# Guide to VAX DEC/Module Management System

Order Number: AA–P119D–TE

**May 1989**

This manual describes how to use MMS for building software systems.

| | |
|---|---|
| **Revision/Update Information:** | This revised manual supersedes the VAX DEC/MMS User's Guide Order No. AA–P119C–TE. |
| **Operating System and Version:** | VMS Version 5.0 or higher |
| **Software Version:** | DEC/MMS V2.5 |

**digital equipment corporation**
**maynard, massachusetts**

**First printing, March 1983**
**Revised, June 1984**
**Revised, April 1987**
**Revised, May 1989**

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.



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

The following are trademarks of Digital Equipment Corporation:

| | | | |
|---|---|---|---|
| ALL-IN-1 | DECUS | P/OS | UNIBUS |
| DEC | DECwriter | Professional | VAX |
| DEC/CMS | DIBOL | Q-bus | VAXcluster |
| DECMMS | EduSystem | Rainbow | VMS |
| DECnet | IAS | RSTS | VT |
| DECmate | MASSBUS | RSX | Work Processor |
| DECsystem-10 | PDP | RT | |
| DECSYSTEM-20 | PDT | ULTRIX | |

digital™

ZK5208

# Contents

## Chapter 2 The MMS Description File

## Command Dictionary

## Appendix A MMS Messages



## Appendix B MMS and UNIX *make* Comparisons

## Appendix C DEC/MMS Built–In Features

## Figures



## Tables

# Preface

## Objective

This manual explains how to use VAX DEC/Module Management System (MMS). It is a guide that provides both tutorial and reference material to illustrate basic and advanced techniques.

## Intended Audience

This manual is primarily intended for software engineers but it can be useful for managers, technical writers, and other MMS users. Those with experience using the UNIX® *make* utility should be able to use MMS with little trouble, because MMS is patterned after *make*.

MMS runs on the VMS operating system Version 5.0 or higher.

## Structure of This Document

The *Guide to VAX DEC/Module Management System* is divided into four chapters, a command dictionary section, three appendixes, and a glossary.

- Chapter 1 gives an overview of MMS, describes how MMS automates the software development cycle, and offers a tutorial for new users of MMS.
- Chapter 2 presents the concepts of description files, built-in rules, macro definitions, and directives.
- Chapter 3 describes advanced techniques for using MMS as efficiently as possible.
- Chapter 4 explains how MMS can process files stored in VMS, CMS, and VAX FMS libraries and records stored in the VAX Common Data Dictionary.
- The Command Dictionary explains the MMS command line format and contains detailed descriptions of all MMS qualifiers. The qualifier descriptions are listed alphabetically by qualifier name.

® UNIX is a registered trademark of American Telephone and Telegraph Company

- Appendix A lists and explains all MMS messages.
- Appendix B describes the differences between MMS and UNIX *make* features.
- Appendix C contains tables of MMS defaults, with explanatory information.
- The Glossary defines important terms.

## Associated Documents

- The *VAX DEC/Module Management System Quick Reference Guide* (Order No. AV–P120D–TE) provides a concise summary of MMS rules and qualifiers.
- *VAX DEC/Module Management System Installation Guide* (Order No. AA– P121D–TE) supplies the instructions for installing MMS on a VMS system.

## Conventions Used in This Document

| Convention | Meaning |
|---|---|
| [ ] | Brackets indicate that the enclosed item is optional. |
| { } | Braces enclose a list from which one element must be chosen. |
| \| | The OR symbol separates alternatives within braces or brackets. For example, { filespec\|"macro" means that you must type either a file specification or a macro enclosed in quotation marks.} |
| . . . | A horizontal ellipsis indicates that the preceding items can be repeated one or more times. |
| 'string' | A term enclosed in apostrophes is information that can vary. (This convention is used frequently in Appendix A.) |
| *italics* | A term that appears in italics is defined in the Glossary. |

Unless otherwise noted the following apply:

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

# Chapter 1

# Introduction to MMS

This chapter introduces you to MMS. It is divided into three sections:

- A brief overview that describes how MMS can help you to solve some of the problems that arise during software development
- A tutorial section on MMS showing you how to specify building a software system, building the system from these specifications, and rebuilding the system if parts of it change
- A summary of MMS command qualifiers

## 1.1 Overview

VAX Module Management System (MMS) is a tool that automates and simplifies the building of software systems. MMS allows you to specify exactly how a software system is to be built. It can build simple programs consisting of one or two source files or complex programs consisting of several source files, message files, and documentation files. MMS builds software systems accurately from their system descriptions and can rebuild systems quickly if parts of the systems change.

If you have used the UNIX® *make* utility, you should easily be able to make the transition to MMS, because MMS is patterned after *make*.

® UNIX is a registered trademark of American Telephone and Telegraph Company

### 1.1.1 MMS and the Software Development Cycle

Software development is an iterative process that involves the following basic steps:

- Thinking about the problem and creating a design
- Writing the code based on the design
- Building the system
- Testing the software

MMS automates the building step in the cycle so that you have more time for the creative aspects of software development.

When more than one programmer is working on a software development project, the components of the software system are usually stored in a common source directory, which may be a VAX Code Management System (CMS) library. Each programmer has copies of the sources, which he or she edits and then replaces. MMS can simplify this procedure.

A software system can have hundreds of source files, object libraries, and include files. It can have multiple compilers and compilation and linking options. Due to these variables, it is hard to exactly reproduce the same program image at each build. The MMS description of a software system can be used by anyone to accurately create the software.

### 1.1.2 Solving Development Problems with MMS

MMS allows you to specify in a description file exactly how a system is to be built. A *description file* is a text file that you can create with any VMS editor. You can describe to MMS all the important components of a system build: the source files that make up the system, the compilers to use, the order in which to link the modules, and the libraries to use during the link.

#### Building Systems Accurately and Consistently

Each time you run MMS, it follows the description and builds the same system. There is no human error during the system creation. You can use older descriptions to recreate previous versions of the system. The software can be built by anyone with or without technical knowledge.

### Identifying Dependencies

As you describe a system to MMS, you also state logical dependencies in that system. For example, you describe the text files used in source files, the source files that make up each object, the objects that make up each image, and the libraries used in a link. After a system is built once, MMS uses the dependencies to rebuild the system quickly.

### Rebuilding Efficiently

During the development cycle, MMS can determine which components in a system have been changed and what other components are affected by these changes. For example, if you change several source files, MMS can determine which corresponding object modules need to be updated and then can update them. Therefore, the entire system is not rebuilt, only those components whose sources have been modified.

If you rebuild a complete system without regard to which components need updating, you waste disk space. MMS has a /SKIP_INTERMEDIATE qualifier that avoids unnecessary building of intermediate files. For example, if you have a source file PROG.C and an executable file PROG.EXE but no intermediate PROG.OBJ file, when you use the /SKIP_INTERMEDIATE qualifier, MMS does not recreate the intermediate file as long as the executable file is newer than the source file.

### Building and Testing Components

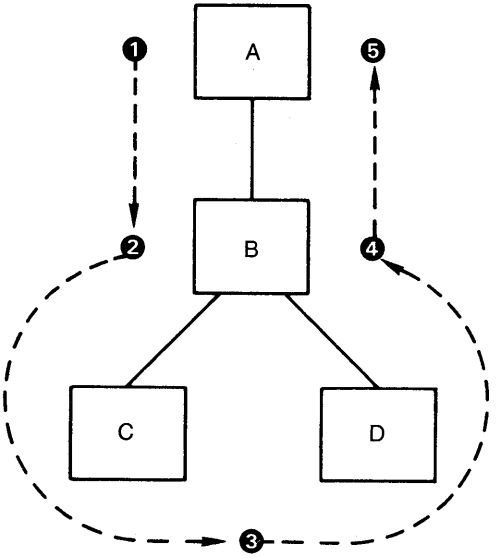
You can use MMS to build and test modules locally before building or testing the modules in the source directory or library. If you use MMS in conjunction with CMS, you can be sure that the replaced modules do not conflict with the edits another programmer may have made. You can use these two tools together to ensure that only your changes, and not another programmer's, are included in the revised module.

CMS helps manage a project's files by storing them in a library, tracking changes, and controlling access to the library that contains the files.

### Building the System with MMS

Figure 1–1 depicts a small software system and describes the basic steps MMS follows when it builds the system. In this system, Component A is the *target*—the file that you want to update. Component B is a *source* for Component A, and Components C and D are sources for Component B. The commands that update B (by using C and D) and A (by using the updated B) are called *actions*. (For example, the LINK command is the action that uses B.OBJ to update A.EXE.)

**Figure 1–1: How MMS Builds a Software System**




ZK-1090-82

❶ MMS checks the revision time of the target (Component A).

❷ MMS checks the revision time of the first source (Component B).

❸ MMS checks the revision time of Components C and D against that of B.

❹ If the times of Components C or D are more recent than that of Component B, MMS updates B according to *action lines* that you specify in the description file (action lines tell MMS what commands to execute to update the components of a software system). If B is more recent than C and D, MMS does not do anything because B is already up to date.

❺ Once Component B is updated, it is more recent than the target. Therefore, MMS updates Component A.

If the target has been modified since the sources were last changed, MMS does not update the target. Instead, it issues a message to inform you that the target is already up-to-date.

## 1.2 Getting Started

This section introduces the basic features of MMS including how to create a description file, how to use this description file to build your system, and how to rebuild your system when only parts of it have changed. This section focuses on the built-in and default features of MMS.

## 1.2.1 The MMS Description File

The first step in using MMS is to write a description file for the system you want to build. An MMS description file is a text file that describes how to build a software system. It must have the .MMS extension or file type. A description file can contain the following:

- The target or goal of building the system (the executable image or set of images)
- The intermediate files that are used to create the target (usually object files)
- The source files used to create the intermediate files (usually program code)
- Optional information to help create the target or intermediate files

You invoke MMS from the DCL command line in the following format:

```
$ MMS
```

When you invoke MMS, it looks in your current directory for a description file called DESCRIP.MMS. If MMS cannot find DESCRIP.MMS, it looks for a description file called MAKEFILE. (If both DESCRIP.MMS and MAKEFILE. exist in your directory, MMS uses only DESCRIP.MMS.) Once it locates the description file, MMS processes it. If you specified a target on the command line, MMS begins processing the description file with the first rule that describes how to build that target. If you did not specify a target on the command line, MMS builds the first target in the description file. If neither DESCRIP.MMS nor MAKEFILE. exists, MMS issues an error message and aborts execution.

You can also use the /NODESCRIPTION qualifier with the MMS command to override the use of a description file. If you specify /NODESCRIPTION with the MMS command, you must specify the target on the MMS command line so that MMS knows what to build. When /NODESCRIPTION is in effect, MMS does not look for a description file but relies entirely on its built-in rules to update the target.

The format of a description file dependency rule is as follows:

```
target, . . .   : [source, . . . ]        [! comment]
     [action line . . . ]            [! comment]
```

The *target* is a file that must be built to complete the software system. Targets can be executable image files or object files. The *source* is a file used to create a target. For example, object files are sources for executable files and source code is the source for object files. The action line is a DCL command that MMS uses to update the target.

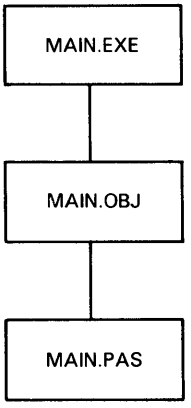
The hyphen (-) is a continuation character and is allowed on any line in a description file. You can use the keywords **DEPENDS_ON** and **ADDITIONALLY_DEPENDS_ON** in place of the colon and double-colon, respectively, in a dependency rule. These keywords are used exclusively in the examples in this section for clarity.

A simple description file called SYSTEM1.MMS could contain the following dependencies:

```
MAIN.EXE DEPENDS_ON MAIN.OBJ
MAIN.OBJ DEPENDS_ON MAIN.PAS
```

SYSTEM1.MMS is a description file that describes a single-object software system. The executable image, MAIN.EXE, is the target of building the system. The executable image comes from one object file, MAIN.OBJ. The object file is generated from one file of source code, MAIN.PAS. Figure 1–2 shows the relationship between the files.

**Figure 1–2: A Single-Object Software System**




MMS builds its own internal dependency tree. By default, it builds the first target in the description. It finds the source for the first target and then the source of the first target's source using built-in rules to build the software system. A built-in rule is an action to build a target with a particular extension from a source with a particular file type. A software system must follow built-in file-naming conventions for built-in rules to work.

MMS builds the system from the bottom up. First it builds the targets at the bottom of the dependency tree, then the targets that use those sources, then the targets that use those sources, and so on, until the primary target is built.

## 1.2.2 Building a Software System

This section explains how to build a single-object software system. Your default directory contains the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.PAS;1              24-JUL-1987 16:10
SYSTEM1.MMS;1           24-JUL-1987 16:11

Total of 2 files.
```

You can invoke MMS and give it the name of the description file as follows:

```
$ MMS/DESCRIPTION=SYSTEM1
```

MMS assumes that the file extension is .MMS. It finds the first target in the description and creates an internal picture of the system or dependency tree. MMS uses the built-in rules, based on the file extensions, to build the system. The built-in rule for building .OBJ files from .PAS files is as follows:

```
PASCAL /NOLIST/OBJECT=MAIN MAIN.PAS
```

The built-in rule for building .EXE files from .OBJ files is as follows:

```
LINK /TRACE/NOMAP/EXEC=MAIN MAIN.OBJ
```

After MMS has built your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;1              24-JUL-1987 16:13
MAIN.OBJ;1              24-JUL-1987 16:11
MAIN.PAS;1              24-JUL-1987 16:10
SYSTEM1.MMS;1           24-JUL-1987 16:11

Total of 4 files.
```

## 1.2.3 Rebuilding a Single-Object System

MMS performs efficient system rebuilding. To rebuild a system, invoke MMS as you would for an initial build. MMS checks the system from the bottom up to see if it is complete and up to date. If any part of the system is missing or any target is older than its sources, the system is recreated. For example, if you edit your source code file, MAIN.PAS, it would be newer than its object file, MAIN.OBJ.

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;1          24-JUL-1987 16:13
MAIN.OBJ;1          24-JUL-1987 16:11
MAIN.PAS;2          24-JUL-1987 16:17
SYSTEM1.MMS;1       24-JUL-1987 16:11

Total of 4 files.
```

When you invoke MMS to rebuild the system, it finds the first target in the description, finds the sources for that target and their sources, then compiles the source code file, MAIN.PAS, because it is newer than its target, MAIN.OBJ. MMS then relinks the object file, MAIN.OBJ, because it is now newer than its target, MAIN.EXE. After MMS rebuilds your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;2          24-JUL-1987 16:19
MAIN.EXE;1          24-JUL-1987 16:13
MAIN.OBJ;2          24-JUL-1987 16:18
MAIN.OBJ;1          24-JUL-1987 16:11
MAIN.PAS;2          24-JUL-1987 16:17
MAIN.PAS;1          24-JUL-1987 16:10
SYSTEM1.MMS;1       24-JUL-1987 16:11

Total of 7 files.
```

MMS works in the same way if one of the components of your system is missing. For example, you invoke MMS and it finds that MAIN.EXE does not exist.

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.OBJ;2          24-JUL-1987 16:18
MAIN.OBJ;1          24-JUL-1987 16:11
MAIN.PAS;2          24-JUL-1987 16:17
MAIN.PAS;1          24-JUL-1987 16:10
SYSTEM1.MMS;1       24-JUL-1987 16:11

Total of 5 files.
```

In this case, MMS builds its internal dependency tree, checks that the system is complete and up-to-date, finds that the executable image is missing, and recreates the executable image. After MMS rebuilds your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;1              24-JUL-1987 16:21
MAIN.OBJ;2              24-JUL-1987 16:18
MAIN.OBJ;1              24-JUL-1987 16:11
MAIN.PAS;2              24-JUL-1987 16:17
MAIN.PAS;1              24-JUL-1987 16:10
SYSTEM1.MMS;1           24-JUL-1987 16:11

Total of 6 files.
```

## 1.2.4 Building a Multi-Object Software System

This section demonstrates two new concepts in system building:

- A target can have more than one source.
- An action line can override a built-in rule in building or updating a target.

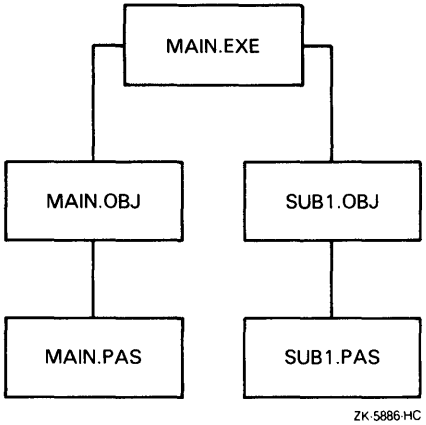
A description file called SYSTEM2.MMS has the following dependencies:

**Example 1–1: Description File for Multi-Object System**

```
MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ
    LINK MAIN.OBJ, SUB1.OBJ
MAIN.OBJ DEPENDS_ON MAIN.PAS
SUB1.OBJ DEPENDS_ON SUB1.PAS
```

The target, MAIN.EXE, has two sources: MAIN.OBJ and SUB1.OBJ. The second line of the description file is the action line, which explains how to build this target. The action line follows a target or source line and specifies how to use the source to create the target. You indent the action line and leave a blank line before the next dependency rule. The built-in rule for linking objects only links one object file; therefore, in this example, you need an action line to override this rule. Figure 1–3 shows the relationship between the files.

**Figure 1–3: A Multi-Object Software System**




Consider a directory that contains the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.PAS;1            24-JUL-1987 16:17
SUB1.PAS;1                24-JUL-1987 16:10
SYSTEM2.MMS;1             24-JUL-1987 16:12
 Total of 3 files.
```

To build your system, invoke MMS with the SYSTEM2.MMS description file as follows:

```
$ MMS/DESCRIPTION=SYSTEM2
```

In this example, MMS identifies the first target, MAIN.EXE, and its sources. It then identifies the sources for each of its object files. MMS builds from the bottom up by compiling the source code files, then creates the executable image from the action line you supplied. After MMS has built your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED
```

```
Directory DISK1:[BUILD]

MAIN.EXE;1              24-JUL-1987 16:21
MAIN.OBJ;1              24-JUL-1987 16:21
MAIN.PAS;1              24-JUL-1987 16:17
SUB1.OBJ;1                  24-JUL-1987 16:20
SUB1.PAS;1                  24-JUL-1987 16:10
SYSTEM2.MMS;1               24-JUL-1987 16:12
 Total of 6 files.
```

## 1.2.5 Rebuilding a Multi-Object System

MMS saves time in system rebuilding. It does only what it must to rebuild the system. To rebuild, invoke MMS as you would for an initial build. MMS checks the system from the bottom up to see if it is complete and up-to-date. If any part of the system is missing or older than its sources, the system is recreated. For example, if you edit your source code file, SUB1.PAS, it would be newer than its object file, SUB1.OBJ.

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;1              24-JUL-1987 16:21
MAIN.OBJ;1              24-JUL-1987 16:21
MAIN.PAS;1              24-JUL-1987 16:17
SUB1.PAS;2              24-JUL-1987 16:23
SUB1.PAS;1              24-JUL-1987 16:20
SUB1.OBJ;1              24-JUL-1987 16:20
SYSTEM2.MMS;1           24-JUL-1987 16:12

Total of 7 files.
```

When you invoke MMS to rebuild the system, it finds the first target in the description file, finds the sources for that target and their sources, then compiles the source code file, SUB1.PAS, because it is newer than its target, SUB1.OBJ. Because MAIN.OBJ is newer than MAIN.PAS, MMS does not compile MAIN.PAS. However, MAIN.EXE is now older than one of its sources, so MMS must relink the system using the action line. After MMS has rebuilt your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED
```

```
Directory DISK1:[BUILD]

MAIN.EXE;2              24-JUL-1987 16:27
MAIN.EXE;1              24-JUL-1987 16:21
MAIN.OBJ;1              24-JUL-1987 16:21
MAIN.PAS;1              24-JUL-1987 16:17
SUB1.PAS;2              24-JUL-1987 16:23
SUB1.PAS;1              24-JUL-1987 16:20
SUB1.OBJ;2              24-JUL-1987 16:26
SUB1.OBJ;1              24-JUL-1987 16:20
SYSTEM2.MMS;1           24-JUL-1987 16:12

Total of 9 files.
```

Whenever you invoke MMS, it checks that a software system is complete, that all components exist. You should not delete object files after using MMS because MMS would recompile all the source code files the next time it is invoked. If none of your targets or sources have changed since your last system build, and you invoke MMS, you will receive the following message from MMS:

```
$ MMS/DESCRIPTION=SYSTEM2
%MMS-I-GWKCURRNT, Target MAIN.EXE is already up to date.
```

MMS determines that nothing needs updating.

## 1.2.6 Building Systems with Multiple Programming Languages

The previously discussed software systems have used one programming language. It is not unusual for systems to use several programming languages. MMS easily handles software composed of multiple languages. There is a built-in rule for most VAX languages that chooses the correct compiler during system building and rebuilding.

A description file for a multilanguage system is the same as for a single language system. MMS uses the different file types to choose the correct compiler during the system build. The following sample description file, MULTI_LANG.MMS, uses comment lines denoted by the exclamation point (!) and, for readability, blank lines to separate source code files for each language.

### Example 1–2: Description File Using Multiple Language Compilers

```
!
!Main executable target, its objects and action line
!

MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ, SUB2.OBJ, SUB3.OBJ
     LINK MAIN, SUB1, SUB2, SUB3

!
!Source code dependencies
!

MAIN.OBJ DEPENDS_ON MAIN.PAS
SUB1.OBJ DEPENDS_ON SUB1.PAS

SUB2.OBJ DEPENDS_ON SUB2.FOR
SUB3.OBJ DEPENDS_ON SUB3.FOR
```

Your current directory contains the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.PAS;1           24-JUL-1987 16:17
MULTI_LANG.MMS;1     24-JUL-1987 16:19
SUB1.PAS;2           24-JUL-1987 16:23
SUB2.FOR;1           24-JUL-1987 16:15
SUB3.FOR;1           24-JUL-1987 16:18

Total of 5 files.
```

To build your system, invoke MMS with the MULTI_LANG.MMS description file as follows:

```
$ MMS/DESCRIPTION=MULTI_LANG
```

In this example, MMS identifies the first target, MAIN.EXE, and its sources. It then identifies the sources for each of its object files. MMS uses the PASCAL compiler to compile files with a .PAS file type and the FORTRAN compiler to compile files with the .FOR file type. After MMS has built your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED
```

```
Directory DISK1:[BUILD]

MAIN.EXE;1              24-JUL-1987 16:29
MAIN.OBJ;1              24-JUL-1987 16:29
MAIN.PAS;1              24-JUL-1987 16:17
MULTI_LANG.MMS;1        24-JUL-1987 16:19
SUB1.OBJ;1              24-JUL-1987 16:28
SUB1.PAS;2              24-JUL-1987 16:23
SUB2.FOR;1              24-JUL-1987 16:15
SUB2.OBJ;1              24-JUL-1987 16:27
SUB3.FOR;1              24-JUL-1987 16:18
SUB3.OBJ;1              24-JUL-1987 16:26

Total of 10 files.
```

If you deleted an object file but still had the executable file, and if you invoked MMS, MMS would recompile the source file and also relink the executable file, even though your executable file was still compatible with your source file. MMS works from the bottom up and propagates any change up the dependency tree.

