the Class Structure of a Library

To find out what classes are defined in your library, use the CMS SHOW CLASS command. CMS lists the class names in alphabetical order with the remark that was entered when the class was created. To obtain a list of all generations in a specific class, use the CMS SHOW CLASS command with the /CONTENTS qualifier. For example, Rick wanted to check the contents of the class the project was using for the first base level:

```
$ CMS SHOW CLASS/CONTENTS BASELEVEL1
```

CMS listed all of the elements and their generations inserted in BASELEVEL1 as follows:

```
Classes in DEC/CMS Library DRA3:[PROJECT.ADSLIB]

BASELEVEL1 "Specifying all gens for first base level"
    ADCONVERT.BAS        5
    DISPLAY.BAS          2
    SAMPLE.BAS           6
    SYNCHRON.BAS         4
```

6.3.4 Assigning Elements to Multiple Classes

In addition to keeping track of an established class, CMS allows you to designate some or all of the element generations in one class as belonging to another class as well. This permits a base level or release to contain various options to the standard version.

On the sample project, Rick wanted an alternate version of the first base level for demonstration purposes. He needed to suppress some of the newer features that were not fully implemented so the data sampler could work without errors when it was demonstrated. He used the following procedure to accomplish this:

1. He reserved the conversion element, ADCONVERT.BAS.
2. He modified it to ignore the new features.
3. He replaced the modified element as a variant of the standard element.
4. He established a new class called DEMOA to contain the modified synchronization element and all of the other elements that made up the base level of the data sampler.

He used the following CMS commands to create class DEMOA:

```
$ CMS CREATE CLASS DEMOA
_Remark: Runnable baselevel1 version - only for demos
%CMS-S-CREATED, class DEMOA created
$ CMS INSERT ADCONVERT.BAS /GENERATION=5D1 DEMOA
_Remark: Modified BL1 version for demos
%CMS-S-GENINSERTED, generation 5D1 of element ADCONVERT.BAS inserted into
class DEMOA
$ CMS INSERT SAMPLE.BAS /GENERATION=BASELEVEL1 DEMOA
_Remark: BL1 version OK for demos
%CMS-S-GENINSERTED, generation 6 of element SAMPLE.BAS inserted into class
DEMOA
$ CMS INSERT SYNCHRON.BAS /GENERATION=BASELEVEL1 DEMOA
_Remark: BL1 version OK for demos
%CMS-S-GENINSERTED, generation 4 of element SYNCHRON.BAS inserted in class
DEMOA
$ CMS INSERT DISPLAY.BAS /GENERATION=BASELEVEL1 DEMOA
_Remark: BL1 version OK for demos
%CMS-S-GENINSERTED, generation 2 of element DISPLAY.BAS inserted in class
DEMOA
```

Figure 6-3 shows the variant, 5D1, branching from the main line of descent for element ADCONVERT.BAS. The two different base levels are indicated. Note that BASELEVEL1 and DEMOA are identical except for ADCONVERT.BAS, which has a different generation in each class.