However, if you deleted your source file and invoked MMS to rebuild the system, MMS would return a fatal error because it cannot recreate source code from object files or executable files.

## 1.2.7 Rebuilding a Multiple Programming Language System

If you have edited two source files in your current directory and want to rebuild your system, invoke MMS as you would for building the system. For example, your current directory might contain the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;1              24-JUL-1987 16:29
MAIN.OBJ;1              24-JUL-1987 16:29
MAIN.PAS;2              24-JUL-1987 16:30
MAIN.PAS;1              24-JUL-1987 16:17
MULTI_LANG.MMS;1        24-JUL-1987 16:19
SUB1.OBJ;1              24-JUL-1987 16:28
SUB1.PAS;2              24-JUL-1987 16:23
SUB2.FOR;1              24-JUL-1987 16:15
SUB2.OBJ;1              24-JUL-1987 16:27
SUB3.FOR;2              24-JUL-1987 16:31
SUB3.FOR;1              24-JUL-1987 16:18
SUB3.OBJ;1              24-JUL-1987 16:26

Total of 12 files.
```

To build your system, invoke MMS with the MULTI_LANG.MMS description file as follows:

```
$ MMS/DESCRIPTION=MULTI_LANG
```

This example shows that source code files, MAIN.PAS and SUB3.FOR, have been edited, so the source code is newer than the object file. To rebuild the system, MMS compiles only the source code that has been updated, and uses the correct language compiler in each case. It then links all the objects to recreate the executable image. After MMS rebuilds your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;2          24-JUL-1987 16:35
MAIN.EXE;1              24-JUL-1987 16:29
MAIN.OBJ;2          24-JUL-1987 16:35
MAIN.OBJ;1              24-JUL-1987 16:29
MAIN.PAS;2          24-JUL-1987 16:30
MAIN.PAS;1              24-JUL-1987 16:17
MULTI_LANG.MMS;1        24-JUL-1987 16:19
SUB1.OBJ;1              24-JUL-1987 16:28
SUB1.PAS;2              24-JUL-1987 16:23
SUB2.FOR;1              24-JUL-1987 16:15
SUB2.OBJ;1              24-JUL-1987 16:27
SUB3.FOR;2              24-JUL-1987 16:31
SUB3.FOR;1              24-JUL-1987 16:18
SUB3.OBJ;2          24-JUL-1987 16:34
SUB3.OBJ;1              24-JUL-1987 16:26

Total of 15 files.
```

## 1.2.8 Source Code with Included Files

Included files are frequently used in software development projects. Any set of variables or constant declarations in a software system can be kept in an included file. Individual programmers do not have to retype this information in their program files; they just include the common file. This ensures that common code does not change between developers. If the included file changes, all developers automatically have the new code when they use the common file. The only drawback to using an included file is that if one changes, all source code files that use the file must be recompiled. Fortunately, MMS handles this automatically.

The description file must specify that included files exist for the system build and must list the included files on the same line after the program code that uses them. The included files can be listed in any order.

Nested included files can cause some confusion, but MMS handles nested included files in the same way it handles nonnested included files. As long as the included files follow the source code file, MMS handles nested included files correctly.

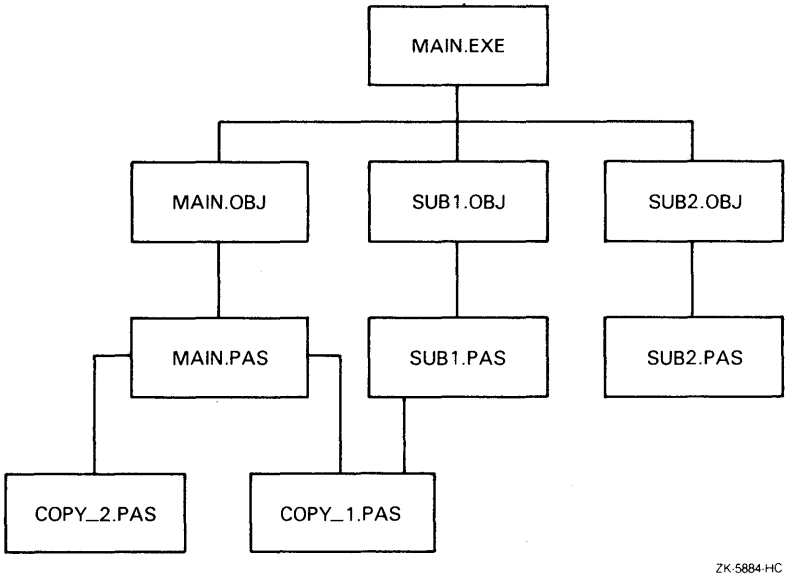
The following sample description file, INCLUDE.MMS, shows how to use included files:

### Example 1–3: Description File with Included Files

```
!
! Main executable target, its objects, and action line
!

MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ, SUB2.OBJ
    LINK MAIN, SUB1, SUB2

!
! Source code files with included files COPY_1.PAS and COPY_2.PAS
!

MAIN.OBJ DEPENDS_ON MAIN.PAS, COPY_1.PAS, COPY_2.PAS
SUB1.OBJ DEPENDS_ON SUB1.PAS, COPY_1.PAS
SUB2.OBJ DEPENDS_ON SUB2.PAS
```

MAIN.PAS is the source code file for the object file, MAIN.OBJ, and contains two included files, COPY_1.PAS and COPY_2.PAS. COPY_1.PAS is shared by another source code file, SUB1.PAS. Figure 1–4 shows the dependencies between the files.

**Figure 1–4: Included Files in a Software System**




During compilation, the PASCAL compiler handles the included files invisibly if they exist. If the included files do not exist, MMS returns with a fatal error and aborts the process. However, MMS has an /IGNORE qualifier, which allows you to control whether MMS aborts after a fatal error.

## 1.2.9 Building a System with Included Files

The ability of MMS to handle included files ensures accurate system building in a production environment. In a large system, it is hard to remember which compilation depends on which included file, and it is easy to forget to perform the compilations when an included file changes. When writing your MMS description file, you should inspect all your source code files for statements that include other files. You can use the DCL SEARCH command to search for these statements.

Consider a directory that contains the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

COPY_1.PAS;1          24-JUL-1987 16:36
COPY_2.PAS;1          24-JUL-1987 16:37
INCLUDE.MMS;1         24-JUL-1987 16:38
MAIN.PAS;1            24-JUL-1987 16:30
SUB1.PAS;1            24-JUL-1987 16:23
SUB2.PAS;1            24-JUL-1987 16:39

Total of 6 files.
```

To build your system, invoke MMS with the INCLUDE.MMS description file as follows:

```
$ MMS/DESCRIPTION=INCLUDE
```

MMS compiles each source code file and then links them to generate the executable image, MAIN.EXE. After MMS has built your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

COPY_1.PAS;1          24-JUL-1987 16:36
COPY_2.PAS;1          24-JUL-1987 16:37
INCLUDE.MMS;1         24-JUL-1987 16:38
MAIN.EXE;1            24-JUL-1987 16:48
MAIN.OBJ;1            24-JUL-1987 16:48
MAIN.PAS;1            24-JUL-1987 16:30
SUB1.OBJ;1            24-JUL-1987 16:47
SUB1.PAS;1            24-JUL-1987 16:23
SUB2.OBJ;1            24-JUL-1987 16:47
SUB2.PAS;1            24-JUL-1987 16:39

Total of 10 files.
```

## 1.2.10 Rebuilding a System with Included Files

If you have edited the included file COPY_1.PAS in your current directory and you want to rebuild your system, invoke MMS as you would for a system build.

For example, consider a directory that contains the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

COPY_1.PAS;2          24-JUL-1987 16:49
COPY_2.PAS;1          24-JUL-1987 16:37
INCLUDE.MMS;1         24-JUL-1987 16:38
MAIN.EXE;1            24-JUL-1987 16:48
MAIN.OBJ;1            24-JUL-1987 16:48
MAIN.PAS;1            24-JUL-1987 16:30
SUB1.OBJ;1            24-JUL-1987 16:45
SUB1.PAS;1            24-JUL-1987 16:23
SUB2.OBJ;1            24-JUL-1987 16:46
SUB2.PAS;1            24-JUL-1987 16:39

Total of 10 files.
```

To rebuild your system, invoke MMS with the INCLUDE.MMS description file as follows:

```
$ MMS/DESCRIPTION=INCLUDE
```

This example shows that the included file has been edited. COPY_1.PAS is newer than any file used to build MAIN.EXE. When you invoke MMS, both MAIN.PAS and SUB1.PAS that include COPY_1.PAS are recompiled and relinked. If you had edited COPY_2.PAS, and you recompiled your system, only MAIN.PAS would have been recompiled and relinked. After MMS rebuilds your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

COPY_1.PAS;2          24-JUL-1987 16:49
COPY_2.PAS;1          24-JUL-1987 16:37
INCLUDE.MMS;1         24-JUL-1987 16:38
MAIN.EXE;2            24-JUL-1987 16:51
MAIN.OBJ;2            24-JUL-1987 16:51
MAIN.OBJ;1            24-JUL-1987 16:48
MAIN.PAS;1            24-JUL-1987 16:30
SUB1.OBJ;2            24-JUL-1987 16:50
SUB1.OBJ;1            24-JUL-1987 16:45
SUB1.PAS;1            24-JUL-1987 16:23
SUB2.OBJ;1            24-JUL-1987 16:46
SUB2.PAS;1            24-JUL-1987 16:39

Total of 12 files.
```

## 1.2.11 Systems with More Than One Executable Image

If a system has a number of executable images that use common object files, it is more efficient to build them from one description file. If any one executable image is especially complicated, it is advisable to place it in its own description file. Executable images that are not related in some significant way should not be in the same description file. However, MMS can handle this complex system build with a description file that has an overall target that includes each executable file. You can choose to build the entire system or selected executable images in the system.

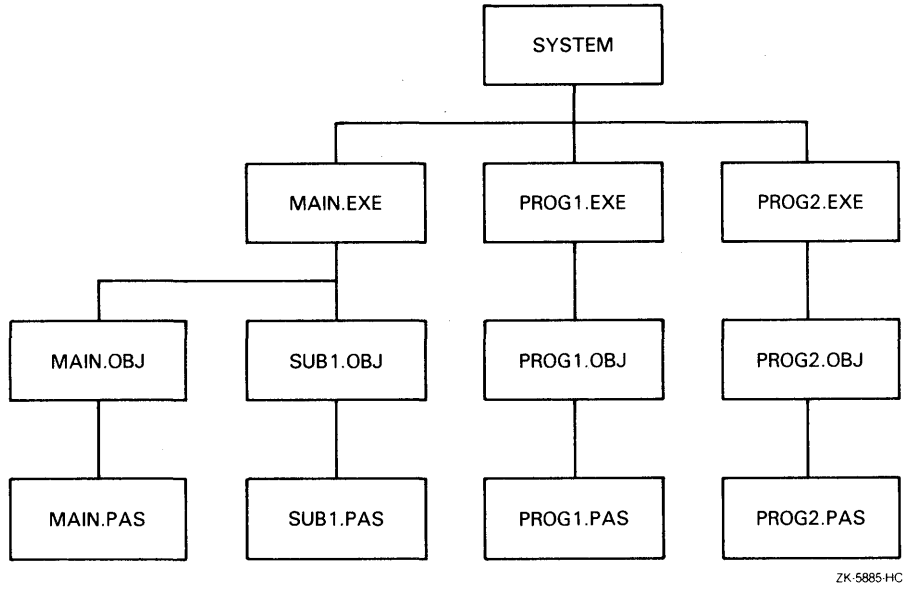
This type of description file is layered with the overall goal coming first, followed by the executable images, the object files, and the source code files. For example, the following sample description file, named MULTI_EXES.MMS, can build an overall target, or one executable image:

### Example 1–4: Description File Using Multiple Targets

```
!
! Overall system target
!

SYSTEM DEPENDS_ON MAIN.EXE, PROG1.EXE, PROG2.EXE
       ! (no special action for system target)

!
! The executable images and their object files
!

MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ
    LINK MAIN.OBJ, SUB1.OBJ

PROG1.EXE DEPENDS_ON PROG1.OBJ
    LINK PROG1.OBJ

PROG2.EXE DEPENDS_ON PROG2.OBJ
    LINK PROG2.OBJ

!
! The object files and their sources
!

MAIN.OBJ DEPENDS_ON MAIN.PAS
SUB1.OBJ DEPENDS_ON SUB1.PAS
PROG1.OBJ DEPENDS_ON PROG1.PAS
PROG2.OBJ DEPENDS_ON PROG2.PAS
```

This description file contains a target, SYSTEM, which includes all the executable images and a null action because SYSTEM cannot actually be created. All the object files and source code files have the same dependencies as in the previous examples. Figure 1–5 shows the dependencies between the files.

**Figure 1–5: A System with More Than One Executable Image**




## 1.2.12 Building a System with More Than One Executable Image

MMS handles multiple executable files with accuracy during the system build. In your description file, the system target is listed first and MMS treats it as the goal of the system build. Each executable image is built in the process of building the SYSTEM. After the executable images are built, MMS takes any specified action to update the SYSTEM target itself, but no action is required.

Consider a directory that contains the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.PAS;1              24-JUL-1987 16:30
MULTI_EXES.MMS;1        24-JUL-1987 16:34
PROG1.PAS;1             24-JUL-1987 16:31
PROG2.PAS;1             24-JUL-1987 16:33
SUB1.PAS;1              24-JUL-1987 16:23

Total of 5 files.
```

To build your system, invoke MMS with the MULTI_EXES.MMS description file as follows:

```
$ MMS/DESCRIPTION=MULTI_EXES
```

MMS attempts to build the first target, that is, all the executable images. It compiles all the source code files and links them to build the executable images. The final action of building SYSTEM is null and the system is complete at this time. After MMS builds your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;1              24-JUL-1987 16:55
MAIN.OBJ;1              24-JUL-1987 16:55
MAIN.PAS;1              24-JUL-1987 16:30
MULTI_EXES.MMS;1        24-JUL-1987 16:34
PROG1.EXE;1             24-JUL-1987 16:54
PROG1.OBJ;1             24-JUL-1987 16:54
PROG1.PAS;1             24-JUL-1987 16:31
PROG2.EXE;1             24-JUL-1987 16:53
PROG2.OBJ;1             24-JUL-1987 16:53
PROG2.PAS;1             24-JUL-1987 16:33
SUB1.OBJ;1              24-JUL-1987 16:52
SUB1.PAS;1              24-JUL-1987 16:23

Total of 12 files.
```

You can also invoke MMS to build a specific target or executable image. For example:

```
$ MMS/DESCRIPTION=MULTI_EXES PROG1.EXE, PROG2.EXE
```

This command line invokes MMS with a "target specifier." A target specifier is a parameter that directs MMS to build a specific target or targets. A target specifier overrules the MMS default of building the first target in the description file. Only the specified targets are built.

## 1.2.13 Rebuilding a System with Several Executable Images

If you have deleted one of the executable files, PROG2.EXE, in your current directory and you want to rebuild your system, invoke MMS as you would for a system build.

```
$ MMS/DESCRIPTION=MULTI_EXES
```

MMS links PROG2.OBJ to produce the executable image. MMS performs only the link necessary to complete the system. After MMS has rebuilt your system, the files in your directory are as follows:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.EXE;1              24-JUL-1987 16:55
MAIN.OBJ;1              24-JUL-1987 16:55
MAIN.PAS;1              24-JUL-1987 16:30
MULTI_EXES.MMS;1        24-JUL-1987 16:34
PROG1.EXE;1             24-JUL-1987 16:54
PROG1.OBJ;1             24-JUL-1987 16:54
PROG1.PAS;1             24-JUL-1987 16:31
PROG2.EXE;1             24-JUL-1987 16:57
PROG2.OBJ;1             24-JUL-1987 16:53
PROG2.PAS;1             24-JUL-1987 16:33
SUB1.OBJ;1              24-JUL-1987 16:52
SUB1.PAS;1              24-JUL-1987 16:23

Total of 12 files.
```

## 1.2.14 Building Systems with Object Libraries

This section assumes that you have some knowledge of VMS libraries. Object libraries are useful for faster compiling and linking during debugging sessions. MMS handles libraries easily and takes care of library creation, module insertion, and updating of libraries during system rebuilds.

To put an object file into a library, you must have the library file name, an object file name, and the library module name. The library file name usually has the .OLB extension; the object file name usually has the .OBJ extension.

The library module name and the object file name are often confused. The object file name is a VMS file name of the object file. The library module name is governed by the TITLE, MODULE, PROGRAM, or SUBROUTINE name in the source code. These two names can be the same.

MMS looks for the library file name just after the target as the main source. The library module name is the first after the parentheses and the object file name is immediately after the equal sign. For example, examine the following sample description file called MAIN_LIB.MMS:

### Example 1–5: Description File Using Object Libraries

```
!
! Main executable target, its objects, and action line
!

MAIN.EXE DEPENDS_ON     MAIN.OBJ, -
                        MAIN_LIB.OLB(OPTIM=OPTIM.OBJ), -
                        MAIN_LIB.OLB(GET_RECORD=GETREC.OBJ), -
                        MAIN_LIB.OLB(PUT_RECORD=PUTREC.OBJ)
      LINK MAIN.OBJ, MAIN_LIB/LIB

!
! Program source code files
!

MAIN.OBJ DEPENDS_ON MAIN.FOR
OPTIM.OBJ DEPENDS_ON OPTIM.FOR
GETREC.OBJ DEPENDS_ON GETREC.FOR
PUTREC.OBJ DEPENDS_ON PUTREC.FOR
```

In this description file the executable image, MAIN.EXE, depends on one object file and three library entries. Note that this file contains two files where the object name and the module name are different, and one where they are the same. You can read the library source line as "Use the library name MAIN_LIB, and the object module in it named OPTIM, which comes from the object file named OPTIM.OBJ."

MMS does a large amount of work when using libraries. It compiles each source code file, creates the library if necessary, and inserts each object module in the library. Library files also have built-in rules.

Consider a current directory that contains the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

GETREC.FOR;1          28-JUL-1987 10:43
MAIN.FOR;1            28-JUL-1987 10:43
MAIN_LIB.MMS;4        28-JUL-1987 10:43
OPTIM.FOR;2           28-JUL-1987 10:43
PUTREC.FOR;1          28-JUL-1987 10:43

Total of 5 files.
```

To build your system, invoke MMS with the MAIN_LIB.MMS description file as follows:

```
$ MMS/DESCRIPTION=MAIN_LIB
```

To build this system, MMS performs the following tasks:

- MMS compiles each library entry.
- MMS checks for the existence of a library and creates the library if one does not exist.
- MMS then inserts each entry into the library.

The directory includes all parts of the system including the library. Your current directory now contains the following files:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

GETREC.FOR;1          28-JUL-1987 10:43
GETREC.OBJ;1          28-JUL-1987 10:44
MAIN.EXE;1            28-JUL-1987 10:45
MAIN.FOR;1            28-JUL-1987 10:43
MAIN.OBJ;1            28-JUL-1987 10:44
MAIN_LIB.MMS;4        28-JUL-1987 10:43
MAIN_LIB.OLB;1        28-JUL-1987 10:44
OPTIM.FOR;2           28-JUL-1987 10:43
OPTIM.OBJ;1           28-JUL-1987 10:44
PUTREC.FOR;1          28-JUL-1987 10:43
PUTREC.OBJ;1          28-JUL-1987 10:44

Total of 11 files.
```

## 1.2.15 Rebuilding a System with Object Libraries

If you have edited the file GETREC.FOR and you want to rebuild your system, type the following command line:

```
$ MMS/DESCRIPTION=MAIN_LIB
```

MMS compiles the source code file GETREC.FOR, checks for the existence of the library, and replaces the old library module with the new one.

## 1.2.16 Using the Description File to Maintain Your System

The description file in Example 1–6 shows how you can use an MMS description file to define macros (a macro gives a name to a character string). It also describes dependencies to perform the following functions:

- Print out the source code files
- Clean up the directory and show the results
- Check the portability of the system
- Generate, print, and delete listings complete with cross references

The comments in the description file in Example 1–6 explain the dependency rules.

### Example 1–6: Description File for Maintaining Your System

```
! The macros are defined:

OBJECTS = MOD1.OBJ, MOD2.OBJ, MOD3.OBJ

SOURCES = DEFS1.H, DEFS2.H, MOD1.C, MOD2.C, MOD3.C

CSOURCES = MOD1.C, MOD2.C, MOD3.C

! The following dependency rules describe target-source
! relationships for the system.  Built-in rules are
! used so that not all the dependencies need to be
! stated explicitly.

PROG.EXE DEPENDS_ON $(OBJECTS)
     LINK/EXEC=PROG $(OBJECTS)
     COPY PROG.EXE PRINTNEW.    ! Use PRINTNEW. as a time stamp

MOD2.OBJ DEPENDS_ON DEFS1.H

MOD2.OBJ, MOD3.OBJ DEPENDS_ON DEFS2.H

! The following rule prints out all the source files:

PRINT :
     PRINT $(SOURCES)

! The following rule cleans up the directory and
! shows the result; the Silent prefix (@) suppresses
! the display of the action lines on your terminal screen

CLEANUP :
     @ DELETE *.BAK;*, *.OBJ;*
     @ DIRECTORY/DATE/SIZE
```

(continued on next page)

**Example 1–6 (Cont.): Description File for Maintaining Your System**

```
! The following rule prints the sources
! that have changed since last build:

PRINTNEW. : $(SOURCES)
     PRINT $(MMS$CHANGED_LIST)

! The following rule checks the portability of the system:

PORTABLE :
     CC /STANDARD=PORTABLE/NOOBJECT/NOLIST $(CSOURCES)

! The following rule generates listings (complete with cross-
! references and symbols), prints them, then deletes them:

CROSSREF :
     CC /CROSS_REFERENCE/LIST/NOMACHINE_CODE/SHOW:SYM $(CSOURCES)
     PRINT/DELETE *.LIS
```

With this sample description file, you can perform several system management tasks by specifying different targets on the MMS command line.

To clean up the directory, type the following command:

```
$ MMS CLEANUP
```

To print all the source files, type the following command:

```
$ MMS PRINT
```

To print all recently changed source files, type the following command:

```
$ MMS PRINTNEW.
```

To get a complete set of hardcopy listings with cross-references and the symbol table, type the following command:

```
$ MMS CROSSREF
```

To check the portability of source code, type the following command:

```
$ MMS PORTABLE
```

As the need for other system management tasks arises, you can add appropriate dependency rules to the description file.

## 1.3 MMS Command Format

The format for the MMS command is as follows:

```
$ MMS [/qualifier . . . ] [target, . . . ]
```

**/qualifier**
An MMS qualifier.

**target**
The name of a target, which can be either a VMS file specification or a logical name.

## 1.4 Qualifiers

The qualifiers for the MMS command are as follows:

| Command Qualifiers | Defaults |
|---|---|
| /[NO]ACTION | /ACTION |
| /[NO]CHECK_STATUS | /NOCHECK_STATUS |
| /[NO]CMS | /NOCMS |
| /[NO]DESCRIPTION=filespec . . . | /DESCRIPTION=DESCRIP.MMS |
| /FROM_SOURCES | None |
| /HELP | None |
| /IDENTIFICATION | None |
| /[NO]IGNORE= { WARNING / ERROR / FATAL } | /NOIGNORE |
| /[NO]LOG | /NOLOG |
| /MACRO= { filespec / "macro" . . . } | None |
| /OUTPUT=filespec | /OUTPUT=SYS$OUTPUT |
| /[NO]OVERRIDE | /NOOVERRIDE |
| /[NO]REVISE_DATE | /NOREVISE_DATE |
| /[NO]RULES[=filespec] | /RULES |
| /[NO]SCA_LIBRARY[=library-name] | /NOSCA_LIBRARY |
| /[NO]SKIP_INTERMEDIATE | /NOSKIP_INTERMEDIATE |
| /[NO]VERIFY | /VERIFY |

# Chapter 2

# The MMS Description File

Chapter 1 showed you how to use MMS to build a variety of software systems. There was a progression from simple systems to complex systems with many files of source code, multiple language compilers, and executable images. MMS built the systems using its built-in features and information obtained from the description file. This chapter describes the elements of the description file and how they work together to build a system.

## 2.1 Overview

The description file instructs MMS how to build your system and explains the relationships among the various components of your system. You can create and modify the description file with any editor, and once you have created the description file, you can invoke it on the MMS command line.

The description file builds your system using some or all of the following elements:

- Action lines
- Comment Lines
- Built-in rules
- User-defined rules
- Directives

## 2.2 Using Dependency Rules

A description file always contains **dependency rules**. Dependency rules indicate how files depend on, or are affected by, other files and specify the actions MMS takes to build or update your system.

A dependency rule consists of targets, optional sources, an optional action line, and an optional comment for each target and source line. The format for dependency rules is as follows:

```
target...: [source...] [!comment]
        [action line...]    [!comment]
```

A target or source can be a VMS file specification or a **mnemonic name** (Section 2.2.3 describes mnemonic names). If you are using DECnet, the file specifications for the target and source can include node information.

A comment is usually a string of text, introduced by an exclamation point (!), that documents the description file. You can continue a comment line onto the next line with the hyphen or backslash. MMS considers any text on the next line following the continuation character as part of the comment line. An action line is a command-language command that MMS uses to update the target. You can specify any number of action lines for a target. An action line is positioned below the corresponding target or source line and must be indented by at least one space or tab. MMS interprets all indented lines as action lines and associates them with the most recently specified target or source line. If you omit the action line, MMS uses built-in rules to update the target if a built-in rule exists. (See Section 2.3 for an explanation of built-in rules).

You begin a target or source line in column 1 of the line and use the colon (:) or the keyword **DEPENDS_ON** to separate the target from the source. If you use a colon to separate the target from the source, insert at least one space or tab on either side of the colon, so that MMS does not interpret the colon as part of a VMS file specification.

To improve the readability of description files, separate dependency rules from each other with one or more blank lines. Do not use blank lines between the action lines of a single dependency rule, because a blank line signals the end of the dependency rule.

By default, MMS expects to find the source and target files in the current default directory unless you specify other directories in your file specification.

Any line in a description file can be continued onto the next line with a hyphen (-). This practice makes the description file easier to read when a dependency rule is too long to fit on one line. For example:

```
❶ TESTS.OBJ DEPENDS_ON -
❷      TEST1.BAS, - ! Source modules for TESTS.OBJ- ❸
       TEST2.BAS, -
       TEST3.BAS, -
       TEST4.BAS, -
       TEST5.BAS
            BASIC/OBJECT=TESTS TEST1+TEST2+TEST3+TEST4+TEST5
```

❶ The hyphen means that the next line is treated as part of the current line.

❷ The second through fifth lines are continuations of the target or source line.

❸ A comment can appear after a continuation character without affecting the processing of the description file.

**NOTE**

When a hyphen appears as the last character on a line, MMS interprets the hyphen as a continuation character, even if the hyphen is part of a comment.

## 2.2.1 Source and Target Files

If you specify an action line but omit the source from a dependency rule, MMS executes the action line only if the target does not exist in the specified directory. For example, consider the following dependency rule:

```
[SYSTEM1]TESTS.OBJ :
    PASCAL/DEBUG [SYSTEM1]TESTS.PAS
```

In this example, MMS executes the PASCAL command only if TESTS.OBJ does not exist in the directory [SYSTEM1].

As MMS checks the revision dates and times of targets and sources, it builds a list of times, which it uses in deciding when a target needs to be updated. If there is no file associated with a target or source (for example, if the target does not exist), MMS records a revision time for it that is older than the times of all other existing targets and sources: 17-NOV-1858 00:00:00.0. (This is the oldest time used by VMS.) All targets and sources that are not existing files are assigned this revision time.

## 2.2.2 Specifying Multiple Targets and Sources

A description file can contain many dependency rules; however, MMS builds only one target. You can specify several targets on the MMS command line, but such a command is executed as a separate invocation of MMS for each target with the specified set of qualifiers.

To specify multiple targets and sources, you must separate them with commas, spaces, or a combination of both. MMS expands the specification of multiple targets into separate dependencies before it executes the action lines. For example, consider the following dependency rule in a description file:

```
KERNEL.OBJ, DRIVER.OBJ DEPENDS_ON COMMON.DEF
```

In this example, MMS uses built-in rules to determine what action is needed to update KERNEL.OBJ and DRIVER.OBJ and expands the previous rule as follows:

```
KERNEL.OBJ DEPENDS_ON KERNEL.C, COMMON.DEF
     CC KERNEL.C

DRIVER.OBJ DEPENDS_ON DRIVER.C, COMMON.DEF
     CC DRIVER.C
```

MMS determines from the built-in rules that KERNEL.C and DRIVER.C, which are not specified as targets in this dependency rule, are sources for KERNEL.OBJ and DRIVER.OBJ, respectively. The original dependency rule expands to two dependency rules, resulting in two compilations if both targets need updating.

Sometimes, if an action line is executed twice, the results may not be what you intended, as in the following example:

```
A.EXE : A.OBJ, A.LIS
     LINK A.OBJ

A.OBJ, A.LIS : A.BAS
     BASIC/LIST A.BAS
```

MMS expands the second rule to the following two dependency rules:

```
A.OBJ : A.BAS
     BASIC/LIST A.BAS

A.LIS : A.BAS
     BASIC/LIST A.BAS
```

Because the second dependency in the description file expands to two dependency rules, each with a separate action, MMS executes the command BASIC/LIST A.BAS twice and produces two .OBJ files and two .LIS files. See Section 3.11.2 for a detailed discussion on avoiding this problem in your description file.

Occasionally MMS executes an action when you do not expect the source to be newer than the target. This situation can result from one of the following conditions:

- If sources are being stored in a library to which more than one person has access, someone else may replace that source in the library after you have invoked MMS but before MMS has checked the source's revision time. Therefore, when MMS checks the time, a source newer than the

corresponding target may exist in the library. MMS would execute an action to update the target.

- If the sources and targets in your description file do not reside on the same node of a network, the clocks on the nodes may not be synchronized and a source may have a revision time that is later than the target. Also, in VAXcluster environment, clocks on different nodes of the cluster may not be synchronized.

### 2.2.3 Using Mnemonic Names for Targets and Sources

You can use a **mnemonic name** for a source or a target but you must supply the action lines that update the target. A mnemonic name is a name that identifies the purpose of a sequence of related actions. MMS relies on the source and target file types to apply built-in rules. Section 2.3 describes how MMS uses built-in rules.

You can use a mnemonic name to represent a source only if it is also a target in a dependency rule in your description file. If MMS encounters a name for which it cannot find a matching file in the specified directory, it assumes that the name is a mnemonic name.

Mnemonic names are useful in several cases, for example:

- To update more than one file
- To group a variety of related actions under a name that identifies the purpose of the whole sequence
- To give a name to a common action or sequence of actions in building a system

The first of these cases is probably the most important, because by default MMS builds only one target. If you need to update several targets, you can make them sources in a dependency rule using a mnemonic name for the target as follows:

```
NEW_SYSTEM DEPENDS_ON A.EXE, B.EXE
     ! no action needed

A.EXE DEPENDS_ON A.OBJ
     LINK A.OBJ

B.EXE DEPENDS_ON B.OBJ
     LINK B.OBJ
```

The target NEW_SYSTEM is considered updated when MMS executes the action line or lines that follow it. In this case, both A.EXE and B.EXE are updated, if necessary.

The following example shows the use of mnemonic names as both targets and sources. MMS updates the target, ALL, by updating the two sources, PROG.EXE and PRINT, which are themselves targets in subsequent dependency rules.

```
ALL DEPENDS_ON PROG.EXE, PRINT
    ! system completely built and the sources printed

PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
    LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ

PRINT DEPENDS_ON MOD1.C, MOD2.C, MOD3.C, DEFS1.H, DEFS2.H
    ! Print the source files
    PRINT MOD1.C, MOD2.C, MOD3.C, DEFS1.H, DEFS2.H
```

## 2.2.4 Specifying the Target on the Command Line

By default, MMS updates the first target specified in the description file. You can force MMS to update a target other than the first one by explicitly including the target name on the MMS command line. Consider the following description file:

```
TEST.OBJ DEPENDS_ON A.OBJ B.OBJ
    LINK/EXE=TEST A,B

A.OBJ DEPENDS_ON A.FOR
B.OBJ DEPENDS_ON  B.FOR
PRINT DEPENDS_ON
   PRINT A.FOR, B.FOR
```

If you specify MMS PRINT on the command line, MMS searches the description file for the dependency rule associated with PRINT, the specified target. MMS tries to update the target PRINT rather than TEST.OBJ, the default. If PRINT is up-to-date, no action takes place.

MMS updates all sources and their dependencies before updating the main target. MMS checks all sources before it updates a target, because sources may themselves be targets with sources in other dependency rules.

## 2.2.5 Hierarchy of Rule Application

MMS has a hierarchy of rule application:

- If an action line exists in a description file, the action line takes precedence over all built-in rules and user-defined rules.
- If a user-defined rule exists in a description file, the user-defined rule takes precedence over a built-in rule.

- A built-in rule is executed only if no action line or user-defined rule exists in the description file.
- If there is no action line, built-in rule, or user-defined rule for updating a target, then MMS issues a fatal-error message.

## 2.3 Using Built-In Rules

When writing a description file, you can explicitly state dependencies and actions, or you can abbreviate them by taking advantage of **built-in rules**, which MMS uses to update targets. A built-in rule is MMS' default method for updating a target with a particular file type from a source with a particular file type.

Built-in rules are made up of default macros, special macros, and string variables. A complete list of the MMS built-in rules is in Table C–6. A file copy of the built-in rules resides in [SYSHLP.EXAMPLES.MMS].

MMS uses its built-in rules when you omit the action line from a dependency rule. If the dependency rule has an action line but no source, then MMS uses the action line.

MMS knows how to build a software system from looking at file types and relating them to its built-in rules. Built-in rules are fixed and go into effect when you invoke MMS. They cannot be changed, but you can override them with user-defined rules. Built-in rules also explain why you must follow standard file naming practices. For example, a PASCAL program must have a .PAS extension. If your PASCAL program does not have the .PAS extension, MMS does not know it is a PASCAL program.

Built-in rules consist of the file extension of the source, the file extension of the target, and the action to update the target using the source. The actions in built-in rules use macros extensively. For example, consider the following:

**Example 2–1: A Built-In Rule**

```
❶    ❷
.PAS.OBJ
❸    $(PASCAL) $(PFLAGS) $(MMS$SOURCE)
```

❶ .PAS is the source file type.

❷ .OBJ is the target file type.

❸ $(PASCAL) $(PFLAGS) $(MMS$SOURCE) is the action line to update the target.

MMS attempts to use its built-in rules only when you omit the action line from a dependency rule. For example, MMS has a built-in rule that instructs it to use .FOR files when updating .OBJ files and to produce the .OBJ files by invoking the FORTRAN compiler. In writing the description file, you can state this relationship as follows:

```
MOD3.OBJ DEPENDS_ON MOD3.FOR
    FORTRAN MOD3.FOR
```

You can rely on MMS built-in rules by omitting the action line as follows:

```
MOD3.OBJ DEPENDS_ON MOD3.FOR
```

MMS uses its built-in rule to invoke the FORTRAN compiler and build MOD3.OBJ from MOD3.FOR.

If you omit the source, MMS can still use built-in rules to locate it because MMS knows about implied dependencies among files with the same name but different file types. MMS uses its suffixes precedence list to determine which file type (source) would result in the target file type. In the previous example, because the target's file name is MOD3, MMS assumes that the source's file name is also MOD3.

MMS also knows that .OBJ files depend on .FOR files with the same file name so you can abbreviate the previous dependency rule even further as follows:

```
MOD3.OBJ :
```

MMS automatically looks for MOD3.FOR and uses it to build MOD3.OBJ because MMS knows that .OBJ files depend on .FOR files with the same file name.

However, consider the following line:

```
MOD3.OBJ DEPENDS_ON MOD2.OBJ
```

If you omit the action line, MMS does not know how to build the target, and you receive an error message.

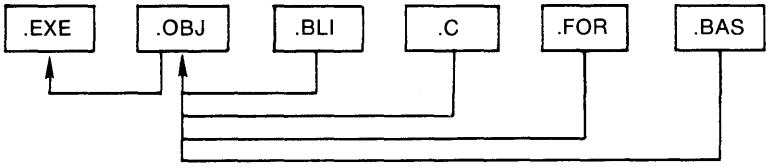
## 2.3.1 The Suffix Precedence List

MMS checks its **suffixes precedence list** to determine the file type of the source and then uses the built-in rules to determine how the various types of files can be generated from the known rules. Consider the following suffixes precedence list:

```
.EXE .OBJ .BLI .C .FOR .BAS
```

According to this list, .EXE files have precedence over .OBJ files, which have precedence over .BLI files, which have precedence over .C files, and so on. The relationship between the suffixes list and the known rules can be represented as follows:

**Figure 2–1: Relationship Between Suffixes**




ZK-1664-84

The arrows in this figure indicate built-in rules. In this figure, a known rule specifies how an .EXE file is made from an .OBJ file. Similarly, the built-in rules direct MMS how to make an .OBJ file from a .BLI file, a .C file, a .FOR file, and a .BAS file. Because .BLI precedes .C in the suffixes list, .BLI files have priority over .C files as a way to build .OBJ files. (The suffixes precedence list is contained in Table C–4; you can alter the order of the suffixes precedence list, as described in Section 2.8.4.)

The figure also shows that .OBJ files can be built from .BLI, .C, .FOR, and .BAS files. If MMS is trying to build MOD3.OBJ, it looks first for a source named MOD3.BLI. If such a source exists in the specified directory, MMS applies the known rule and creates MOD3.OBJ; if it finds no match for the file name and type, it continues looking in the specified directory for the same file name and the next file type from the suffixes list that can update the target. If MOD3.BLI

does not exist, MMS next looks for MOD3.C. If MOD3.C does not exist, the next possible source is MOD3.FOR, and so on.

If MMS finally matches MOD3.OBJ with MOD3.FOR and locates MOD3.FOR in your directory, it updates the target MOD3.OBJ from the source MOD3.FOR by using its built-in rule. This procedure explains why a dependency rule as brief as the following is complete:

```
MOD3.OBJ :
```

This rule equates to the full dependency rule as follows:

```
MOD3.OBJ DEPENDS_ON MOD3.FOR
    FORTRAN/OBJ=MOD3 MOD3.FOR
```

If, however, MMS fails to find a source from which to build the new target, it repeats the entire process by determining whether it can build one of the nonexistent sources.

If MMS exhausts all the possible file types without finding a way to build any of the sources, it issues an error message and aborts processing.

Once MMS locates the correct source for updating a target, it checks whether the source itself needs updating before using it to update the original target. To do this, MMS repeats the process of finding a file in the specified directory that matches the file name of the source and each file type in the suffixes list that can update the target type. MMS repeats this process every time it finds a source that could update the target so that all the sources are guaranteed to be up-to-date.

The following example shows a description file where dependencies are explicitly stated.

```
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
     LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ

MOD1.OBJ DEPENDS_ON MOD1.C
     CC MOD1.C

MOD2.OBJ DEPENDS_ON MOD2.C, DEFDIR:DEFS1.H, DEFDIR:DEFS2.H
     CC MOD2.C

MOD3.OBJ DEPENDS_ON MOD3.C, DEFDIR:DEFS2.H
     CC MOD3.C
```

The following description file of the same system takes advantage of MMS built-in rules:

**Example 2–2: A Description File Using Built-In Rules**

```
❶ PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
     LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ

❷

❸ MOD2.OBJ, MOD3.OBJ DEPENDS_ON DEFDIR:DEFS2.H

❹ MOD2.OBJ DEPENDS_ON DEFDIR:DEFS1.H
```

❶ The first dependency rule lists the object files and states that PROG.EXE is constructed by executing the DCL command LINK.

❷ The rule for building MOD1.OBJ need not be specified because a built-in rule directs MMS to build it from MOD1.C.

❸ The second dependency rule says that MOD2.OBJ and MOD3.OBJ depend on DEFS2.H, which is located in the directory defined by DEFDIR. Neither the .C file dependencies nor the actions taken to build the objects are stated.

❹ The third dependency rule says that MOD2.OBJ also depends on DEFDIR:DEFS1.H.

## 2.3.2 Default Macros

A **macro** is a name that represents a character string. MMS **default macros** can help you use MMS more efficiently because they define commonly used operations. MMS built-in rules are expressed in terms of default macros.

## 2.4 Defining Your Own Macros

In addition to providing built-in rules, MMS allows you to define your own rules. Defining your own rules may involve deleting, adding to, or replacing the built-in rules. Section 2.6 describes when and how to define new rules. MMS allows you to use three kinds of macros: default macros (VMS utilities or qualifiers), special macros (target or source file names), and user-defined macros. These macros can use other macros in their definition. The full list of default macros is in Table C–1 and the list of special macros is in Table C–3.

In MMS, macros contain the following information:

- The names of compilers, the linker and library utilities
- The default qualifiers for compiling, linking, and using the library utilities
- The file name of the target
- The list of sources for each target

The following table lists some default and special macros available with MMS.

| Macro | Description | Value |
|---|---|---|
| PASCAL | PASCAL compiler | PASCAL |
| PFLAGS | Default PASCAL qualifiers | /NOLIST/OBJECT=$(MMS$TARGET_NAME) |
| MMS$TARGET_NAME | Target file name | Depends on target or source line |
| MMS$SOURCE | First file name in source list | Depends on target or source line |
| MMS$SOURCE_LIST | All file names in source list | Depends on target or source line |

The default macros, PASCAL and PFLAGS, contain a fixed value and are stored in an internal MMS list. They are created when MMS is invoked. The special macros, MMS$TARGET_NAME, MMS$SOURCE, AND MMS$SOURCE_LIST, are also maintained by MMS but their value changes according to the source or target line MMS is evaluating.

If your description file reuses the same file name or if you have several action lines that invoke a compiler with the same set of qualifiers, you can define a macro to represent the file name or the list of qualifiers. You then can use the macro name throughout your description file. If you need to change the file name or qualifiers, you edit only the macro definition.

## 2.4.1 Formatting Macro Definitions

A macro definition has the following format:

```
name = string
```

The name of the macro can consist of any characters except a space, a tab, a carriage return, an equal sign, the sequence $( ), quotation marks, and control characters. A macro name can be any length. The macro string is the text that replaces the macro name when the macro is expanded. A macro string can consist of any character sequence. You can use a hyphen (-) as a continuation character to continue a macro string onto the next line of the description file.

You must begin a macro definition in column 1 of the line. You can place macro definitions anywhere in the description file, but placing all macro definitions at the beginning of the description file makes it easier to find and modify them.

After you have defined a macro, you can invoke it anywhere in the description file. To invoke a macro, simply specify a macro's name in the following format:

```
$(name)
```

The dollar sign and parentheses surrounding the macro name are required punctuation. MMS replaces the name (and the punctuation) with the equivalent text string when it processes your description file.

Note that you must define a macro before you can use it; otherwise, the macro's expanded value is the null string. To determine whether a macro has been defined, keep in mind the order in which MMS processes macro definitions (see Section 2.4.2).

## 2.4.2 Order of Processing Macros

When processing macros, MMS applies definitions in the following order:

1. Command line
2. Description file
3. Built-in
4. CLI symbol

Once MMS finds a definition for a macro, it does not search those locations farther down the list for more definitions.

You can define a macro only once in a description file. If MMS finds two or more definitions of the same macro, it issues an error message and uses the first definition in the file. To change a macro definition, you can redefine the macro with the /MACRO qualifier on the command line or you can replace the definition with the /OVERRIDE qualifier. (See the Command Dictionary for descriptions of these qualifiers.)

## 2.4.3 Invoking Macros

A macro string can also contain macro invocations that are expanded when the macro is defined. The macro invocations must denote macros that you have already defined in the description file. For example, suppose the following macro definitions appear in your description file:

```
BUILD1 = /DEBUG
BUILD2 = /LIST $(BUILD1)
```

The macro invocation $(BUILD1) is expanded to /DEBUG because BUILD1 has already been defined. If the positions of the macro definitions were reversed, BUILD1 would be expanded to the null string because it has not been previously defined and therefore cannot be expanded. In this case, MMS does not issue an error message.

MMS macros are not recursive. MMS expands a macro invocation only once. If during the expansion of a macro MMS encounters another macro invocation, the second invocation is not expanded.

The following description file, CPROG.MMS, defines two macros: FNAME, which expands to the string TESTS, and CCQUALS, which expands to the string /NOLIST.

**Example 2–3: Macro Definitions in a Description File**

```
FNAME = TESTS
CCQUALS = /NOLIST

$(FNAME).EXE : $(FNAME).OBJ, SYS$LIBRARY:STARLET.OLB
     LINK $(FNAME),-
          SYS$LIBRARY:STARLET.OLB/LIB

$(FNAME).OBJ : $(FNAME).C
     CC $(CCQUALS) $(FNAME).C
```

When MMS starts building the target (in this case, the .EXE file), it replaces every occurrence of FNAME with TESTS and the occurrence of CCQUALS with the string /NOLIST. As a result, MMS interprets the description file as the following:

```
TESTS.EXE : TESTS.OBJ, SYS$LIBRARY:STARLET.OLB
     LINK TESTS,-
          SYS$LIBRARY:STARLET.OLB/LIB

TESTS.OBJ : TESTS.C
     CC /NOLIST TESTS.C
```

## 2.4.4 Defining Macros on the Command Line

You can define macros on the MMS command line by using the /MACRO qualifier. /MACRO allows you to define new macros or to redefine macros you defined in the description file. When you redefine an existing macro with /MACRO, the new definition overrides the one in the description file. The format of the /MACRO qualifier is as follows:

```
/MACRO = {filespec | "macro"... }
```

The *filespec* is a VMS file specification or a logical name for a file that contains only macro definitions. The default file type is .MMS. The *"macro"* is a macro definition enclosed in quotation marks. Use the same format that you would use to define a macro in a description file: *name = string*. If you specify more than one macro, separate the macros with commas and enclose the list in parentheses. The /MACRO qualifier is described in detail in the Command Dictionary.

To build a new program called TEST1.EXE, you can use the same description file with which you built TESTS.EXE (as shown in the Example 2–3). You can redefine FNAME and override the macro definition in the description file as follows:

```
$ MMS/DESCRIPTION=CPROG/MACRO="FNAME=TEST1"
```

MMS then interprets the description file as follows:

```
TEST1.EXE : TEST1.OBJ, SYS$LIBRARY:STARLET.OLB
     LINK TEST1,-
          SYS$LIBRARY:STARLET.OLB/LIB

TEST1.OBJ : TEST1.C
     CC/NOLIST TEST1.C
```

The definition of the macro CCQUALS remains the same.

As indicated by the format for /MACRO, you can store macro definitions in a file from which MMS extracts them. Suppose that you want to redefine the macro FNAME in your description file and change the qualifiers to the CC command. First, you create a file to hold the macro definitions. For example, a macro definitions file might be called MACROS.MMS and contain the following:

```
FNAME = TEST1
CCQUALS = /LIST/DEBUG
```

Then you invoke MMS with the /MACRO qualifier and the name of the macro definitions file:

```
$ MMS/DESCRIPTION=CPROG/MACRO=MACROS
```

MMS interprets the previous description file as follows:

```
TEST1.EXE : TEST1.OBJ, SYS$LIBRARY:STARLET.OLB
     LINK TEST1, SYS$LIBRARY:STARLET.OLB/LIB

TEST1.OBJ : TEST1.C
     CC/LIST/DEBUG TEST1.C
```

You invoke a default macro in a dependency rule just as you would invoke a macro you have defined yourself. For example, if you want to compile a C program using the /NOLIST and /OBJECT qualifiers, you can instead invoke the default macro CFLAGS:

```
PROG.OBJ : PROG.C
     CC $(CFLAGS) PROG.C
```

MMS expands CFLAGS to its equivalent, /NOLIST/OBJECT, and assumes that the object file and the specified target have the same name. Since MMS has a built-in rule for generating .OBJ files from .C files, and since this rule invokes the default macro CFLAGS, you can get the same results with the simple dependency rule:

```
PROG.OBJ :
```

You can redefine a default macro so that you can use different qualifiers. The following example redefines CFLAGS:

```
CFLAGS = /LIST

PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
     LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ

MOD1.OBJ DEPENDS_ON

MOD2.OBJ DEPENDS_ON DEFDIR:DEFS1.H

MOD2.OBJ, MOD3.OBJ DEPENDS_ON DEFDIR:DEFS2.H
```

MMS interprets the description file as follows:

```
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
     LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ

MOD1.OBJ DEPENDS_ON MOD1.C
     CC/LIST MOD1.C

MOD2.OBJ DEPENDS_ON MOD2.C, DEFDIR:DEFS1.H, DEFDIR:DEFS2.H
     CC/LIST MOD2.C

MOD3.OBJ DEPENDS_ON MOD3.C, DEFDIR:DEFS2.H
     CC/LIST MOD3.C
```

If you later decide that you want the C source files to be compiled with the /DEBUG qualifier, you can redefine CFLAGS on the command line by typing:

```
$ MMS/MACRO="CFLAGS=/DEBUG/NOLIST"
```

MMS then interprets the description file as follows:

```
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
     LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ

MOD1.OBJ DEPENDS_ON MOD1.C
     CC/DEBUG/NOLIST MOD1.C

MOD2.OBJ DEPENDS_ON MOD2.C, DEFDIR:DEFS1.H, DEFDIR:DEFS2.H
     CC/DEBUG/NOLIST MOD2.C

MOD3.OBJ DEPENDS_ON MOD3.C, DEFDIR:DEFS2.H
     CC/DEBUG/NOLIST MOD3.C
```

## 2.5 Using Special Macros

MMS **special macros** expand to source or target names in the dependency currently being processed. You use them instead of target and source file specifications when you are writing general user-defined rules.

MMS provides nine special macros, which you can use in the following places in a description file:

- In user-defined rules

- In macro definitions
- In action lines
- In comments

You cannot redefine a special macro or use a special macro on a target or source line in a description file.

Table C–3 lists the MMS special macros and describes their functions. The table also lists a symbol that you can use as an abbreviation for each macro.

More information on the special macros that relate to the VAX DEC/Code Management System (CMS) can be found in Section 4.2.

**NOTE**

The strings $*, $%, and $? always denote special macros. If an action line contains these character combinations, the asterisk (*), percent sign (%), and question mark (?) are not interpreted as wildcard characters.

The following example shows how MMS defines a built-in rule using the MMS$SOURCE special macro:

```
.C.OBJ
     $(CC) $(CFLAGS) $(MMS$SOURCE)
```

CC and CFLAGS are default macros that invoke the C compiler with the /NOLIST and /OBJECT qualifiers. Consider the following dependency rule:

```
[ALDEN]MOD2.OBJ DEPENDS_ON [STANLEY]MOD2.C
```

MMS applies the built-in rule that updates an .OBJ file from a .C file, expanding the special macros in this rule as follows:

```
CC /NOLIST/OBJECT=[ALDEN]MOD2.OBJ [STANLEY]MOD2.C
```

You can use the MMS$CHANGED_LIST special macro to get listings of files that have changed since the last time the system was built. For example, consider the following description file:

```
PROG.EXE : PRINT.FLG, MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
     COPY NLA0: PRINT.FLG
     ! Make the revision date of PRINT.FLG more current
     PURGE PRINT.FLG
     LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ

PRINT.FLG : MOD1.C, MOD2.C, MOD3.C
     ! Print the sources that have changed
     PRINT $(MMS$CHANGED_LIST)
```

The COPY command in the first dependency rule ensures that PRINT.FLG has approximately the same revision time as PROG.EXE. Sources newer than PROG.EXE will also be newer than PRINT.FLG and will be printed only when they are more recent than the last linking of PROG.EXE. The MMS$CHANGED_LIST special macro expands to a list of all the source files that have changed, and each changed source listing is submitted to the print queue.

The following example shows how you could use MMS$TARGET and MMS$CHANGED_LIST in an action line to represent the current target and a list of the revised sources. Consider the following description file:

```
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
     LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
          ! Needed to update $(MMS$CHANGED_LIST) to make $(MMS$TARGET)
```

Your directory contains the following entries:

```
$ DIR/DATE=MODIFIED

Directory USER$:[MICHAELS]

MOD1.OBJ;2            2-DEC-1987 13:50
MOD2.OBJ;1            2-DEC-1987 09:22
MOD3.OBJ;2            2-DEC-1987 14:06
PROG.EXE;1            2-DEC-1987 11:47

Total of 4 files
$
```

Because MOD1.OBJ and MOD3.OBJ have changed since PROG.EXE was last linked, the following lines are displayed when you run MMS:

```
LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
! Needed to update MOD1.OBJ, MOD3.OBJ to make PROG.EXE
```

MMS$TARGET is expanded to the name of the target being updated, and MMS$CHANGED_LIST is expanded to a list of the revised sources.

## 2.6 Defining Your Own Rules

MMS has built-in rules that allow it to figure out unstated dependencies and to perform actions necessary to update targets. However, the list of built-in rules may not contain all the rules you need, or you may want to redefine existing rules. MMS provides you with the ability to include **user-defined rules** in a description file. Once you define a new rule, MMS uses the new rule every time it builds your system with that description file. The user-defined rule overrides the built-in rule.

## 2.6.1 Creating a User-Defined Rule

You create a user-defined rule by listing the source and target file types and writing an action to update the target. The file types of the source and target must be known to MMS through the suffixes precedence list. The user-defined rule can use multiple action lines that can consist of default macros, special macros, or constant strings.

The format of a user-defined rule is as follows:

```
.SRC.TAR                    [!comment]
     action line... [!comment]
```

.SRC is the file type of the source. .TAR is the file type of the target. The action line is a command-language command that MMS should execute to update a file of the target type from a file of the source type. You can specify as many action lines as necessary to update the target.

For example, the following description file, NEWLINK.MMS, contains a user-defined rule that redefines the default MMS rule for linking.

### Example 2–4: A Description File Using a User-Defined Rule

```
$ TYPE NEWLINK.MMS

  !
  ! User-defined rule
  !
  .OBJ.EXE
❶     $(LINK) $(LINKFLAGS) $(MMS$SOURCE_LIST)
  !
  ! Executable images and their sources
  !

❷  MAIN.EXE  DEPENDS_ON MAIN.OBJ, SUB1.OBJ

  !
  !  Object files and their sources
❸ MAIN.OBJ DEPENDS_ON MAIN.PAS
  SUB1.OBJ DEPENDS_ON SUB1.PAS
```

❶ This user-defined rule allows more than one object to link by changing the special macro MMS$SOURCE to MMS$SOURCE_LIST, which expands to a list of all the target's sources. The user-defined rule also uses the default macros, LINK and LINKFLAGS, for linking the object files.

❷ The executable target no longer needs an action line because the user-defined rule takes precedence.

❸ The built-in rule for compiling PASCAL source code files is used because there is no user-defined rule to override it.

You can use this user-defined rule for building a multi-object software system. Notice that you do not need another action line for the executable image target. The user-defined rule logically comes before any targets in your description file. The action line is listed on the line after the source and target pair and is indented at least one space or tab.

### 2.6.2 Using User-Defined Rules

To use the description file NEWLINK (shown in Example 2–4) to build your software system, you must have the following files in your current directory:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.PAS;1                3-JUL-1987 13:48
NEWLINK.MMS;2            14-JUL-1987 13:20
SUB1.PAS;10               3-JUL-1987 13:47

Total of 3 files.
$ MMS/DESCRIPTION=NEWLINK

❶ PASCAL /NOLIST/OBJECT=MAIN MAIN.PAS
  PASCAL /NOLIST/OBJECT=SUB1 SUB1.PAS
❷ LINK /TRACE/NOMAP/EXEC=MAIN MAIN.OBJ, SUB1.OBJ
```

❶ MMS output shows that MMS uses a built-in rule for compiling.

❷ MMS output shows that MMS uses the user-defined rule for linking.

## 2.7 Using Action Lines

You need action lines when you want to link more than one object file or you want to use different compilation options for each source code file. Built-in rules do not allow for these cases because built-in rules only handle one object file and compile each source code file with the same defaults.

You can supply an *action line* for any source or target line. Action lines override all built-in rules and user-defined rules in a description file. Action lines are made up of any combintion of default macros, special macros, and user-supplied strings. To keep your description file simple, use built-in rules as much as possible. One user-defined rule can apply to several different target and source lines, so it is still preferable to an action line. However, when the

action is so specific that it must be described for that individual case, then you must use action lines.

Consider the following description file, ACTION_LINES.MMS:

**Example 2–5: A Description File Using Action Lines**

```
$ TYPE ACTION_LINES.MMS
  !
  ! Executable image and its sources
  !

  MAIN.EXE  DEPENDS_ON MAIN.OBJ, SUB1.OBJ
❶         $(LINK) $(LINKFLAGS) $(MMS$SOURCE_LIST)

  !
  ! Object files and their sources
  !
  MAIN.OBJ DEPENDS_ON MAIN.PAS
❷          PASCAL /LIST MAIN.PAS

  SUB1.OBJ DEPENDS_ON SUB1.PAS
❸          $(PASCAL) /LIST /MACHINE_CODE $(MMS$SOURCE)
```

❶ This action line is composed entirely of macros and controls the way MMS links its object files. Earlier you used a user-defined rule for the same result. When a rule is used more than once, using a user-defined rule is the better approach. However, in this case, the rule is applied only once, so using the action line results in a simpler description file.

❷ This action line has no macros, only explicit strings.

❸ This action line has a combination of explicit strings and macros.

It is better to write a description file with consistent action lines than to mix the actions lines as in this example. This was done for the purpose of demonstrating the variety of ways that you can write an action line.

## 2.7.1 Multiple Action Lines

Sometimes a target requires a series of actions to update it. In that case, you can use more than one action line after a target or source line. Consider the description file, BALANCE.MMS:

**Example 2–6: A Description File Using Multiple Action Lines**

```
❶ RESULTS.DIF DEPENDS_ON ACCOUNTS.EXE, BENCHMARK.DAT
❷       RUN ACCOUNTS.EXE                    ! Runs ACCOUNTS
❸       DIFFERENCES/OUTPUT=RESULTS.DIF -
           ACCOUNTS.DAT, BENCHMARK.DAT
           ! Compares program output to master file
❹       TYPE RESULTS.DIF           ! Displays results of comparison

  ACCOUNTS.EXE DEPENDS_ON ACCOUNTS.OBJ
       LINK ACCOUNTS.OBJ          ! Links ACCOUNTS program
```

❶ If either ACCOUNTS.EXE or BENCHMARK.DAT is newer than RESULTS.DIF, MMS executes the action lines that update RESULTS.DIF. If ACCOUNTS.OBJ is newer than ACCOUNTS.EXE, MMS first executes the action line to update ACCOUNTS.EXE.

❷ MMS then runs the program ACCOUNTS.EXE

❸ MMS runs the DIFFERENCES utility to compare the program's output with a master file.

❹ MMS displays the results of the comparison.

If you use the previous description file example, the output would be as follows:

```
$ MMS/DESCRIPTION=BALANCE
LINK ACCOUNTS.OBJ           ! Links ACCOUNTS program
RUN ACCOUNTS.EXE            ! Runs ACCOUNTS
DIFFERENCES/OUTPUT=RESULTS.DIF      ACCOUNTS.DAT, BENCHMARK.DAT
! Compares program output to master file
TYPE RESULTS.DIF    ! Displays results of comparison
Number of difference sections found: 0
Number of difference records found: 0
DIFFERENCES /MERGED=1/OUTPUT=USER$:[ALISON]RESULTS.DIF;1-
    USER$:[ALISON]ACCOUNTS.DAT;19-
    USER$:[ALISON]BENCHMARK.DAT;27
$
```

When you run MMS, all action lines and any comments specified on action lines are written to SYS$OUTPUT or to a file you specify with the MMS /OUTPUT qualifier. The /OUTPUT qualifier is described in the Command Dictionary.

## 2.7.2 $STATUS and $SEVERITY

As each action line completes execution, MMS executes a command in the subprocess to write the value of $STATUS to a mailbox. The parent process can then determine if the action line executed successfully. The values of $STATUS and $SEVERITY are set when the execution of this internal MMS command succeeds. Consequently, $STATUS and $SEVERITY always indicate

success. You cannot test the values of these variables in a description file. You can, however, control the behavior of MMS with the /[NO]IGNORE qualifier because it tells MMS what to do when it encounters Warning, Error, or Fatal errors. See the Command Dictionary for more information on severity errors.

## 2.7.3 MMS$STATUS

MMS uses a special symbol, MMS$STATUS, to record the return status of the last action line it executed. MMS$STATUS is set in the parent process running MMS and reflects the value of $STATUS returned from the child process. The value of MMS$STATUS is constantly changing with the completion of each action line. You cannot use MMS$STATUS from within the child process because symbols are passed only from the parent to the child process when the child process is created. When MMS exits, MMS$STATUS reflects the status of the last command executed in the child process.

If the value of MMS$STATUS is an even number, the last action line terminated with an error. If the value of MMS$STATUS is an odd number, the last action line executed successfully. To check the value of MMS$STATUS, issue the DCL command SHOW SYMBOL after MMS has finished processing your description file. Do not confuse MMS$STATUS with the $STATUS condition value returned by MMS itself. MMS$STATUS contains the status of the last action line executed; $STATUS contains the status resulting from the termination of the MMS image.

## 2.7.4 Action Line Prefixes

An **action line prefix** is a single-character modifier that controls the processing of a single action line in a description file.

The two action line prefixes are described in Table 2–1.

**Table 2–1: MMS Action Line Prefixes**

| Prefix | Function |
|---|---|
| - (Ignore) | Causes MMS to ignore errors generated by the action line on which the prefix appears. |

(continued on next page)

**Table 2–1 (Cont.): MMS Action Line Prefixes**

| Prefix | Function |
|---|---|
| @ (Silent) | Suppresses the writing to the output file of the action line on which the prefix appears. (The output file can be either SYS$OUTPUT or the file specified by the /OUTPUT qualifier.) |

You cannot override either action line prefix from the MMS command line.

An action line prefix must appear as the first nonblank character on an action line; however, a prefix may not appear in column 1 of the line. The rest of the action line must be separated from the prefix by at least one space or tab. You can use both prefixes on the same action line by typing them next to each other with no intervening spaces or tabs. The following example shows the use of both prefixes:

```
A : B
     @- Write SYS$OUTPUT "It worked!"
```

MMS also provides two directives (discussed in Section 2.8), .IGNORE and .SILENT, that are similar in function to - and @, respectively. The difference between the action line prefixes and the directives with the same functions (.IGNORE and .SILENT) is that a prefix affects the processing of only one line in the description file, while a directive affects the processing of the entire file.

## 2.7.5 The Ignore Prefix (-)

The Ignore action line prefix (-) directs MMS to ignore any errors that occur during the processing of the action line on which the prefix appears.

The following dependency rule tests the BASIC compiler with a source file known to contain errors. Normally, the BASIC compiler aborts the compilation when it encounters an error, and MMS aborts execution as well. In this case, the Ignore prefix directs MMS to ignore the error and execute the EDIT command.

```
TESTERR : ERRORS.BAS
     - BASIC /LIST=ERRORS ERRORS
     EDIT/COMMAND=EXTRACT.EDT ERRORS.LIS
```

## 2.7.6 The Silent Prefix (@)

The Silent action line prefix (@) stops MMS from writing an action line to SYS$OUTPUT or to the file specified by the /OUTPUT qualifier. This prefix, which affects only the action lines on which it appears, is useful when you do not want certain commands echoed at execution.

For example, the Silent action line prefix directs MMS to suppress the display of the following action line:

```
@ DELETE *.LIS;*
```

The Silent action line prefix can be useful in cleanup procedures. In the next example, MMS deletes compilation listings from the [LISTINGS] directory, and returns to the [WORKING] directory. Because the Silent prefix suppresses the action lines, MMS can do its work silently and then display the text "Cleanup done" when the task is completed.

```
CLEANUP :
     @ SET DEFAULT [LISTINGS]
     @ DELETE *.*;*
     @ SET DEFAULT [WORKING]
     @ WRITE SYS$OUTPUT "Cleanup done"
```

MMS assumes that an at sign (@) followed by a space or tab signifies the Silent prefix. If you want to invoke a command procedure from an action line, you must omit the space between the at sign and the name of the command procedure.

## 2.7.7 Action Line Restrictions

Action lines are subject to the following restrictions:

- The maximum length of an uncontinued action line is 251 characters. The maximum length of a continued action line is 1019 characters. If you use VAX DEC/Shell as your Command Language Interpreter (CLI), the limits are 131 and 507 characters, respectively, for uncontinued and continued action lines.
- The maximum length of a quoted string of a comment in action lines is restricted to 130 characters.
- Quotes imbedded in other quotes on action lines may not behave as expected. However, if you assign the inner quote to be a DCL symbol, you can use the DCL symbol within the outer quoted string.

- An action line cannot receive data from SYS$INPUT. For example, an action line may not contain the DCL command CREATE and cannot read data from the terminal.
- An action line may not contain the DCL commands LOGOUT, EXIT, or STOP.
- An action line can spawn a subprocess only by using the $(MMS) reserved macro (see Section 3.3). The DCL command SPAWN is not allowed in an action line.
- An action line may not contain the DCL commands SET VERIFY or SET ON. You can use these commands in a command procedure that you invoke from an action line. If you use SET VERIFY, however, you must be sure to issue SET NOVERIFY before the command procedure ends.
- An action line may not contain the DCL command GOTO or labels. You can use GOTO or labels in a command procedure that you invoke from an action line because action lines are executed individually.
- You cannot direct output to TT: from an action line because MMS equates TT: as SYS$INPUT and this can result in MMS hanging.
- You cannot test the values of $STATUS and $SEVERITY in your description file because the value is always success. The value is set when the execution of an internal MMS command succeeds. The value of MMS$STATUS also changes with the completion of each action line executed.

## 2.8 Using Directives

A **directive** is a word that instructs MMS to take a certain action as it processes a description file. A directive can appear on any line in the description file, but it controls the processing of the entire file.

A directive must start in column 1 of a line. You can type a directive in either uppercase or lowercase letters, or a combination of both. Table 2–2 lists the directives and their functions. Detailed descriptions of the directives are provided in the sections that follow.