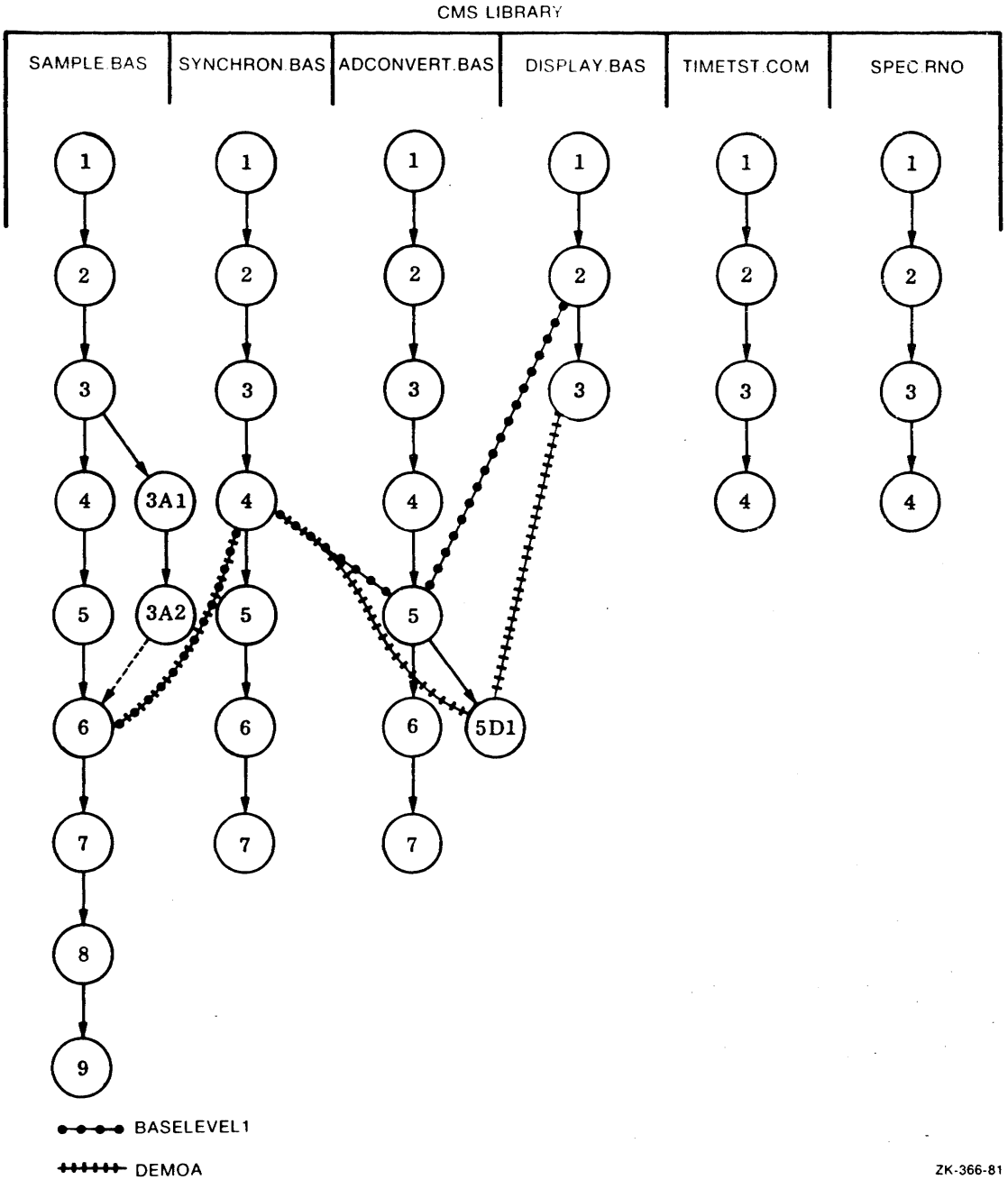


Figure 6-3: Multiple CMS Classes

6.3.5 Removing an Element from a Class

The CMS REMOVE GENERATION command allows you to eliminate an element generation from a class. For example, suppose that the first base level was going to be reduced in scope, and DISPLAY.BAS dropped from the class BASELEVEL1. The command to accomplish this task would be:

```
$ CMS REMOVE GENERATION DISPLAY.BAS BASELEVEL1
```

CMS would then revise its information about BASELEVEL1 so that DISPLAY.BAS would not be included in the class. All future references to BASELEVEL1 would refer to the class without DISPLAY.BAS.

6.3.6 Defining a Release with CMS

The process for establishing a release version of library elements is the same as it is for defining a base level – you create a class and then insert appropriate element generations into it. The class name could be the release version number (REL1 or RELEASE2A, for instance). CMS still maintains the audit trails back to the beginning of the project so that the complete project history is available.

6.3.7 Supporting a Product Using CMS

When an error is reported in either a base level or a release version of a product, careful tracking is needed to ensure that the appropriate generation is corrected. It is also necessary to maintain the integrity of that version with proper historical information. CMS helps merge the modification into the product code and preserves the integrity of the library history.

The CMS INSERT GENERATION command, when used with the /ALWAYS qualifier, lets you correct a class and replace the problem generation with a corrected generation. Any existing generation of that element is removed from the class, and the generation specified takes its place.

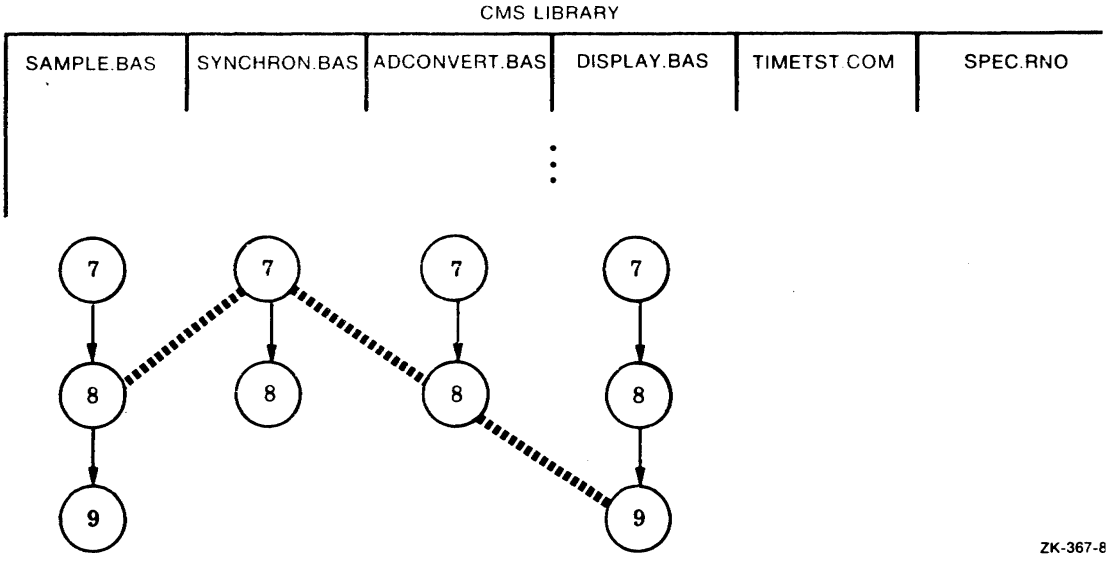
In the sample project, an error within an element used in the first release required correction. The corrected element was needed to replace the incorrect module in the release class. Mike corrected the problem in the following way.

1. He reserved the element generation containing the error.
2. He modified and tested the element until it was satisfactory.
3. He replaced the corrected element as a variant.
4. He inserted the generation in the class while superseding the previous generation of the element.

These are the commands he used:

```
$ CMS RESERVE DISPLAY.BAS /GENERATION=RELEASE1
_Remark: PR No. 4 - To fix uninitialized variable
%CMS-S-RESERVED, element DISPLAY.BAS, generation 9 reserved
    .
    .
    .
(debug, modification and test)
    .
    .
    .
$ CMS REPLACE DISPLAY.BAS /VARIANT=B
_Remark: Additional initialization done -  R1-PR-4 fixed
%CMS-S-GENCREATED, generation 9B1 of element DISPLAY.BAS created
$ CMS INSERT GENERATION DISPLAY.BAS /GENERATION=9B1 RELEASE1 /ALWAYS
_Remark: PR No. 4 - replacement for gen 9 bug in Rel 1
%CMS-S-GENINSERTED, generation 9B1 of element DISPLAY.BAS inserted into
 class RELEASE1
```

Part A of Figure 6-4 shows the portion of the library with the class RELEASE1 in its original form. Part B of Figure 6-4 shows the project library after the error in element DISPLAY.BAS had been corrected. The class name RELEASE1 refers to the class containing the corrected element DISPLAY.BAS. The class that contained generation 9 of DISPLAY.BAS is not maintained.