**Table 2–2: MMS Directives**

| Directive | Function |
|---|---|
| .IGNORE | Causes MMS to ignore all errors generated by all action lines and to continue processing the description file. |
| .SILENT | Suppresses the writing of all action lines to the output file (whether to SYS$OUTPUT or to the file specified by the /OUTPUT qualifier). |
| .DEFAULT | Indicates actions to be performed if MMS built-in rules or user-defined rules do not specify how to update a target. |
| .SUFFIXES | Clears, adds to, or redefines the suffixes precedence list. |
| .INCLUDE | Includes the specified file in the description file. |
| .FIRST | Indicates actions to be performed before MMS has executed any action lines to update the target. |
| .LAST | Indicates actions to be performed after MMS has executed all the action lines that update the target. |
| .IFDEF | Causes subsequent lines of a description file to be processed only if the specified macro is defined. |
| .ELSE | Causes subsequent lines of a description file to be processed if the specified macro for the .IFDEF directive is undefined. |
| .ENDIF | Terminates the set of lines in the description file whose processing is controlled by .IFDEF or .ELSE. |

## 2.8.1 The .IGNORE Directive

The .IGNORE directive tells MMS to ignore warnings, errors, and fatal errors that occur during the execution of an action line and to continue processing the description file. Without the .IGNORE directive, MMS aborts execution if it detects an error while processing an action line.

The .IGNORE directive in the following description file tells MMS to continue processing even if it encounters errors while running DIGITAL Standard Runoff (DSR) to update the target:

```
.IGNORE

BOOK.MEM : CHAPTER1.MEM, CHAPTER2.MEM, CHAPTER3.MEM, CHAPTER4.MEM
     COPY/LOG CHAPTER1.MEM BOOK.MEM
     APPEND/LOG CHAPTER2.MEM BOOK.MEM
     APPEND/LOG CHAPTER3.MEM BOOK.MEM
     APPEND/LOG CHAPTER4.MEM BOOK.MEM

CHAPTER1.MEM : CHAPTER1.RNO
     RUNOFF CHAPTER1
```

```
CHAPTER2.MEM : CHAPTER2.RNO
     RUNOFF CHAPTER2

CHAPTER3.MEM : CHAPTER3.RNO
     RUNOFF CHAPTER3

CHAPTER4.MEM : CHAPTER4.RNO
     RUNOFF CHAPTER4
```

If CHAPTER3.RNO contains DSR errors, and you run MMS with this description file (BOOK.MMS), the following lines may appear on your screen.

```
$ MMS/DESCRIPTION=BOOK
RUNOFF CHAPTER1
DIGITAL Standard Runoff Version V2.0-014: No errors detected
5 pages written to "USER$:[MICHAELS]CHAPTER1.MEM;1"
RUNOFF CHAPTER2
DIGITAL Standard Runoff Version V2.0-014: No errors detected
16 pages written to "USER$:[MICHAELS]CHAPTER2.MEM;1"
RUNOFF CHAPTER3
%RUNOFF-W-CJL, Can't justify line
        on output page 2; on input line 46 of page 1 of file "USER$:[MICHAELS]CH
APTER3.RNO;1"
%RUNOFF-W-CJL, Can't justify line
        on output page 2; on input line 52 of page 1 of file "USER$:[MICHAELS]CH
APTER3.RNO;1"
%RUNOFF-W-TFE, Too few end commands
        on output page 3; on input line 77 of page 1 of file "USER$:[MICHAELS]CH
APTER3.RNO;1"
%RUNOFF-W-BMS, Bad margin specification: ".lm70
        on output page 4; on input line 102 of page 1 of file "USER$:[MICHAELS]C
HAPTER3.RNO;1"
%RUNOFF-W-COR, Can't open required file "TABLE1.RNO"
        on output page 5; on input line 154 of page 1 of file "USER$:[MICHAELS]C
HAPTER3.RNO;1"
DIGITAL Standard Runoff Version 2.0-014: 5 diagnostic messages reported
10 pages written to "USER$:[MICHAELS]CHAPTER3.MEM;1"
RUNOFF CHAPTER4
DIGITAL Standard Runoff Version V2.0-014: No errors detected
13 pages written to "USER$:[MICHAELS]CHAPTER4.MEM;1"
COPY/LOG CHAPTER1.RNO BOOK.MEM
%COPY-S-COPIED, USER$:[MICHAELS]CHAPTER1.MEM;1 copied to USER$:[MICHAELS]BOOK.ME
M;1 (35 blocks)
APPEND/LOG CHAPTER2.MEM BOOK.MEM
%APPEND-S-APPENDED, USER$:[MICHAELS]CHAPTER2.MEM;1 appended to USER$:[MICHAELS]B
OOK.MEM;1 (1452 records)
APPEND/LOG CHAPTER3.MEM BOOK.MEM
%APPEND-S-APPENDED, USER$:[MICHAELS]CHAPTER3.MEM;1 appended to USER$:[MICHAELS]B
OOK.MEM;1 (1508 records)
APPEND/LOG CHAPTER4.MEM BOOK.MEM
%APPEND-S-APPENDED, USER$:[MICHAELS]CHAPTER4.MEM;1 appended to USER$:[MICHAELS]B
OOK.MEM;1 (621 records)
$
```

Although errors occurred in the processing of CHAPTER3.RNO, MMS continued to execute action lines, successfully processing CHAPTER4.RNO. Had .IGNORE not been specified, MMS would have terminated execution upon encountering errors in CHAPTER3.RNO; the last action line would not have been executed.

**NOTE**

You should be careful about executing MMS with the .IGNORE directive. If errors occur during processing, the target may be updated yet still contain errors of which you will be unaware.

To override the .IGNORE directive for a particular MMS build, use the /NOIGNORE, /IGNORE, /IGNORE=WARNING, or /IGNORE=ERROR qualifier on the MMS command line when invoking MMS. (See the Command Dictionary for more information on the /IGNORE qualifier.)

## 2.8.2 The .SILENT Directive

The .SILENT directive tells MMS to suppress the display of action lines. Normally, MMS writes action lines either to SYS$OUTPUT or into a file specified by the /OUTPUT qualifier. Action lines are always executed even if they are not displayed, unless you specify the /NOACTION qualifier on the command line. The /OUTPUT and /NOACTION qualifiers are described in the Command Dictionary.

The .SILENT directive does not suppress the display of error messages generated by execution of action lines.

The following example illustrates the use of the .SILENT directive.

```
.SILENT

PROG.EXE : MOD1.OBJ, MOD2.OBJ
     LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ

MOD1.OBJ : MOD1.C

MOD2.OBJ : MOD2.C
```

MMS processes this description file without displaying action lines. The Command Language Interpreter (CLI) prompt returns when the target, PROG.EXE, has been updated.

To override the .SILENT directive for a particular MMS build, use the /VERIFY qualifier on the MMS command line when invoking MMS. (The /VERIFY qualifier is described in the Command Dictionary.)

## 2.8.3 The .DEFAULT Directive

The .DEFAULT directive tells MMS to continue processing the description file even if it encounters a dependency rule for which there is neither a specified action line nor applicable built-in or user-defined rules. Rather than abort execution in such a situation, MMS executes the default action you specify and continues processing the description file.

The .DEFAULT directive has the following format:

```
.DEFAULT
     action line...
```

The action line is a command-language command that MMS executes by default. You can specify as many action lines as you like.

The .DEFAULT directive can be useful when you are developing a system that contains inoperative parts and you want MMS to process the operating portions and inform you about the inoperative parts. If you have only one module, TEST.D, finished for your system, you can build the system if your description file, TESTSYS.MMS, is as follows:

```
.DEFAULT
     ! Source $(MMS$TARGET) not yet added

TEST.A : TEST.B

TEST.B : TEST1.C TEST2.E TEST3.F

TEST1.C : TEST1.D
     COPY TEST1.D TEST1.C
```

When MMS processes the TESTSYS.MMS description file, it expands the MMS$TARGET special macro to the name of the target and writes the following lines to SYS$OUTPUT:

```
$ MMS/DESCRIPTION=TESTSYS
COPY TEST1.D TEST1.C
! Source TEST2.E not yet added
! Source TEST3.F not yet added
! Source TEST.B not yet added
! Source TEST.A not yet added
$
```

By using the .DEFAULT directive, MMS reminds you of the modules you have not yet implemented.

Another way to use the .DEFAULT directive is to copy files from one directory to another. For example:

```
.DEFAULT :
    COPY $(MMS$SOURCE) $(MMS$TARGET)

TEST.BLI : [PROJECT.FILES]TEST.BLI

PROG.BLI : [PROJECT.FILES]PROG.BLI
```

The sources in this description file exist in a common directory for the project. Because these dependency rules have no action lines and there are no built-in or user-defined rules that apply, MMS executes the action line specified by .DEFAULT and copies the required files into your directory. (The MMS$SOURCE and MMS$TARGET special macros are described in Section 2.5.)

**NOTE**

.DEFAULT cannot be changed or overridden from the MMS command line.

## 2.8.4 The .SUFFIXES Directive

The .SUFFIXES directive allows you to redefine the suffixes precedence list so that you can reorder the list of file types, add new file types, or disable recognition of all file types.

MMS uses the suffixes precedence list to determine the order in which it looks for sources and targets when applying built-in rules. MMS also uses this list to determine which built-in rule will update the specified target. Section 2.3 contains a detailed discussion of how the suffixes precedence list and MMS built-in rules work together. Also, Table C–4 lists the suffixes in order of their precedence.

The .SUFFIXES directive has the following format:

```
.SUFFIXES [file types list]
```

The file types list is a list of file types in order of precedence. If you omit the file types list entirely, the suffixes precedence list is cleared and all built-in rules are disabled.

Once you set up a new list of suffixes, MMS recognizes only the specified file types and enables built-in and user-defined rules for the specified suffixes.

## 2.8.5 Adding a New File Extension to the Suffixes List

In previous examples, the description files used file types that MMS knew about through its built-in rules. Sometimes during software development, you need file types that MMS does not know about–for example, when you use a new programming language, or you use input and output files for a custom application, or you have included files with other file types. You can use user-defined rules and action lines in the description file to tell MMS about new file types. First, you add the file types to the MMS suffixes list, and then you write a user-defined rule or action line for the file type.

MMS uses the suffixes precedence list to analyze the relationship between file types. Table C–4 lists the suffixes or file types in their order of precedence, from left to right, targets to sources. The targets at the beginning of the list are created from some source to the right in the list. If you attempt to write a user-defined rule for a new file type without adding the file type to the list, the description file fails when it is run.

## 2.8.6 Using .SUFFIXES in a Description File

To add a new file type to your description file, use the *directive* .SUFFIXES. It clears the default suffixes list and allows you to write a new list in the description file. Table 2–2 lists the directives and their functions.

If you want MMS to access a CMS library, all the file types that could come from the library must also be present in the suffixes list. (See Table C–4 for the Suffixes Precedence List.) When you write a new suffixes list in the description file, it must contain all the file types that occur in your software system, including the following:

- The executable files
- The different types of library files
- The object files
- The source code files
- The included files

Consider the following description file, NEW_SUFFIX.MMS:

## Example 2–7: A Description File Using .SUFFIXES

```
$ TYPE NEW_SUFFIX.MMS
  !
  ! Set a new suffixes list
  !
❶ .SUFFIXES
❷ .SUFFIXES .EXE .OBJ .NEW .FOR

  !
  ! User-defined rules
  !

❸ .NEW.OBJ
  @NEW_COMPILER $(MMS$SOURCE) $(MMS$TARGET)

❹ .OBJ.EXE
        $(LINK) $(LINKFLAGS) $(MMS$SOURCE_LIST)

  !
  ! The executable and its sources
  !

  MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ

  !
  ! Object files and their sources
  !

❺ MAIN.OBJ DEPENDS_ON MAIN.FOR
❻ SUB1.OBJ DEPENDS_ON SUB1.NEW
```

❶ The first .SUFFIXES list clears the default suffix list. If you do not clear the default suffixes list before adding a new file type, MMS appends the new file types that you add to the end of the list. This is risky because there is an implied hierarchy in the suffixes list. Adding a new type of a source code file to the end of the list works, but adding an intermediate file type to the end of the list destroys the order of the suffixes precedence list.

❷ This line contains the new suffixes list with all the file types used to build this system.

❸ This is the user-defined rule for the new file type just added to the suffix precedence list. The command procedure NEW_COMPILER.COM is invoked to call the new compiler.

❹ The new compiler compiles the source code file in MMS$SOURCE into an object file named by (MMS$SOURCE_LIST).

❺ A built-in rule is applied for compiling the FORTRAN code file.

❻ The new user-defined rule is applied for compiling and linking .NEW.

## 2.8.7 Building a System with a New File Extension

To build your system with the description file, NEW_SUFFIX, you must have the following files in your current directory:

```
$ DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.FOR;2              17-JUL-1987 14:27
NEW_COMPILER.COM;5      17-JUL-1987 14:27
NEW_SUFFIX.MMS;1        17-JUL-1987 14:27
SUB1.NEW;1              17-JUL-1987 14:27

Total of 4 files

$ MMS/DESCRIPTION=NEW_SUFFIX

  FORTRAN /NOLIST/OBJECT=MAIN MAIN.FOR
❶ @NEW_COMPILER SUB1.NEW SUB1.OBJ
 LINK /TRACE/NOMAP/EXEC=MAIN MAIN.OBJ SUB1.OBJ
```

❶ MMS uses the new user-defined rule to compile the new language.

### Order of Suffixes

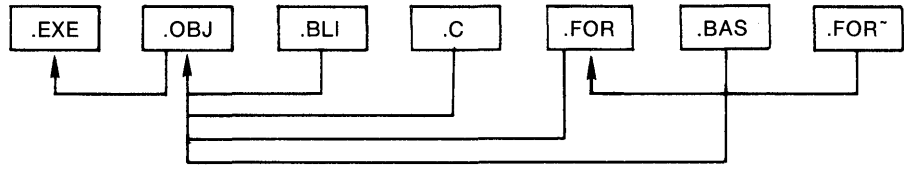
The order of suffixes changes according to their use:

- In a dependency rule, the target is on the left and the source is on the right.
- In suffixes list, the target is on the left and the source is on the right.
- In a user-defined rule, the source is on the left and the target is on the right.

## 2.8.8 The .SUFFIXES Directive Used with CMS Files

If you add a rule in your description file that directs MMS to build a .FOR file by fetching it from a CMS library, the relationship between the rules and the file types might be as follows:

**Figure 2–2: CMS Rules**




ZK-1665-84

Section 4.2 explains how to specify CMS elements in description files. (The tilde ( ~ ) signifies a file in a CMS library.) When MMS considers .FOR as a possible target, it discovers that a rule exists for building .FOR files from .FOR~ files. Therefore, it looks for a file named MOD3.FOR in the CMS library. If one exists, it applies the known rule to update the .FOR target; if it cannot find such a file, it continues searching for a file to use. If MMS does locate MOD3.FOR in the library, it can then use this file to create all the necessary sources that finally result in an updated MOD3.EXE, the original target. The simple dependency MOD3.EXE : could result in the following sequence of actions:

```
MOD3.FOR DEPENDS_ON MOD3.FOR~
     CMS FETCH MOD3.FOR

MOD3.OBJ DEPENDS_ON MOD3.FOR
     FORTRAN/OBJ=MOD3 MOD3.FOR

MOD3.EXE DEPENDS_ON MOD3.OBJ
     LINK/EXEC=MOD3 MOD3.OBJ
```

You can specify a null file type in the suffixes precedence list by using a free-standing period. For example, the following precedence list directs MMS to look for files with null file types before looking for .B32 files:

```
.SUFFIXES .EXE .OBJ . .B32
```

## 2.8.9 The .INCLUDE Directive

The .INCLUDE directive allows you to include other files in a description file. You can use this directive when you have stored common macros or user-defined rules in a separate file that can then be included by several description files.

The .INCLUDE directive has the following format:

```
.INCLUDE filespec
```

A filespec is a VMS file specification or a logical name that identifies the included file. The default file type is .MMS.

The line in the description file on which the .INCLUDE directive occurs is replaced with the contents of the specified file.

Included files may themselves include files, up to a depth of 16 or the maximum open file limit for your current process (as indicated by the FILLM quota) or whichever is less. MMS treats lines read from an included file as though they came from the original description file, except when it detects syntax errors. If an error occurs, the error message indicates the line number and the file in which the error was detected.

## 2.8.10 The .FIRST Directive

The .FIRST directive tells MMS to execute certain action lines before it executes the action lines that update the target. The .FIRST directive works with single or multiple targets. If you have selected multiple targets, then .FIRST is executed before the entire group of targets.

The .FIRST directive has the following format:

```
.FIRST
     action line...
```

The action line is a command-language command that MMS executes before it updates the target. You can specify as many action lines with .FIRST as you like.

MMS executes the action lines that accompany the .FIRST directive only if the target requires updating. The actions are executed before those that actually update the target.

The following example shows how you might use .FIRST to send a mail message to your process to notify you when MMS begins processing your description file:

```
.FIRST
     OPEN/WRITE MSGTEXT MSGTEXT.TXT
     WRITE MSGTEXT "Build of $(MMS$TARGET) now beginning"
     CLOSE MSGTEXT
     MAIL MSGTEXT.TXT ANDERSON -
          /SUBJECT="Report from MMS"

BOOK.MEM : CHAPTER1.MEM, CHAPTER2.MEM, CHAPTER3.MEM, CHAPTER4.MEM
     COPY/LOG CHAPTER1.MEM BOOK.MEM
     APPEND/LOG CHAPTER2.MEM BOOK.MEM
     APPEND/LOG CHAPTER3.MEM BOOK.MEM
     APPEND/LOG CHAPTER4.MEM BOOK.MEM
```

```
CHAPTER1.MEM : CHAPTER1.RNO
     RUNOFF CHAPTER1

CHAPTER2.MEM : CHAPTER2.RNO
     RUNOFF CHAPTER2

CHAPTER3.MEM : CHAPTER3.RNO
     RUNOFF CHAPTER3

CHAPTER4.MEM : CHAPTER4.RNO
     RUNOFF CHAPTER4
```

When this description file (BOOK.MMS) is processed, the following lines appear on your terminal (or in your output file):

```
OPEN/WRITE MSGTEXT MSGTEXT.TXT
WRITE MSGTEXT "Build of BOOK.MEM now beginning"
CLOSE MSGTEXT
MAIL MSGTEXT.TXT ANDERSON           /SUBJECT="Report from MMS"
RUNOFF CHAPTER1
RUNOFF CHAPTER2
RUNOFF CHAPTER3
RUNOFF CHAPTER4
COPY CHAPTER1.MEM BOOK.MEM
APPEND CHAPTER2.MEM BOOK.MEM
APPEND CHAPTER3.MEM BOOK.MEM
APPEND CHAPTER4.MEM BOOK.MEM
```

## 2.8.11 The .LAST Directive

The .LAST directive tells MMS to execute certain action lines after it has executed the action lines that update the target. The .LAST directive works with a single target or with multiple targets. If you have selected multiple targets, then .LAST is executed after the entire group of targets.

The .LAST directive has the following format:

```
.LAST
     action line...
```

The action line is a command-language command that MMS executes after it updates the target or targets. You can specify as many action lines with .LAST as you like.

MMS executes the action lines that accompany the .LAST directive only if the target requires updating. The actions are executed after those that actually update the target.

The following example shows how you might use .LAST:

```
A.EXE : A.OBJ
     LINK [GREGORY.OBJECTS]A.OBJ

A.OBJ : A.FOR
     FORTRAN/LIST=[GREGORY.LISTINGS]A.LIS -
         /OBJECT=[GREGORY.OBJECTS] A.FOR

.LAST
     SET DEFAULT [GREGORY.OBJECTS]
     DELETE/LOG A.OBJ;*
     SET DEFAULT [GREGORY.LISTINGS]
     PURGE/LOG A.LIS
```

If A.EXE needs to be updated, the LINK command is executed and produces an object file in the directory [GREGORY.OBJECTS]. If A.OBJ needs to be updated before it can update A.EXE, the FORTRAN command is executed and produces a listing in the directory [GREGORY.LISTINGS]. After A.EXE is up-to-date, the action lines associated with .LAST are executed to delete the object file and purge the listings directory. Notice the output that might be produced on your screen when you use the description file, ADESC.MMS.

```
$ MMS/DESCRIPTION=ADESC
FORTRAN/LIST=[GREGORY.LISTINGS]A.LIS /OBJECT=[GREGORY.OBJECTS] A.FOR
LINK [GREGORY.OBJECTS]A.OBJ
SET DEFAULT [GREGORY.OBJECTS]
DELETE/LOG A.OBJ;*
%DELETE-I-FILDEL, USER$:[GREGORY]A.OBJ;1 deleted (3 blocks)
SET DEFAULT [GREGORY.LISTINGS]
PURGE/LOG A.LIS
%PURGE-I-FILPURG, USER$:[GREGORY.LISTINGS]A.LIS;4 deleted (6 blocks)
```

MMS allows you to perfom a set of commands before or after all other actions through the use of the .FIRST and .LAST directives. The .FIRST and .LAST sections are executed only if MMS decides to take other actions to update a target.

## 2.8.12 The .IFDEF, .ELSE, and .ENDIF Directives

The .IFDEF directive tests whether a specified macro is defined. You use this directive to cause MMS not to process certain lines in your description file if the macro is undefined.

The .IFDEF directive has the following format:

```
.IFDEF macro
[description file line]...
.ENDIF
```

Macro is the name of the macro being tested. The description file line is zero or more action lines that are valid in a description file.

The .IFDEF directive must always be accompanied by a matching .ENDIF directive. MMS checks for a definition of the macro specified with the .IFDEF directive. If the macro is undefined, all lines of the description file between .IFDEF and .ENDIF (even lines that contain .IFDEF directives) are ignored.

Consider the following description file, FORPROG.MMS:

```
.IFDEF VAX

A.OBJ : A.FOR
     FORTRAN A

.ENDIF

.IFDEF PDP11

A.OBJ : A.FOR
     FORTRAN/PDP11 A

.ENDIF
```

When you invoke MMS with this description file, you can define one of the macros on the command line to determine which action line gets executed. For example:

```
$ MMS/DESCRIPTION=FORPROG/MACRO="VAX"
FORTRAN A /NOLIST/OBJECT=A.OBJ A.FOR
LINK A.OBJ
$
```

Because the command line defines the macro VAX, the command FORTRAN A is executed and the commands associated with the undefined macro PDP11 are ignored.