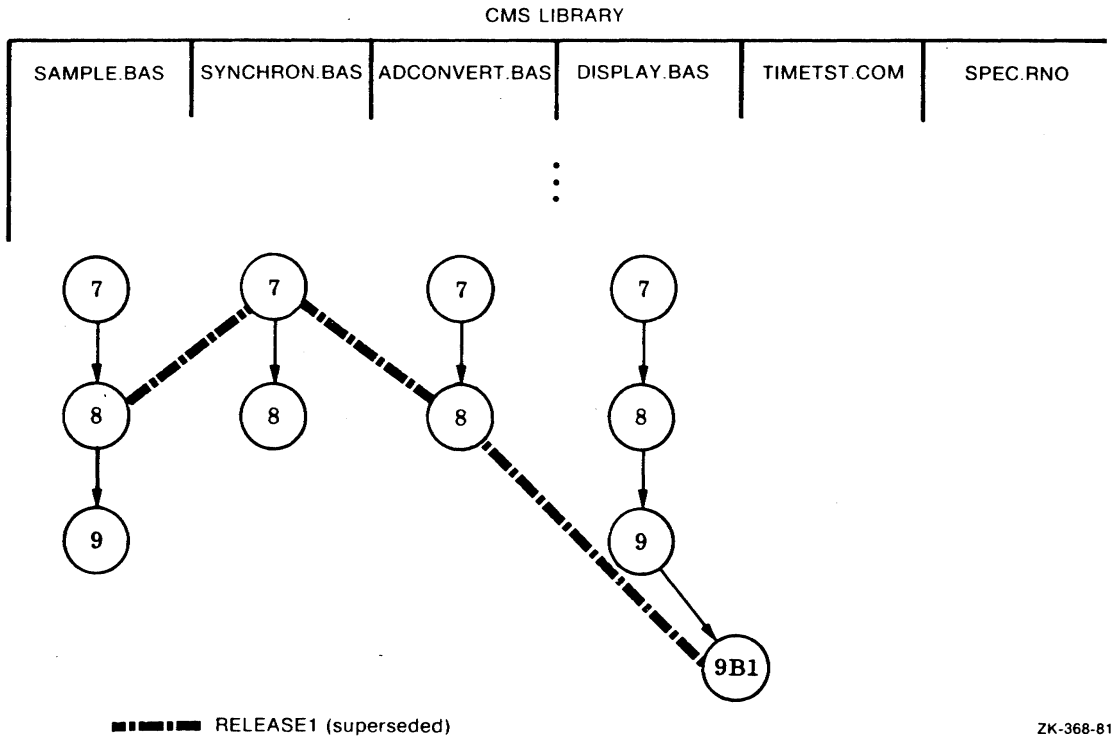


Figure 6-4 : A CMS Class Containing a Superseded Element Generation

Chapter 7

Additional CMS Commands

This chapter describes the CMS ANNOTATE and CMS DIFFERENCES commands:

- The CMS ANNOTATE command provides the history and line-by-line information about a library element. CMS lists the history of all changes ever made to the element and then lists each line of the element, tagging each line with the number of the generation in which the line appears. CMS does not record an ANNOTATE operation in the library history.
- The CMS DIFFERENCES command performs a line-by-line comparison of any two ASCII files. CMS does not record a DIFFERENCES operation in the library history.

7.1 Getting Line-by-Line Information

CMS maintains information about changes to each element in the library. The CMS ANNOTATE command produces a line-by-line listing of the changes made in any generation of a specified element. The annotated listing begins with a summary of changes to the element (when it was modified, which generation was modified, and who did the modifying). Then, CMS lists the program and labels all lines that were not in the first generation. Each line that has been added (or changed) is marked with the generation number to indicate when the line was added or when it was last modified. The annotation process writes the listing in a file in your default directory. By default, CMS gives the file the same name as the element, but with a file type ANN. To annotate an earlier generation of the element, use the /GENERATION qualifier.

When Sue Kelley wanted to see what modifications Rick Sanchez made to the element TIMECVT.BAS, she used the following command:

```
$ CMS ANNOTATE TIMECVT.BAS
%CMS-S-ANNOTATED, Element TIMECVT.BAS, generation 3 annotated
```

CMS delivered a file, TIMECVT.ANN, to Sue's current default directory. Example 7-1 shows the printed annotated output. The historical information at the top of the listing indicates the user, date, time, and remark entered each time a new generation was created. The second entry of the history shows that Rick replaced a modified copy of TIMECVT.BAS, creating generation 2. The line-by-line information indicates that Rick modified line 6.