You may use the .ELSE directive in conjunction with the .IFDEF directive but never alone. If the specified macro for the .IFDEF directive is undefined, MMS skips all the subsequent lines of the description file until it comes to a .ELSE or a .ENDIF directive. The next example of a description file shows the format for a .IFDEF directive using .ELSE and a nested .IFDEF directive:

```
.IFDEF VAX
.IFDEF CURRENT
A.OBJ : A.FOR
     FORTRAN A
.ENDIF
A.EXE : A.OBJ
     LINK A.OBJ

.ELSE
A.OBJ : A.FOR
     FORTRAN/PDP11 A
.ENDIF
```

MMS reads the line beginning with the .IFDEF directive and tests whether the macro is defined. If the macro is defined, MMS processes the action lines between .IFDEF and the second .ENDIF except for the lines between the .ELSE and the second .ENDIF. If the specified macro for the .IFDEF directive is undefined, MMS skips all the action lines including the nested .IFDEF, until it reaches the .ELSE directive. MMS then processes the subsequent lines to the .ENDIF. When you invoke MMS with the FORPROG description file, you can define the nested macro on the command line as follows:

```
$ MMS/DESCRIPTION=FORPROG/MACRO="VAX=CURRENT"
FORTRAN A
$
```

Chapter 3

# Advanced Description File Techniques

Once you become familiar with MMS, you can use advanced techniques in your description file to make it more flexible and useful. This chapter describes the following techniques:

- Double colon dependencies
- Invoking MMS from a description file
- Invoking MMS from a command procedure
- Invoking a command procedure from a description file
- Gathering statistics
- Doing parallel processing
- Producing multiple outputs
- Changing build options

## 3.1 Using Double Colon Dependencies

In writing MMS dependency rules, you can specify the same target in more than one dependency rule, provided that you specify only one action for updating that target. For example, the following construction is legal:

```
MOD2.OBJ, MOD3.OBJ : DEFS1.DEF

MOD2.OBJ : DEFS2.DEF
     PASCAL MOD2
```

MOD2.OBJ appears in the target list of two dependency rules, but only one action (PASCAL MOD2) is specified for it.

In contrast, the following construction is invalid:

```
MOD2.OBJ, MOD3.OBJ : DEFS1.DEF
     PRINT DEFS1.DEF

MOD2.OBJ : DEFS2.DEF
     PASCAL MOD2
```

Two different actions are specified for MOD2.OBJ, requiring MMS to take two different actions if one of MOD2.OBJ's dependencies is changed.

If you want MMS to take different actions depending on which sources have changed, MMS allows you to use a double colon rather than a single colon to separate the target list from the source list in a dependency rule. (You can use the keyword **ADDITIONALLY_DEPENDS_ON** in place of the double colon.) For example, in the previous dependency rules, MMS was to execute the PRINT command if DEFS1.DEF had changed and the PASCAL command if MOD2.PAS had changed. The double colon or the keyword **ADDITIONALLY_DEPENDS_ON** directs MMS to allow the same target to be specified in more than one dependency rule, each of which may require different actions to update the target. By using the double colon, you can modify the previous example to execute as you intended:

```
MOD2.OBJ, MOD3.OBJ :: DEFS1.DEF ! If at least one source is
     PRINT DEFS1.DEF     ! newer than targets, print DEFS1.DEF

MOD2.OBJ :: DEFS2.DEF    ! If MOD2.PAS or DEFS2.DEF is newer than
     PASCAL MOD2         ! MOD2.OBJ, compile MOD2.PAS
```

Note that in this example, if DEFS2.DEF or MOD2.PAS is newer than MOD2.OBJ and DEFS1.DEF, both action lines are executed. However,that is the normal behavior of MMS. In effect, double colons produce the same result as single colons, so their usefulness is limited.

In a description file, a given target can be included in either a single colon dependency rule or in a double colon dependency rule, but not in both. MMS issues an error message if you try to specify both kinds of rules for the same target.

## 3.2 Maintaining a Library of Object Files

You can use MMS to maintain a library of object files. Consider the library called UTIL.LIB, which contains three object modules: MOD1.OBJ, MOD2.OBJ, and MOD3.OBJ. If one of these object modules is updated, it should be added to the library. A description file for this system might look like the following:

```
UTIL.LIB :: MOD1.OBJ
     LIBR UTIL.LIB MOD1.OBJ

UTIL.LIB :: MOD2.OBJ
     LIBR UTIL.LIB MOD2.OBJ

UTIL.LIB :: MOD3.OBJ
     LIBR UTIL.LIB MOD3.OBJ
```

UTIL.LIB depends on all three object modules, but MMS takes different actions depending on which module is out of date. However, library module file specifications are not allowed as targets in double colon dependency rules. For example, the following dependency rules cause MMS to perform the same actions as the previous examples but they are written with the single colon rule:

```
UTIL(MOD1) : MOD1.OBJ
     LIBR UTIL.LIB MOD1.OBJ

UTIL(MOD2) : MOD2.OBJ
     LIBR UTIL.LIB MOD2.OBJ

UTIL(MOD3) : MOD3.OBJ
     LIBR UTIL.LIB MOD3.OBJ
```

When using the library module specification format, only the single colon is acceptable to MMS.

## 3.3 Invoking MMS from a Description File

When you invoke MMS, it runs two processes as it updates a target. The first process, your current process, executes MMS. The second process, a spawned subprocess, executes the action line you specified in the description file to update the target. MMS creates a spawned subprocess only when the target needs updating, and it creates only one spawned subprocess to execute all the actions in the description file. The subprocess is created when the first action is executed; it remains active until the MMS image terminates.

While a subprocess executes an action (such as a DCL command), the parent process waits until it is notified that the subprocess has finished executing commands. If you monitor the parent process, you may find it idle.

You can invoke MMS from a description file while MMS is updating a target. Consider the following description file:

```
MAIN.EXE DEPENDS_ON MAIN.OBJ
        MMS/DESCRIPTION=TOOLS_LIB
        LINK MAIN.OBJ, TOOLS_LIB/LIB

MAIN.OBJ DEPENDS_ON MAIN.PAS
```

In this example, the executable file MAIN.EXE is rebuilt if MAIN.OBJ has been changed since the last build. However, before MMS performs the linking action, an MMS subprocess is invoked to rebuild and update the library if necessary.

## 3.3.1 Invoking MMS from a Description File with $(MMS)

You can also invoke MMS from within a description file by specifying the reserved macro $(MMS) on an action line where you want MMS to be invoked again.

When you invoke MMS from a description file with the $(MMS) reserved macro, another subprocess is used to execute the new invocation of MMS. The second invocation of MMS runs as a spawned subprocess that inherits any existing symbol definitions. The original subprocess is treated as a parent process for the subsequent MMS execution.

As MMS processes the description file, it executes any action line that contains the reserved macro $(MMS), even if you specified the /NOACTION qualifier on the command line. (/NOACTION suppresses the execution of action lines and is described in the Command Dictionary.) Thus, the MMS subprocess is created but no other actions are performed.

## 3.3.2 Process Quotas for MMS Subprocesses

When a subprocess is created, the VMS operating system automatically assigns it a portion of the quotas established for your main process. Under heavily loaded systems, it is possible for the top-level MMS subprocess to complete before VMS has finished with the exit-cleanup code for the second-level MMS subprocess. If MMS tries to invoke another subprocess, you may receive the following error message:

```
%MMS-F-DRVINSQUO, Your process needs a PRCLM of at least 2, current value is 0.
```

In this case, you can increase your PRCLM quota or add the DCL WAIT statement in your description file, for example:

```
IF F$SEARCH('(MMS$SOURCE") .NES. "" THEN-
    DESCRIPTION = "/DESCRIPTION=$(MMS$SOURCE)"
- $(MMS) /NOSKIP"MMS_QUALFIERS"-
  /OVERRIDE/RULES=BUILD_COM:DESCRIP_BUILD-
  "DESCRIPTION" $(MMS$SOURCE)
  WAIT 0:0:5
- $(MMS) /NOSKIP"MMS_QUALIFIERS"-
  /OVERRIDE/RULES=BUILD_COM:PLI-
  /DESCRIPTION=$(MMS$SOURCE)
```

## 3.3.3 Process Quotas for Using MMS

To invoke MMS as a top-level process, you need the following minimum process quotas:

**Table 3–1: MMS Process Quotas**

| Process | Quota |
|---|---|
| PRCLM | A subprocess limit of 2 |
| FILLM | An open file limit of 16 |
| BYTLM | A buffered I/O byte limit of 8192 (nonrecursive) and 13000 (recursive) |
| ASTLM | An asynchronous trap limit of 25 |

If you get the error message reflecting a "virtual memory exceeded" error, you may also need to increase your PGFLQUO (page file quota). If you use MMS recursively, that is, you invoke MMS from within an MMS process, then MMS requires even higher quotas.

## 3.3.4 MMS Reserved Macros

MMS includes two other reserved macros, $(MMSQUALIFIERS) and $(MMSTARGETS), which you can use when you invoke MMS as a subprocess. Both of these qualifiers pass to the subprocess the same information you specified on the command line that invoked MMS:

- $(MMSQUALIFIERS) passes the command-line qualifiers.
- $(MMSTARGETS) passes the targets from the command line.

These two macros and $(MMS) are reserved macros; you cannot redefine them.

The $(MMSQUALIFIERS) macro does not pass the /DESCRIPTION, /OUTPUT, /IGNORE, and /NORULES qualifiers. To use these qualifiers when invoking MMS from a description file, you must explicitly specify them after $(MMSQUALIFIERS), as shown in the following example:

```
TESTS.EXE :
     $(MMS) $(MMSQUALIFIERS) -
          /DESCRIPTION=[GREGORY]TESTBUILD -
          $(MMSTARGETS)
```

If you do not use the $(MMSQUALIFIERS) macro, MMS uses the default qualifiers. A list of the default qualifiers and complete descriptions of all MMS qualifiers are contained in the Command Dictionary.

The following example shows a description file, ALL.MMS, that contains two subprocess invocations of MMS:

```
ALL.EXE : A.OBJ, B.OBJ
     LINK/EXEC=ALL A, B

A.OBJ :
     $(MMS) $(MMSQUALIFIERS) /DESCRIPTION=A A.OBJ

B.OBJ :
     $(MMS) $(MMSQUALIFIERS) /DESCRIPTION=B B.OBJ
```

Before MMS can update the target, ALL.EXE, it must check the two sources, A.OBJ and B.OBJ, to make sure they are up to date. If either needs to be updated, MMS spawns a subprocess, using the specified description file. If both A.OBJ and B.OBJ need to be updated, the output from this example is the following:

```
$MMS/DESCRIPTION=ALL
MMS  /DESCRIPTION=A A.OBJ
PASCAL A
MMS  /DESCRIPTION=B B.OBJ
PASCAL B
LINK/EXEC=ALL A,B
$
```

If you invoke MMS with the /NOACTION qualifier and the same description file, the following output results:

```
$MMS/DESCRIPTION=ALL/NOACTION
MMS /NOACTION /DESCRIPTION=A A.OBJ
PASCAL A
MMS /NOACTION /DESCRIPTION=B B.OBJ
PASCAL B
LINK/EXEC=ALL A,B
$
```

The MMS subprocesses are created, but the PASCAL and LINK commands are not executed to update the targets because you specified /NOACTION on the MMS command line.

## 3.4 Invoking MMS from a Command Procedure

The previous description files have built software systems in a fixed way. You can build variations of your software system without modifying the description file by invoking MMS from a command procedure. The command procedure controls the actions MMS performs. You can use user-defined macros in a command procedure to change MMS's default actions.

Some of the ways you can vary building your software systems can include the following:

- Using /DEBUG versus /NODEBUG images
- Producing listing files versus no listing files during compilation
- Using /OPTIMIZE versus /NOOPTIMIZE during compilaton

It is better to have the description file reflect the basic structure of your software system and use command procedures to produce variations in your build process. The description file should not be concerned with changing the details of compile and link options.

### Command Procedures and User-Defined Macros

MMS has two types of built-in macros: default macros and special macros. MMS default macros stand for parts of built-in actions and can be overridden. MMS special macros stand for targets and sources and cannot be overridden. The special macros are defined when you invoke MMS and cannot be changed. Default macros are a set of string variables containing the names of VMS utilities and qualifiers.

You can create a user-defined macro for a CLI symbol. With the /OVERRIDE qualifier, you can instruct MMS to use the value of the user-defined macro in your command procedure over the default value of the CLI symbol. For example, consider the following command procedure, DEBUG_VERSION.COM, and the MMS description file SYSTEM1.MMS:

## Example 3–1: Invoking MMS from a Command Procedure

```
$   TYPE DEBUG_VERSION.COM

  $ ! ------------------------------------------------------
  $ !
  $ ! Command procedure using the /OVERRIDE qualifier with MMS
  $ !
  $ ! Create the user defined macros we want
  $ !
❶ $ PFLAGS = "/LIST/NOOPTIM/DEBUG"
❶ $ LINKFLAGS = "/MAP/DEBUG"
  $ !
  $ ! Invoke MMS and direct it to use our macros
  $ !
❷ $ MMS /DESCRIPTION=SYSTEM1 /OVERRIDE
  $ !
  $ ! ------------------------------------------------------
  $ !

❸ $   TYPE SYSTEM1.MMS

MAIN.EXE DEPENDS_ON MAIN.OBJ
MAIN.OBJ DEPENDS_ON MAIN.PAS
```

❶ PFLAGS and LINKFLAGS are CLI symbols with the same names as those of the default macros but set with new values.

❷ MMS with the /OVERRIDE qualifier is invoked from within the command procedure to use the new CLI symbol values instead of the built-in default values.

❸ The description file describes the dependencies.

This example produces a software system very different from the one MMS would normally have built. The user-defined macro definitions in the command procedure appear in the MMS output exactly where the default macro actions would have appeared. To invoke this command procedure you need the following files in your current directory:

```
$  DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

MAIN.PAS;3            18-JUL-1987 16:35
DEBUG_VERSION.COM;2   18-JUL-1987 16:35
SYSTEM1.MMS;1         18-JUL-1987 16:35

Total of 3 files.


$  @DEBUG_VERSION
```

```
PASCAL /LIST/NOOPTIM/DEBUG MAIN.PAS
LINK /MAP/DEBUG MAIN.OBJ
```

MMS uses the user-defined rules to compile and link your program.

## 3.5 Invoking a Command Procedure from a Description File

You can invoke DCL command procedures from your description file. The following example describes a command procedure that loops until a given file becomes available. You can invoke that command procedure, called GETFILE.COM, from your description file as in Example 3–2.

**Example 3–2: Invoking a Command Procedure from a Description File**

```
$ TYPE GETFILE.COM
$ LABEL:
$ IF "''F$SEARCH(" 'P1' ")'" .NES. "" THEN GOTO DONE
$ WAIT +:'P2'
$ GOTO LABEL
$ DONE:
$ TYPE GET_NEXT.MMS
GET_NEXT_INFO :
     MAIL NL: $(MY_PROC)/SUBJECT="string"
     @GETFILE ANSWER.IN 15
     @ANSWER.IN
$ MMS/DECRIPTION=GET_NEXT
```

You can use this command procedure when you start MMS in a batch job. ANSWER.IN corresponds to the P1 parameter, and 15 is the polling interval (in minutes) that corresponds to the P2 parameter. ANSWER.IN might modify the environment in some way—for example, it might set a CMS library at a point where MMS cannot find the right CMS library.

The $(MY_PROC) macro in this description file is assumed to be a DCL symbol that represents a valid electronic mail address.

**NOTE**

Be sure you do not leave a space between the "at" sign (@) and the name of the command procedure, so that MMS does not interpret the "at" sign as the Silent action line prefix.

## 3.6 Changing System Build Options

During software development, the number of description files and major aspects of build procedures can change frequently. You can edit the command procedure to make the changes. As the MMS description file becomes stable, you can create a command procedure that prompts you for different system building options. You can write a very complicated command procedure that asks you for compilation options, for link options, and for executable targets (when your description file has multiple targets). The command procedure can send you mail when the build is complete and move the results to another directory. You can also do some error checking for your input or add a default option for your input.

The following command procedure CHANGE_OPTIONS.COM uses the DCL INQUIRE command to change compiling and linking options for a system build:

### Example 3–3: A Command Procedure to Change Build Options

```
$ TYPE CHANGE_OPTIONS.COM

  $ !
  $ ! Command procedure to vary build options for MMS
  $ !
  $ ! Ask for compilation and linking options
  $ !
  $ INQUIRE PFLAGS "Enter PASCAL compilations options"
  $ INQUIRE LINKFLAGS "Enter link options"
  $ !
  $ ! Invoke MMS and direct it to use new default macros
  $ !
  $ MMS /DESCRIPTION=SYSTEM1 /OVERRIDE
  $ !


$ TYPE SYSTEM1.MMS

MAIN.EXE DEPENDS_ON MAIN.OBJ
MAIN.OBJ DEPENDS_ON MAIN.PAS


❶ Enter PASCAL compilations options:  /LIST/NOOPTIM/DEBUG
❶ Enter link options:  /MAP/DEBUG
❷ PASCAL /LIST/NOOPTIM/DEBUG MAIN.PAS
❷ LINK /MAP/DEBUG MAIN.OBJ

❸ $ EDIT MAIN.PAS
```

(continued on next page)

**Example 3–3 (Cont.): A Command Procedure to Change Build Options**

```
❹ $  @CHANGE_OPTIONS

❶ Enter PASCAL compilations options:  /NODEBUG/NOLIST
❶ Enter link options:  /NOMAP/NODEBUG

❷ PASCAL /NODEBUG/NOLIST MAIN.PAS
❷ LINK /NOMAP/NODEBUG MAIN.OBJ
```

❶ The command procedure prompts you for compiling and linking options.

❷ The system is built using the options you supplied.

❸ The MAIN.PAS file is edited to force MMS to rebuild the system. If you invoked MMS again without changing anything, MMS would find the system up-to-date and take no action.

❹ The command procedure prompts you for options that change the way you built your system the last time.

Note that if you change the way a system is built either by modifying the description file or by modifying the command procedure that calls it, MMS does not necessarily rebuild the system. MMS bases its actions on the dates of the software system itself. MMS does not compile or relink the system just because you added a user-defined macro to your description file or your command procedure.

To invoke the CHANGE_OPTIONS command procedure, you need the following files in your current directory:

```
$  DIR/DATE=MODIFIED

Directory DISK1:[BUILD]

CHANGE_OPTIONS.COM;1
                    21-JUL-1987 15:51
MAIN.PAS;3          18-JUL-1987 16:35
SYSTEM1.MMS;1       18-JUL-1987 16:35

Total of 3 files.


$  @CHANGE_OPTIONS
```

## 3.7 Gathering Statistics

The examples in the following sections describe the methods for using MMS to gather statistics about your files.

### 3.7.1 Finding Missing Sources

If you have stored the sources for your software system in a source directory or CMS library, and want to make sure all the sources you need are there, you can get a list of any missing files by inserting a default action in your description file, using the .DEFAULT directive. For example:

```
.DEFAULT :
     IF "''F$SEARCH("MISSING.SRC")'" .EQS. "" -
     THEN OPEN/WRITE MSING MISSING.SRC
     IF "''F$SEARCH("MISSING.SRC")'" .NES. "" -
     THEN OPEN/APPEND MSING MISSING.SRC
     WRITE MSING "missing $(MMS$TARGET_NAME)"
     CLOSE MSING
```

When you process this description file with MMS, MISSING.SRC contains the list of missing files.

### 3.7.2 Creating a Checkpoint File

You can use MMS to create a checkpoint file that indicates when MMS finishes building a target. For example, if your directory contains source files TEST1.C, TEST2.C, and TEST3.C and you want MMS to create .EXE files from each of these sources and also to inform you when each target is complete, the following example shows a description file that accomplishes these tasks. This description file builds TEST1.EXE, TEST2.EXE, and TEST3.EXE and creates a file called CHECK.PNT that indicates the time the executable files were completed.

```
 ! Suffixes list with .PNT in the first position.
.SUFFIXES
.SUFFIXES .PNT .EXE .OBJ .C .C~

 ! User-defined rule to build .EXE files from .PNT files.
.EXE.PNT :
     IF "''F$SEARCH("CHECK.PNT")'" .EQS. "" -
          THEN OPEN/WRITE CHECK CHECK.PNT
     IF "''F$SEARCH("CHECK.PNT")'" .NES. "" -
          THEN OPEN/APPEND CHECK CHECK.PNT
     WRITE CHECK "Completed build of $(MMS$SOURCE) at ''f$time()'"
     CLOSE CHECK
```

```
MAIN_TARGET : TEST1.PNT, TEST2.PNT, TEST3.PNT
     MAIL CHECK.PNT MICHAELS -
     /SUBJECT="Build summary of $(MMS$TARGET_NAME) ending at ''f$time()'"
     DELETE CHECK.PNT;
```

**NOTE**

The executable files will be built before the .PNT files are processed. They are temporary files that allow the actions that produce the file to be localized in one place (the .EXE.PNT rule).

When you run MMS, the action lines are displayed as follows:

```
CC /NOLIST TEST1.C
LINK /TRACE TEST1.OBJ
IF "''F$SEARCH("CHECK.PNT")'" .EQS. "" THEN OPEN/WRITE CHECK CHECK.PNT
IF "''F$SEARCH("CHECK.PNT")'" .NES. "" THEN OPEN/APPEND CHECK CHECK.PNT
WRITE CHECK "Completed build of TEST1.EXE at ''f$time()'"
CLOSE CHECK
CC /NOLIST TEST2.C
LINK /TRACE TEST2.OBJ
IF "''F$SEARCH("CHECK.PNT")'" .EQS. "" THEN OPEN/WRITE CHECK CHECK.PNT
IF "''F$SEARCH("CHECK.PNT")'" .NES. "" THEN OPEN/APPEND CHECK CHECK.PNT
WRITE CHECK "Completed build of TEST2.EXE at ''f$time()'"
CLOSE CHECK
CC /NOLIST TEST3.C
LINK /TRACE TEST3.OBJ
IF "''F$SEARCH("CHECK.PNT")'" .EQS. "" THEN OPEN/WRITE CHECK CHECK.PNT
IF "''F$SEARCH("CHECK.PNT")'" .NES. "" THEN OPEN/APPEND CHECK CHECK.PNT
WRITE CHECK "Completed build of TEST3.EXE at ''f$time()'"
CLOSE CHECK
MAIL CHECK.PNT MICHAELS/SUBJECT="Build summary of MAIN_TARGET
ending at ''f$time()'"
DELETE CHECK.PNT;
```

The mail message sent to your process looks like the following:

```
From:   MICHAELS     21-FEB-1987 14:48
To:     MICHAELS
Subj:      Build summary of MAIN_TARGET ending at 21-FEB-1987 14:48:06.85

Completed build of TEST1.EXE at 21-FEB-1987 14:47:32.99
Completed build of TEST2.EXE at 21-FEB-1987 14:47:49.65
Completed build of TEST3.EXE at 21-FEB-1987 14:48:06.33
```

## 3.8 Creating and Using Time Stamps

You can use MMS to create time stamps for such purposes as tracking the progress of the system and determining whether any sources have changed since the last time the system was built.

You can use either DCL symbols or included files to create a time stamp.

## 3.8.1 Creating a Time Stamp File Using DCL Symbols

The following description file creates the file CMSMODS.RPT, which reports the number of modified sources by checking replace operations in the CMS library.

```
PROJECT_SOURCES = PARSE.Y, TOUCH.C, GM.C, DRIVE.C, CLP.C, -
                  LEX.C, GRAFBUILD.C, GRAFWALK.C, LFS.C, -
                  MACROBANK.C, MB.C, MMSPRINT.C, UTILS.C, -
                  EXECCMD.C, RULES.C, LBR.C, CMSACCESS.C, -
                  MMSMSG.MSG, FILTER.C, GRAPH.H, GLOBALS.H, -
                  LBRDEF.H, PDEFS.H, TOKEN.H, CLP.H, TC.H

 ! Special CMS filetypes not included by default.
.SUFFIXES : .Y .Y~

 ! New CMS rules (Note: no real CMS fetches occur)
.MSG~.MSG :
     COPY NL: $(MMS$TARGET_NAME).MSG       ! Create the new time stamp file
     PUR $(MMS$TARGET_NAME).MSG            ! Remove the old one, if any
     MODS = MODS + 1       ! Increment the modification counter
.H~.H :
     COPY NL: $(MMS$TARGET_NAME).H
     PUR $(MMS$TARGET_NAME).H
     MODS = MODS + 1

.C~.C :
     COPY NL: $(MMS$TARGET_NAME).C
     PUR $(MMS$TARGET_NAME).C
     MODS = MODS + 1

.Y~.Y :
     COPY NL: $(MMS$TARGET_NAME).Y
     PUR $(MMS$TARGET_NAME).Y
     MODS = MODS + 1
 ! Primary Target
MODS : INIT $(PROJECT_SOURCES)
     IF "''F$SEARCH("CMSMODS.RPT")'" .EQS. "" -
          THEN OPEN/WRITE CHECK CMSMODS.RPT
     IF "''F$SEARCH("CMSMODS.RPT")'" .NES. "" -
           THEN OPEN/APPEND CHECK CMSMODS.RPT
     WRITE CHECK "''MODS' MODIFICATIONS DETECTED AT ''F$TIME()'"
     CLOSE CHECK

INIT :
     MODS = 0
```

CMSMODS.RPT can be used in some form as input to a program that prints a graph of CMS replace operations with relation to a number of days. Such a graph can be used as an indication of how stable a given project's source code is with respect to its milestones.

It is a good idea to run a description file such as the one in this example on a daily or otherwise frequent basis. You may want to put the appropriate MMS command in your LOGIN.COM file.

## 3.8.2 Creating a Time Stamp File Using Included Files

Consider the following directories and files:

```
[DIR1] contains FILE1.X

[DIR2] contains FILE2.Y

[DIR3] contains FILE3.Z
```

MMS can build a file to report changes to these files. The following description file creates the file CHANGES.DOC, which reports when changes were made to the source.

```
.SILENT

RECORD_CHANGE = .INCLUDE CHANGE.REC

REPORT_CHANGE : INIT FILE1.TIM FILE2.TIM FILE3.TIM
     IF "''F$SEARCH("CHANGES.DOC")'" .NES. "" -
          THEN TYPE CHANGES.DOC
     IF "''F$SEARCH("CHANGES.DOC")'" .EQS. "" -
          THEN WRITE SYS$OUTPUT "No changes detected"

INIT :
     IF "''F$SEARCH("CHANGES.DOC")'" .NES. "" -
          THEN DELETE CHANGES.DOC;*/NOLOG

 ! Testing the time stamps
FILE1.TIM : [DIR1]FILE1.X
$(RECORD_CHANGE)

FILE2.TIM : [DIR2]FILE2.Y
$(RECORD_CHANGE)

FILE3.TIM : [DIR3]FILE3.Z
$(RECORD_CHANGE)
```

Because the .SILENT directive suppresses the display of action lines, MMS displays one of two pieces of information when it processes this description file:

- If no changes were made to the files, MMS prints "No changes detected," as instructed in the REPORT_CHANGE action line.
- If changes were made to the files, MMS displays the contents of the file CHANGES.DOC, as instructed in the REPORT_CHANGE action line. CHANGES.DOC lists the files that were changed and the times the changes were made.

CHANGE.REC, the file included by the RECORD_CHANGE macro, is the recording procedure (rule) for making a change. It contains the following actions:

```
     IF "''F$SEARCH("CHANGES.DOC")'" .NES. "" -
          THEN OPEN/APPEND CHANGE CHANGES.DOC
     IF "''F$SEARCH("CHANGES.DOC")'" .EQS. "" -
          THEN OPEN/WRITE CHANGE CHANGES.DOC
     WRITE CHANGE "Changes to $(MMS$SOURCE) noted ''f$time()'"
     CLOSE CHANGE
     COPY NL: $(MMS$TARGET_NAME).TIM
     PURGE $(MMS$TARGET_NAME).TIM
```

You can substitute different recording procedure files for CHANGES.REC without changing the description file every time. To do so, create the same description file described in the example, but omit the RECORD_CHANGE macro. Also, replace the invocations of the RECORD_CHANGE macro with .INCLUDE $(REC_PROC). After these changes, the description file looks like this:

```
.SILENT

REPORT_CHANGE : INIT FILE1.TIM FILE2.TIM FILE3.TIM
     IF "''F$SEARCH("CHANGES.DOC")'" .NES. "" -
          THEN TYPE CHANGES.DOC
     IF "''F$SEARCH("CHANGES.DOC")'" .EQS. "" -
          THEN WRITE SYS$OUTPUT "No changes detected"

INIT :
     IF "''F$SEARCH("CHANGES.DOC")'" .NES. "" -
          THEN DELETE CHANGES.DOC;*/NOLOG

 ! Testing the time stamps
FILE1.TIM : [DIR1]FILE1.X
.INCLUDE $(REC_PROC)

FILE2.TIM : [DIR2]FILE2.Y
.INCLUDE $(REC_PROC)

FILE3.TIM : [DIR3]FILE3.Z
.INCLUDE $(REC_PROC)
```

REC_PROC is a macro that you define on the MMS command line to be the name of a recording procedure file you want to use at the time. Type the following command line to use the file of your choice:

```
$ MMS/MACRO="REC_PROC=@filename"
```

## 3.9 Selectively Deleting Files

Usually after updating your system, you will want to delete the intermediate files from your working directory. Or you might want intermediate files to be deleted automatically after an MMS build. You can accomplish this task in three different ways:

- Create a command procedure.
- Use a macro definition.

- Use the .LAST directive.

The first two methods are described in the following sections. The use of the .LAST directive is described in Section 2.8.11.

## 3.9.1 Creating a Command Procedure to Selectively Delete Files

To use a command procedure to delete files selectively, create the procedure in the description file. Modify the dependencies or the default rules to include the following actions:

```
IF "''F$SEARCH("DELETE.COM")'" .EQS. "" -
     THEN COPY NL: DELETE.COM
OPEN/APPEND DEL_FILE DELETE.COM
     WRITE DEL_FILE "$ DELETE $(MMS$SOURCE);"
```

**NOTE**

Usually, you will want to modify only the .OBJ.OLB rule to include these actions. However, to delete everything, you can modify all the rules you use; take care that you are deleting only those files you want deleted.

The modified .OBJ.OLB rule looks like this:

```
.OBJ.OLB :
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
          THEN $(LIBR)/CREATE $(MMS$TARGET)
     $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
     IF "''F$SEARCH("DELETE.COM")'" .EQS. "" -
          THEN COPY NL: DELETE.COM
     OPEN/APPEND DEL_FILE DELETE.COM
     WRITE DEL_FILE "$ DELETE $(MMS$SOURCE);"
```

Once you have modified the rule, add a target such as the following to your description file:

```
DELETE : MYPROG.EXE  ! The name of the target
     - @DELETE.COM
```

Note that the Ignore action line prefix (-) is used to prevent MMS from aborting execution if it detects errors (such as the absence of files) while deleting files.

To delete .OBJ files that MMS created during a build, you need type only the following:

```
$ MMS/SKIP_INTERMEDIATE DELETE
```

The /SKIP_INTERMEDIATE qualifier causes MMS not to rebuild the files that were deleted.

## 3.9.2 Using a Macro Definition to Selectively Delete Files

There are two ways of using macros for the selective deletion of files:

- Use a macro definition on the MMS command line.
- Use a DCL symbol as a macro.

To use a macro on the command line to delete files, modify the desired rule to include the following action:

```
IF "$(CLEAN)" .NES "" THEN DELETE $(MMS$SOURCE);
```

Thus, the .OBJ.OLB rule looks like this:

```
.OBJ.OLB :
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
          THEN $(LIBR)/CREATE $(MMS$TARGET)
     $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
     IF "$(CLEAN)" .NES "" THEN DELETE $(MMS$SOURCE);
```

The command line is the following:

```
$ MMS/CMS/SKIP/MACRO="CLEAN=CLEAN"
```

You can equate the macro to any character string that you like; MMS simply needs to be able to expand the CLEAN macro to something other than the null string.

To use a DCL symbol as a macro for deleting files, add the same action line to the desired rule as for using a macro on the command line. However, substitute "CLEAN' for $(CLEAN), as follows, where CLEAN is a global CLI symbol:

```
.OBJ.OLB :
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
          THEN $(LIBR)/CREATE $(MMS$TARGET)
     $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
     IF "''CLEAN'" .NES "" THEN DELETE $(MMS$SOURCE);
```

You can use the same macro definition on the command line as in the previous example. If you do not want to define the macro on the command line, make sure that the DCL symbol CLEAN is defined before you invoke MMS. Then the command line can be shortened as follows:

```
$ MMS/CMS/SKIP
```

## 3.10 Doing Parallel Processing

If you have a very large system to build, you can process different parts of it simultaneously by adding rules, such as the following, to the beginning of your existing description file:

```
PARALLEL_PROC : TARG1 TARG2 TARG3  ! Names for parts of your system
      ! Files submitted

TARG1 :
     MMS/CMS/OUT=TARG1.COM/NOACTION PROG.EXE
     SUBMIT $(MMS$TARGET_NAME)

TARG2 :
     MMS/CMS/OUT=TARG2.COM/NOACTION MOD.EXE
     SUBMIT $(MMS$TARGET_NAME)
   .
   .
   .

 ! The rules building the parts of your system
PROG.EXE : PROG.OBJ
     action

MOD.EXE : MOD.OBJ
     action
   .
   .
   .
```

This description file causes MMS to process the parts of your system "in parallel" or simultaneously, resulting in shorter processing time and earlier error detection.

## 3.11 Complex Examples Using MMS

This section demonstrates advanced uses of MMS:

- A description file that uses object libraries
- A description file that results in multiple outputs

### 3.11.1 MMS and Object Libraries

Example 3–4 contains a sample MMS description file using object libraries.

## Example 3–4: Description File Using Object Libraries

```
!
! DESCRIP.MMS
!
!       This command procedure builds the INTERCOM facility.
❶!
!
!+
!       Define macros for the following commands and built-in rules
!
DEBUG               = /noDEBUG
TRACE               = /noTRACE
LIST                = /LIST=LIS$:
BFLAGS              = $(LIST) $(DEBUG) /TERM=STAT
MFLAGS              = $(LIST) $(DEBUG)
CLDFLAGS            = $(LIST)
!LIBRFLAGS          = /LOG
LIBRFLAGS           =
LINKFLAGS           = /FULL $(DEBUG) $(TRACE) /MAP=LIS$:
!
! TNAME gives just the name portion of the target
!
TNAME               = 'F$PARSE("MMS$TARGET_NAME",,,"NAME","SYNTAX_ONLY")
!+
!       Define "built-in" rules
❷!
!
.SUFFIXES       : ;
.SUFFIXES       : .EXE .OLB .OBJ -
                  .B32 .BLI .MAR .CLD .L32 .R32 .REQ .SDL .MSG

.B32.OBJ        : ; $(BLISS) $(BFLAGS) /OBJ=$(MMS$TARGET) $( MMS$SOURCE)
.MSG.OBJ        : ; MESSAGE $(LIST) /OBJ=$(MMS$TARGET) $(MMS$SOURCE)
.MAR.OBJ        : ; $(MACRO) $(MFLAGS) /OBJ=$(MMS$TARGET) $(MMS$SOURCE)
.CLD.OBJ        : ; SET COMMAND /OBJECT=$(MMS$TARGET) $(CLDFLAGS) $(MMS$SOURCE)
.REQ.L32        : ; $(BLISS) /LIBRARY $(BFLAGS) /NOOBJ $(MMS$SOURCE)
```

(continued on next page)

## Example 3–4 (Cont.): Description File Using Object Libraries

```
❸.OBJ.OLB       :       !
        @ IF F$SEARCH(F$PARSE("$(MMS$TARGET)") .NES. F$SEARCH("$(MMS$TARGET)")-
          THEN COPY/LOG $(MMS$TARGET) $(MMS$TARGET)
        @ IF F$SEARCH(F$PARSEF$SEARCH("$(MMS$TARGET)") .EQS. "" -
          THEN $(LIBR)/CREATE/LOG $(MMS$TARGET)
        $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
!+
!       Define groups of OBJs, how modules in the OLB relate to OBJs
!
OLB_ELEMENTS =                                  -
 NOTEFILE=OBJ$:NOTEFILE.OBJ                    ,-
 CLASS=OBJ$:CLASS.OBJ                          ,-
 ENTRY=OBJ$:ENTRY.OBJ                          ,-
 KEYWORD=OBJ$:KEYWORD.OBJ                      ,-
 NOTE=OBJ$:NOTE.OBJ                            ,-
 PROFILE=OBJ$:PROFILE.OBJ                      ,-
 USER=OBJ$:USER.OBJ                            ,-
 CALLABLE_INTERCOM=OBJ$:CALLABLE_INTERCOM.OBJ  ,-
 CALLUSER=OBJ$:CALLUSER.OBJ                    ,-
 PARSEACT=OBJ$:PARSEACT.OBJ                    ,-
 INTERCOMTPU=OBJ$:INTERCOMTPU.OBJ              ,-
 -
 FILEIO=OBJ$:FILEIO.OBJ                        ,-
 HANDLER=OBJ$:HANDLER.OBJ                      ,-
 ITEMSIZE=OBJ$:ITEMSIZE.OBJ                    ,-
 ITEMLIST=OBJ$:ITEMLIST.OBJ                    ,-
 IDPARSE=OBJ$:IDPARSE.OBJ                      ,-
 INTERCOMMSG=OBJ$:INTERCOMMSG.OBJ              ,-
 INTERCOMUTIL=OBJ$:INTERCOMUTIL.OBJ            ,-
 INTERCOM$COMMAND_TABLE=OBJ$:COMMANDS.OBJ      ,-
 INTERCOM$MAIN=OBJ$:INTERCOM$MAIN.OBJ          ,-
 INTERCOM$SERVER=OBJ$:INTERCOM$SERVER.OBJ      ,-
 TFRVEC=OBJ$:TFRVEC.OBJ

!+
!       Define the main targets.  These are put in the kit.
!
MAIN_TARGETS =          -
 OBJ$:INTERCOM$SHARE.EXE, -
 OBJ$:INTERCOM$MAIN.EXE, -
 OBJ$:INTERCOM$SERVER.EXE, -
 OBJ$:INTERCOM$SECTION.GBL, -
 OBJ$:INTERCOM$HELP.HLB, -
 SRC$:INTERCOM_INTERFACE.TPU, -
 SRC$:INTERCOM$STARTUP.COM, -
 SRC$:INTERCOMDCL.CLD
```

(continued on next page)

## Example 3–4 (Cont.): Description File Using Object Libraries

```
!+
!       Define dependency rules.
!
.FIRST                          :
       CONTINUE         ! No initialization needed
!+
!       Specify the main target(s) -- everything.
!       This is the first dependency rule in this file;
!       by calling this target "*", we can say "MMS *" at DCL level.
!
*                       : $(MAIN_TARGETS)
       CONTINUE

!+
!       Define the kit (this is not built by default)
!       Use "MMS OBJ$:INTERCOM_KIT" to build the installation kit.
!
OBJ$:INTERCOM_KIT               : OBJ$:INTERCOM000.A
        CONTINUE
OBJ$:INTERCOM000.A              : $(MAIN_TARGETS), SRC$:KITINSTAL.COM SRC$:SPKITBLD.COM
❹!
       - DELETE NNP$:[INTERCOM.TEMP]*.*.*
       COPY $(MAIN_TARGETS), SRC$:KITINSTAL.COM  NNP$:[INTERCOM.TEMP]
       @SRC$:SPKITBLD.COM $(TNAME) OBJ$: NNP$:[INTERCOM.TEMP]*.*
       SET NOON
       - DELETE NNP$:[INTERCOM.TEMP]*.*.*
       SET ON

!+
!       Build the documents (not built by default)
!
SPECS : OBJ$:INTERCOMPLAN.MEM, OBJ$:INTERCOMSPEC.MEM         !To say "MMS SPECS" at DCL
       CONTINUE

OBJ$:INTERCOMPLAN.MEM         : SRC$:INTERCOMPLAN.RNO
       COPY NL: SYS$SCRATCH:INTERCOMPLAN.RNT         !Create dummy file
       RUNOFF /INTERMEDIATE=SYS$SCRATCH:INTERCOMPLAN /NOOUTPUT SRC$:INTERCOMPLAN
       RUNOFF /CONTENTS /OUTPUT=SYS$SCRATCH:INTERCOMPLAN SYS$SCRATCH:INTERCOMPLAN
       RUNOFF SRC$:INTERCOMPLAN /OUT=OBJ$:INTERCOMPLAN
       SET NOON
       - DELETE SYS$SCRATCH:INTERCOMPLAN.*;*
       SET ON


OBJ$:INTERCOMSPEC.MEM         : SRC$:INTERCOMSPEC.RNO
       COPY NL: SYS$SCRATCH:INTERCOMSPEC.RNT         !Create dummy file
       RUNOFF /INTERMEDIATE=SYS$SCRATCH:INTERCOMSPEC /NOOUTPUT SRC$:INTERCOMSPEC
       RUNOFF /CONTENTS /OUTPUT=SYS$SCRATCH:INTERCOMSPEC SYS$SCRATCH:INTERCOMSPEC
       RUNOFF SRC$:INTERCOMSPEC /OUT=OBJ$:INTERCOMSPEC
       SET NOON
       - DELETE SYS$SCRATCH:INTERCOMSPEC.*;*
       SET ON
```

(continued on next page)

## Example 3–4 (Cont.): Description File Using Object Libraries

```
!+
!       Build the executables and libraries
!
❺LINK_SHR = $(LINK) $(LINKFLAGS) /SHAR=$(MMS$TARGET) $(MMS$SOURCE)/OPT /NODEBU          !
LINK_EXE = $(LINK) $(LINKFLAGS) /EXEC=$(MMS$TARGET) $(MMS$SOURCE)/OPT

OBJ$:INTERCOM$SHARE.EXE       : SRC$:INTERCOM$SHARE.OPT  OBJ$:INTERCOM.OLB ; $(LINK_SHR)
OBJ$:INTERCOM$MAIN.EXE        : SRC$:INTERCOM$MAIN.OPT   OBJ$:INTERCOM.OLB ; $(LINK_EXE)
OBJ$:INTERCOM$SERVER.EXE      : SRC$:INTERCOM$SERVER.OPT OBJ$:INTERCOM.OLB ; $(LINK_EXE)

OBJ$:INTERCOM$SECTION.GBL     : SRC$:INTERCOM_INTERFACE.TPU
       DEFINE /NOLOG /USER INTERCOM$SECTION OBJ$:INTERCOM$SECTION
       - EDIT /TPU /SECTION=EVESECINI /NOJOURNAL /NODISPLAY /COMMAND=$(MMS$SOURCE)

❸OBJ$:INTERCOM$HELP.HLB       : SRC$:INTERCOMHELP.HLP          !
       @ IF F$SEARCH(F$PARSEF$SEARCH("$(MMS$TARGET)")) .NES. F$SEARCHF$SEARCH("$(MMS$TARGET)")-
         THEN COPY/LOG $(MMS$TARGET) $(MMS$TARGET)
       @ IF F$SEARCH(F$PARSEF$SEARCH("$(MMS$TARGET)")) .EQS. "" -
         THEN $(LIBR)/CREATE/LOG/HELP $(MMS$TARGET)
       $(LIBR) $(LIBRFLAGS) /HELP $(MMS$TARGET) $(MMS$SOURCE)

! Anything depending on INTERCOM.OLB is presumed to depend on all its modules.
!
OBJ$:INTERCOM.OLB             : OBJ$:INTERCOM.OLB($(OLB_ELEMENTS))
            CONTINUE

! Any BLISS modules that REQUIRE 'INTERCOMREQ' also depend on INTERCOMLIB.L32,
! since it is referenced by INTERCOMREQ.REQ.  We combine these two in a macro.
!
COM_REQ               = SRC$:INTERCOMREQ.REQ OBJ$:INTERCOMLIB.L32

! .B32 sources
!
OBJ$:NOTEFILE.OBJ          : SRC$:NOTEFILE.B32          $(COM_REQ) OBJ$:CXF.L32
OBJ$:CLASS.OBJ             : SRC$:CLASS.B32             $(COM_REQ) OBJ$:CXF.L32
OBJ$:ENTRY.OBJ             : SRC$:ENTRY.B32             $(COM_REQ) OBJ$:CXF.L32
OBJ$:NOTE.OBJ              : SRC$:NOTE.B32              $(COM_REQ) OBJ$:CXF.L32
OBJ$:KEYWORD.OBJ           : SRC$:KEYWORD.B32           $(COM_REQ) OBJ$:CXF.L32
OBJ$:PROFILE.OBJ           : SRC$:PROFILE.B32           $(COM_REQ) OBJ$:CXF.L32
OBJ$:USER.OBJ              : SRC$:USER.B32              $(COM_REQ) OBJ$:CXF.L32
OBJ$:FILEIO.OBJ            : SRC$:FILEIO.B32            $(COM_REQ)
OBJ$:HANDLER.OBJ           : SRC$:HANDLER.B32           $(COM_REQ)
OBJ$:ITEMLIST.OBJ          : SRC$:ITEMLIST.B32          $(COM_REQ)
OBJ$:ITEMSIZE.OBJ          : SRC$:ITEMSIZE.B32          $(COM_REQ)
OBJ$:INTERCOMUTIL.OBJ      : SRC$:INTERCOMUTIL.B32      $(COM_REQ)
OBJ$:IDPARSE.OBJ           : SRC$:IDPARSE.B32           $(COM_REQ)
OBJ$:CALLUSER.OBJ          : SRC$:CALLUSER.B32          $(COM_REQ) OBJ$:USERDEF.L32
OBJ$:PARSEACT.OBJ          : SRC$:PARSEACT.B32          $(COM_REQ) OBJ$:USERDEF.L32
OBJ$:INTERCOMTPU.OBJ       : SRC$:INTERCOMTPU.B32       $(COM_REQ) OBJ$:USERDEF.L32
OBJ$:INTERCOM$MAIN.OBJ     : SRC$:INTERCOM$MAIN.B32     $(COM_REQ)
OBJ$:INTERCOM$SERVER.OBJ   : SRC$:INTERCOM$SERVER.B32   $(COM_REQ)
OBJ$:CALLABLE_INTERCOM.OBJ : SRC$:CALLABLE_INTERCOM.B32 $(COM_REQ) OBJ$:USERDEF.L32
```