Example 7-1 shows a sample of the information delivered by the CMS ANNOTATE command.

```
Annotated listing for element TIMECVT.BAS in DEC/CMS Library DRA3:[LEWIS.BCCMS]  25-MAR-1984 15:50:29


*3    25-MAR-1984 15:49:01 KELLEY "add check for invalid delta time"
*2    25-MAR-1984 15:39:58 SANCHEZ "JP - fixed length string required"
*1    25-MAR-1984 15:37:11 SANCHEZ "time conversion program"

Annotated listing for element TIMECVT.BAS in DEC/CMS Library DRA3:[LEWIS.BCCMS]  25-MAR-1984 15:50:29


        1       10      rem Program to compute an absolute time given the present time
        2               rem and a delta time. The result is written to a file.
        3
        4       20      OPTION TYPE = EXPLICIT
        5               DECLARE STRING DELTA_TIME
2       6               MAP (STRING_LEN) STRING ASC_TIME = 80
        7               DECLARE LONG RETCODE
        8               DIM LONG BINARY_DELTA(1)
        9               DIM LONG NOW(1)
        10              DIM LONG BINARY_CVT_TIME(1)
        11
        12      100     EXTERNAL LONG CONSTANT SS$_NORMAL
3       13              EXTERNAL LONG CONSTANT SS$_IVTIME
        14              EXTERNAL LONG FUNCTION LIB$ADDX
        15              EXTERNAL LONG FUNCTION LIB$SUBX
        16              EXTERNAL LONG FUNCTION LIB$INT_OVER
        17              EXTERNAL LONG FUNCTION SYS$BINTIM (STRING BY DESC, LONG BY REF)
        18              EXTERNAL LONG FUNCTION SYS$GETTIM (LONG BY REF)
        19              EXTERNAL LONG FUNCTION SYS$ASCTIM (LONG BY REF, STRING BY DESC, &
        20                                                 LONG BY REF, LONG BY REF)
        21      150     LET RETCODE = LIB$INT_OVER(0)
        22              PRINT "Input delta time"
        23              INPUT DELTA_TIME
        24              LET RETCODE = SYS$BINTIM ( DELTA_TIME, BINARY_DELTA(0) )
3       25      175     IF (RETCODE = SS$_NORMAL) THEN GOTO 200
3       26              ELSE IF RETCODE = SS$_IVTIME THEN      &
3       27                   PRINT ,"INVALID TIME"
3       28                   GOTO DONE
3       29                   END IF
3       30              END IF
3       31      200     LET RETCODE = SYS$GETTIM(NOW(0))
3       32              IF (VAL( DELTA_TIME ) > 0 ) THEN       &
        33              RETCODE=LIB$ADDX(NOW(0),BINARY_DELTA (0) ,BINARY_CVT_TIME(0))
        34              END IF
        35              LET RETCODE = SYS$ASCTIM(,ASC_TIME,BINARY_CVT_TIME(0),)
        36              OPEN "TIME.TMP" FOR OUTPUT AS FILE #1
        37              PRINT #1,ASC_TIME
        38              CLOSE #1
        39      32767   Done:  END
        40
```

Example 7-1: Sample Annotated Listing

7.2 Comparing Files

The CMS DIFFERENCES command allows you to do line-by-line comparisons of two ASCII files. The files can exist in a CMS library or in a non-library directory. (If the files are in a CMS library, they must be in the same library; you cannot use CMS to compare elements from different libraries.)

When CMS compares two files, it outputs a listing file to your default directory. By default, the listing file has the same name as the first file specified in the CMS DIFFERENCES command, with a file type of DIF. The file lists the names of the two files that were compared, and then lists all differing lines of the two files in sequential order with line numbers and full line text. CMS DIFFERENCES does not generate an output file if the comparison does not result in any differences.

On the data sampler project, Rick used the CMS DIFFERENCES command to check a bug fix. He compared the element generation that contained the bug against the next generation by using the CMS DIFFERENCES command:

```
$ CMS DIFFERENCES TIMECVT.BAS /GENERATION=2 TIMECVT.BAS /GENERATION=1
%CMS-I-DIFFERENT, files are different
```

Rick used the /GENERATION qualifier to direct CMS to use generations 1 and 2 of TIMECVT.BAS in the comparison. Example 7-2 shows the report from this comparison, TIMECVT.DIF.

```
DEC/CMS File Comparison Utility
Files Compared By SANCHEZ On 25-MAR-1984 15:51:58
   (1)  Element TIMECVT.BAS Generation 2
   (2)  Element TIMECVT.BAS Generation 1


**** Generation Differences ****

TIMECVT.BAS
  3     25-MAR-1984 15:49:01 KELLEY "add check for invalid delta time"
 *2     25-MAR-1984 15:39:58 SANCHEZ "JP - fixed length string required"
 *1     25-MAR-1984 15:37:11 SANCHEZ "time conversion program"


**** Text Differences ****


+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
Element TIMECVT.BAS(2) Line 6
      1)        MAP (STRING_LEN) STRING ASC_TIME = 80

Element TIMECVT.BAS(1) Line 6
      2)        DECLARE STRING ASC_TIME
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -



**** End of Differences ****
```

Example 7-2: Sample CMS DIFFERENCES Listing

TIMECVT.DIF shows the element generations used in the comparison, the generation history for the element, and the differences between the two generations.

Appendix A

Notes on Using CMS Libraries

This appendix contains information that may be useful to you as you build and use your CMS library. In general, project planning will have the greatest impact on how you can best use CMS. Each project has its own characteristics that determine how you should organize it.

The Library Storage Method

CMS stores the entire text of the first generation of an element. Each time you replace an element, CMS determines what has been changed in the element file(s). In order to save storage space, only the new and changed lines of successive generations are stored. The rule of thumb for estimating online mass storage for the library is to allow three times the amount of space that you would normally allow for one copy of all project files.

A file in a CMS library can be portrayed as shown in Example A-1.

```
    .
    .
    .
(1,2,3)  APPLES
(1,2,3)  BANANAS
(1,2,3)  CHERRIES
(1)  POOCHES
(2)  PAUNCHES
(3)  PEACHES
(1,2,3)  ELDERBERRIES
    .
    .
    .
```

Example A-1: The CMS Storage Method (Simplified)

Example A-1 shows that each data item is numbered according to the element generations in which the item appears. The line "POOCHES" (occurring in the first generation) has been changed to "PAUNCHES" in the second generation, and changed again to "PEACHES" in the third generation.

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 person who made those changes.) However, in the library, the data require only seven lines of storage space. Conventional storage methods would require 15 lines, five lines for each of the three copies of the data.

Project members do not have to keep back-up copies of CMS library files in their own account. Normal system backup procedures should be followed for the contents of a CMS library, as for any valuable files. CMS itself keeps a certain amount of back-up information to 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 only stores 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. Since the bulk of the program does not change, this kind of element is ideal for a CMS library. On the other hand, the same program listing file would change drastically 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.

Glossary

Class

A set of element generations with only one generation per element.

Concurrent replacement

A library transaction that replaces an element that has been marked as concurrently reserved.

Concurrent reservation

Two or more reservations in effect for the same element at the same time.

Element

An ASCII file that is stored in a DEC/CMS library.

Fetch

Retrieves a copy of a library element that (in terms of library tracking) is for read-only purposes; the element is not reserved.

Generation

Representation of a phase in the development of an element: every time you retrieve and then return an element to the library (see Reservation and Replacement) a new generation is created. Any generation of an element can be retrieved; each generation reflects the changes that were made at that particular point in development.

Generation number

A number or a combination of numbers and letter(s) that identify an element generation.

Group

A set of elements that can be manipulated as a unit.

History

The historical record of library access (generally includes all transactions except those that display library information or that access library contents for read-only purposes).

Library

The largest group of files that DEC/CMS recognizes as a unit.

e of descent

A series of generations of an element, created by successive reservation and replacement transactions.

n line of descent

The series of generations of an element that are identified by single generation numbers.

'ging

Combining two generations of an element that do not lie on the same line of descent.

lacement

A library transaction in which a user returns a reserved element to a library, thus ending the reservation.

ervation

A library transaction in which a user retrieves a copy of an element file from the library. DEC/CMS marks that library element as reserved by the user. For the duration of the reservation, if any user reserves that element, CMS indicates that it is reserved.

iant generation

An element generation that does not lie on the same line of descent as its predecessor.

iant letter

A letter used in a generation number to identify a variant line of descent.

iant line of descent

A line of descent that is separate from the main line of descent: an alternate development path. Generation numbers of variant line generations consist of combinations of numbers and variant letter(s).

INDEX

D

E

F

G

H

S

T

U

V

W

User's Introduction to
VAX DEC/CMS
AA-L371B-TE

Reader's Comments

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement. _____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number.

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

---Do Not Tear - Fold Here and Tape---

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

---Do Not Tear - Fold Here and Tape---