(continued on next page)

**Example 3–4 (Cont.): Description File Using Object Libraries**

```
! .MAR sources
!
OBJ$:TFRVEC.OBJ                 : SRC$:TFRVEC.MAR

! .REQ sources
!
OBJ$:CXF.L32              : SRC$:CXF.REQ            OBJ$:INTERCOMLIB.L32
OBJ$:USERDEF.L32          : SRC$:USERDEF.REQ        OBJ$:INTERCOMLIB.L32
!
! Several require files are combined to form a single BLISS library
!
OBJ$:INTERCOMLIB.L32        : SRC$:INTERCOMTRUC.REQ SRC$:INTERCOMMAC.REQ SRC$:RTN.REQ -
                          OBJ$:INTERCOMMSG.R32 OBJ$:NOTEITEMS.R32
               $(BLISS) /LIBRARY $(BFLAGS) /NOOBJ /NOLIST /LIBR=$(MMS$TARGET)-
                          SRC$:INTERCOMTRUC.REQ+SRC$:INTERCOMMAC.REQ+SRC$:RTN.REQ+-
                          OBJ$:INTERCOMMSG.R32+OBJ$:NOTEITEMS.R32

! .MSG source
!
OBJ$:INTERCOMMSG.OBJ        : SRC$:INTERCOMMSG.MSG

! .CLD sources
!
OBJ$:COMMANDS.OBJ         : SRC$:COMMANDS.CLD
```

❶ The following logical names are used in this MMS file:

| | |
|---|---|
| SRC$ | Directory containing all sources that can be modified |
| OBJ$ | Directory containing all machine-produced files |
| LIS$ | Directory containing listing files |

A simple command procedure is used to define these logical names. These can refer to the group-wide source directory or can be defined as a search list for a local copy of a module instead of the group-wide or shared module. The logical names were defined in either of the following formats:

```
$ DEFINE SRC$ NNP$:[INTERCOM.SRC]

$ DEFINE SRC$ NNP$:[HILT.INTERCOM.SRC], NNP$:[INTERCOM.SRC]
```

No CMS elements are explicitly mentioned in the MMS description file. Instead, NNP$:[INTERCOM.SRC] is specified as a CMS reference directory (see HELP CMS MODIFY LIBRARY /REFERENCE_COPY). Any module replaced in the CMS library causes a new version of the file to be put in this directory.

All file references include the file directory to allow the MMS file to be used correctly from any default directory. Similarly, dependencies are explicitly described.

❷ Only the necessary suffixes were included in the suffixes list. This together with explicit specification of the dependencies should help MMS improve performance.

❸ The rules for building an .OLB from .OBJ files (or an .HLB from .HLP files) takes search-lists into account. Only the first directory in the search-list is used to store the new .OBJs.

The default actions were redefined to allow you to use the entire source and target specifications instead of just the file name. For example, /OBJ=OBJ$:CLASS.OBJ was used instead of just /OBJ=CLASS.

The first action line checks whether local .OLB is different from the first .OLB found by the search list. If so, the .OLB is copied to the local directory. The second line tests whether the .OLB really exists, and the third inserts the .OBJ into the .OLB. The first two lines are prefixed by @, so they are not echoed in the log files.

❹ The SPKITBLD command procedure builds a VMS installation kit. Creating an empty, temporary subdirectory as a staging area is necessary because SPKITBLD uses BACKUP to create the save-set. BACKUP would copy all versions instead of only the highest versions of the files.

❺ Options files are used to build all the executable images. The LINK_EXE and LINK_SHR macros are used to specify the actions and the qualifiers to produce the .EXE file. LINK_SHR overrides any /DEBUG qualifier that might be specified in LINKFLAGS.

## 3.11.2 Producing Multiple Outputs with MMS

Using MMS to describe and build systems that have actions with more than one output becomes complicated because MMS cannot express multiple output dependencies. Most actions involved in building a system have a single input and a single output. For example:

```
ABC.OBJ DEPENDS_ON ABC.FOR
        FORTRAN/NOLIST/OBJECT=ABC ABC.FOR
```

This example contains the action line that uses the FORTRAN compiler to produce an object file from a FORTRAN source file.

MMS can also describe cases in which multiple inputs are present. For example:

```
ABC.OBJ DEPENDS_ON ABC.FOR DEF.TXT
        FORTRAN/NOLIST/OBJECT=ABC ABC.FOR
```

This example contains a source file, ABC.FOR, which contains an include file, DEF.TXT. Both files are needed to produce the object file, ABC.OBJ.

However, MMS syntax cannot express the case in which multiple outputs are needed. For example, if you want to produce a listing of the compilation in the first example, you could use the following dependency and action lines.

```
ABC.LIS ABC.OBJ DEPENDS_ON ABC.FOR
        FORTRAN/LIST=ABC/OBJECT=ABC ABC.FOR
```

This example does produce the correct results, in that valid listing and object files will be produced, but it does not produce the results correctly. The previous example is really a shorthand notation for the following:

```
ABC.LIS DEPENDS_ON ABC.FOR
        FORTRAN/LIST=ABC/OBJECT=ABC ABC.FOR

ABC.OBJ DEPENDS_ON ABC.FOR
        FORTRAN/LIST=ABC/OBJECT=ABC ABC.FOR
```

We can assume that both the listing and object target appear as sources elsewhere in the description file. When you invoke MMS, both targets are built but the action line is triggered twice. This example produces the listing and object files redundantly and, therefore, violates the principle of minimal action of MMS.

### 3.11.2.1 When Outputs Are Independent

In the previous example, you may consider the object and the listing files as independent. The compiler produces them independently and the revision time of the files is different. When the outputs are independent, it seems safe to produce them independently. You might describe their relationship in the following description file:

```
ABC.LIS DEPENDS_ON ABC.FOR
        FORTRAN/LIST=ABC/NOOBJECT=ABC ABC.FOR

ABC.OBJ DEPENDS_ON ABC.FOR
        FORTRAN/NOLIST/OBJECT=ABC ABC.FOR
```

This example solves the problem of producing two listings and does produce correct results because no actions on listing and object files are sensitive to whether the files were created by the same operation. However, if a configuration change occurs (for example, the compiler is updated), and the object file ABC.OBJ is missing, then invoking MMS with this description file results in a new object file but no new listing file. This demonstrates that the files are not independent. You would normally expect that listing and object files are produced by the same operation. The files are consistent as long as they convey the same information—for example, compiler version indentification.

### 3.11.2.2 When Outputs Are Dependent

The dependence of outputs is demonstrated clearly by the environment files and object files produced from a PASCAL source file. Consider the following two PASCAL source files, A.PAS and B.PAS:

```
{A.PAS}
[ INHERIT('b') ] PROGRAM a ;
BEGIN
bproc
END.

{B.PAS}
MODULE b ;
PROCEDURE bproc;
    BEGIN
    END;
END.
```

When compiled and linked, these modules produce the executable image, AB.EXE. Consider the dependencies in the following description file:

```
AB.EXE DEPENDS_ON A.OBJ B.OBJ
       LINK/EXECUTABLE=AB.EXE A.OBJ, B.OBJ

A.OBJ DEPENDS_ON A.PAS B.PEN
       PASCAL/OBJECT=A A.PAS

B.OBJ DEPENDS_ON B.PAS
       PASCAL/ENVIRONMENT=B/OBJECT=B B.PAS
```

This description file should work correctly but it is flawed. The file, B.PEN, never appears as a target. It is created when B.PAS is compiled and is then used as a source for creating A.OBJ. The description file would be more accurate if the B.OBJ dependency rule is changed as follows:

```
B.OBJ B.PEN DEPENDS_ON B.PAS
        PASCAL/ENVIRONMENT=B/OBJECT=B B.PAS
```

This example does produce a redundant compilation, but at least the description file is complete and consistent. However, under certain circumstances, the system does not link correctly because the linker checks that all references to an environment file are consistent. The linker does the checking with an "entity identification check," which is inserted into object files that refer to an environment file. Because you cannot predict the order of compilation in all circumstances, MMS can produce object files referring to different environment files (with different identifications). The linker then sends the warning message ENTIDMTCH.

If you attempt to produce the environment file separately, you may build the system incorrectly. For example, consider the following description file:

```
B.OBJ DEPENDS_ON B.PAS
        PASCAL/NOENVIRONMENT=B/OBJECT=B B.PAS

B.PEN DEPENDS_ON B.PAS
        PASCAL/ENVIRONMENT=B/OBJECT=B B.PAS
```

In this example, an object file produced with the /NOENVIRONMENT qualifier does not contain the entity indentification check, and therefore, strong typing across modules is defeated.

## 3.11.3 Multiple Outputs Work-Around

Because you cannot safely express the idea of multiple outputs in the source-target part of the dependency rule, you can modify the action line to produce safe results. It is safest to proceed from the redundant compilation description file. If you can avoid the extra compilation, then the description file is complete and the resulting system is built correctly and with a minimum of actions.

You can introduce a context variable into the action block to monitor whether the compilation has or has not been performed. Consider the following description file:

```
AB.EXE DEPENDS_ON A.OBJ B.OBJ
       LINK/EXECUTABLE=AB.EXE A.OBJ, B.OBJ

A.OBJ DEPENDS_ON A.PAS B.PEN
       PASCAL/OBJECT=A A.PAS

B.OBJ B.PEN DEPENDS_ON B.PAS
       IF F$TYPE(B_COMPILED)  .EQS. "" THEN B_COMPILED=0
       IF .NOT. B_COMPILED THEN-
         PASCAL/ENVIRONMENT=B/OBJECT=B B.PAS
       IF .NOT. B_COMPILED THEN-
         B_COMPILED=1
```

In this example, B.OBJ and B.PEN are always produced by the same operation. However, the case in which B.OBJ is missing still generates an inconsistency. You can avoid this only by introducing false information into the A.OBJ dependency rule, as follows:

```
A.OBJ DEPENDS_ON A.PAS B.PEN B.OBJ
     PASCAL/OBJECT=A A.PAS
```

In this example, MMS can guarantee that both B.OBJ and B.PEN must exist, and because of their coproduction, they are simultaneously updated. The net effect of this work-around is to treat B.OBJ and B.PEN as one entity.

# Chapter 4

# Accessing Libraries with MMS

This chapter describes how you can specify sources and targets that are stored in libraries. The following sections describe how MMS can access or update information in:

- VMS libraries created with the LIBRARY utility
- VAX DEC/Code Management System (CMS) libraries
- VAX FMS libraries
- The VAX Common Data Dictionary (CDD)
- VAX Source Code Analyzer (SCA) libraries

## 4.1 Creating and Accessing Files in VMS Libraries

You can use MMS to access files that are contained in VMS libraries and to create library files using certain built-in rules. The built-in rules that MMS uses to create library files are listed in Table C–7 in Appendix C. These rules tell MMS to create the specified library if one does not already exist; they then cause MMS to replace the source module in the target library.

**NOTE**

You cannot use MMS to access modules in an RSX library because only a module's revision date is recorded, not its revision time.

### 4.1.1 Formatting Library Module Specifications

To specify that a source or target in a dependency rule is a module in a VMS library, use the following format. Avoid adding any spaces or tabs; they can cause problems in processing.

```
library(module[=filespec], . . . )
```

The library is a VMS file specification that denotes a library. The default file type is .OLB if you are referring to a module within the library. If you are referring to the entire library, there is no default file type. The module is the name of the module in the library. The filespec is the VMS file specification that corresponds to the module in the library. The default file type depends on the file type of the library.

The format shown in this section describes how to refer to modules within a library. You can also use MMS to process the library file itself simply by providing the library file specification (for example, CRTLIB.OLB).

There is a restriction in using library module specifications as targets in a double-colon dependency rule. See Section 3.1 for more information.

### 4.1.2 Using Logical Names in a Library Module Specification

You can use a logical name for the name of the module if you supply the action lines that update the target. MMS cannot apply its built-in rules to a logical name because it relies on file types to determine which built-in rule is appropriate.

For example, CRTLIB(C$STRLEN=STRLEN.OBJ) designates that the module C$STRLEN is in library CRTLIB.OLB; C$STRLEN is found in the file named STRLEN.OBJ.

If you use a logical name as the directory name of an object library, that logical name must not translate to another logical name because the built-in rules that MMS uses to update libraries are defined in terms of the DCL function F$SEARCH. F$SEARCH translates only one level of logical name.

### 4.1.3 Specifying Multiple Libraries

You can specify multiple modules in the same library in two ways:

- You can enclose the module names in one set of parentheses and separate them with commas. For example, to refer to three modules in the library CRTLIB.OLB, you can specify CRTLIB(C$STRLEN=STRLEN.OBJ, C$STRPAD=STRPAD.OBJ, C$STRIND=STRIND.OBJ).
- You can use the VMS * and % wildcard characters. For example, the specification CRTLIB(C$*) directs MMS to look for all modules in the library CRTLIB.OLB whose names begin with the characters C$. In the same way, the specification CRTLIB(C$STR%) directs MMS to look for all modules in CRTLIB.OLB whose names begin with the characters C$STR followed by only one character.

### 4.1.4 Accessing Library Modules with Non-VMS File Specifications

You can use a complete library specification when you need to access library modules whose names do not correspond to VMS file specifications (for example, C$STRLEN=STRLEN.OBJ). However, MMS can interpret shorter specifications as follows:

- If the module's name in the library is the same as its file name, you can provide just the module name in parentheses after the library name. For example, if the module in the previous example were named STRLEN, you could refer to the module in the library as CRTLIB(STRLEN).
- If the module's file type is associated by default with the type of the library, you can omit the file type. In the specification CRTLIB(STRLEN), MMS assumes that the file name is STRLEN.OBJ, because .OLB libraries are assumed to contain .OBJ modules. If the module's file type is something other than the default for that kind of library, you must supply the file type. For example, if the module were STRLEN.C, you would have to specify CRTLIB(STRLEN.C). MMS would then expand this specification to the following two dependencies:

```
CRTLIB(STRLEN=STRLEN.OBJ) : STRLEN.OBJ

STRLEN.OBJ : STRLEN.C
```

### 4.1.5 Using Special Macros with Library Specifications

When used with library specifications, certain MMS special macros have slightly different meanings:

- If a library is the source in a dependency rule, MMS$SOURCE expands to the complete specification of the module in the library. For example, in the specification CRTLIB(C$STRLEN=STRLEN), MMS$SOURCE expands to CRTLIB.OLB(C$STRLEN=STRLEN.OBJ).
- If a library is the target in a dependency rule, MMS$TARGET expands to the name of the library. For example, in the specification CRTLIB(C$STRLEN), MMS$TARGET becomes CRTLIB.
- If a library is the target in a dependency rule, MMS$TARGET_NAME expands to the module name (without its file type). For example, if the library module is STRLEN, MMS$TARGET_NAME expands to STRLEN.

In addition to these MMS special macros, there is another special macro, MMS$LIB_ELEMENT, which you can use only in library specifications. MMS$LIB_ELEMENT expands to the string between parentheses, that is, to the module name and its corresponding file specification. For example, in the specification CRTLIB(STRLEN), MMS$LIB_ELEMENT becomes STRLEN=STRLEN.OBJ. The expansion of MMS$LIB_ELEMENT does not depend on whether the library is the source or the target in a dependency rule. If the library is specified as both the source and the target, then the target is expanded.

### 4.1.6 Using Libraries as a Source

The use of libraries as a source is illustrated in the following example. Suppose your description file contains the following dependency rules:

```
TOOLTITLE.EXE : SYS$LIBRARY:CRTLIB.OLB, USER$:[WATKINS]FLIB.OLB
    LINK TOOLTITLE.OBJ, SYS$LIBRARY:VAXCRTL/LIB, USER$:[WATKINS]FLIB/LIB

TOOLTITLE.OBJ : SMGTEXLIB.TLB(SMGDEF=SMGDEF.H)
    CC TOOLTITLE + SMGTEXLIB/LIB
```

TOOLTITLE.C is a C program that includes the file SMGDEF.H, which is stored in the text library SMGTEXLIB.TLB under the name SMGDEF. The action line in the second dependency rule invokes the C compiler to compile TOOLTITLE.C and the text library. You must state this action line explicitly. In this case specifying only the target and source is not sufficient. The first dependency rule invokes the VAX Linker to link TOOLTITLE.OBJ with one system library and one user library.

## 4.2 Using MMS with CMS

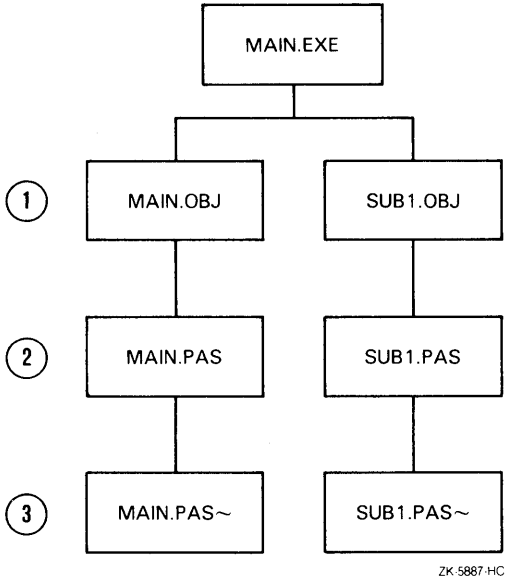
If the VAX DEC/Code Management System (CMS) is installed on your system, you can use MMS to access elements in CMS libraries. CMS elements are denoted by the tilde ( ~ ). You should be familiar with CMS before you read this section.

MMS provides default macros and special macros tailored for use with CMS. Appendix C lists the default macros and the special macros.

MMS can build your system software from source code files stored in CMS libraries. MMS fetches the source code file from its library, compiles the code into object files, then links the object files into executable images. MMS treats the CMS library as the *source* for your source code files.

MMS treats the source code in your directory and an element in a CMS library, just as it does any other source or target pair. If the source code file in the default directory is missing, MMS fetches that element from its CMS library. If the source code file in the default directory is older than the library element, MMS fetches the element. Figure 4–1 illustrates a software system using CMS libraries.

**Figure 4–1: A Software System Using CMS Libraries**




❶ The executable file depends on the object files.

❷ The object files depend on the sources in the your directory.

❸ The sources in your directory depend on elements in CMS libraries.

MMS supports only one CMS qualifier: /GENERATION. The default macro CMSFLAGS expands to the /GENERATION qualifier. You can also specify /GENERATION and a generation number after the tilde ( ~ ) in the element name, as shown in this example:

```
PROG.OBJ : [OTHER.CMS]PROG.C~/GEN=4A1
```

The tilde format for an explicit reference to a CMS element is useful when you are certain that the most up-to-date source is stored in the CMS library. A more convenient use of MMS with CMS is to let MMS determine where the newest source is located and to fetch the CMS element automatically, if necessary. To take advantage of this feature, you must use the /CMS qualifier on the MMS command line and you should not use the tilde format for specifying the source.

### 4.2.1 Using CMS Commands in a Description File

You can use any CMS command in an MMS description file.

As MMS examines target and source lines in your main process, it uses the CMS$LIB logical name to establish the CMS directory from which sources will be fetched if the target must be updated. If you use the CMS SET LIBRARY command in an action line, that command is executed in the subprocess where MMS normally executes action lines. Such an action establishes a new library as the current default for any subsequent action lines that execute CMS commands; however, the value of CMS$LIB is not changed in the main process because the CMS SET LIBRARY command is executed in the subprocess. MMS still looks for sources in the library represented by CMS$LIB.

The MMS qualifier /REVISE_DATE has no effect when MMS is accessing elements in CMS libraries.

### 4.2.2 Automatic Access of CMS Elements from Dependency Rules

The /CMS qualifier directs MMS to look for sources in the current default CMS library, as well as in the directories specified in the description file. If the CMS element has been replaced in the library since the file in the specified directory was last revised, MMS directs CMS to fetch the source from the library so that the target can be rebuilt. MMS has built-in rules that instruct CMS how to find the correct source (see Table C–9 for the built-in CMS rules). MMS uses the sources fetched from CMS to update the target by executing the action lines in the description file. The CMSFLAGS default macro determines which generation of an element is fetched as the source or from which class the source element is fetched.

If the file in the specified directory is newer than the CMS element, MMS uses that file. Therefore, you could edit a source in your directory and build a new system with the edited source, rather than with the corresponding CMS element.

For example, consider a description file that contains the following dependencies:

```
A.EXE : A.OBJ

A.OBJ : A.PAS
```

If you invoke MMS with the /CMS qualifier, MMS processes the description file by looking in the current default CMS library for A.PAS. If it locates that source, it compares the revision time with the revision time of A.PAS in the current directory (if A.PAS exists there). If the CMS element is newer, MMS uses it to update A.OBJ.

The CMSFLAGS default macro always fetches the most recent generation of an element on the main line of descent. You can redefine CMSFLAGS to indicate a specific element generation or the element generation that belongs to a particular class. However, if you do so, you must be aware that if newer generations exist in the library, they will not be fetched; MMS will check the time of the element designated by the CMSFLAGS macro against the time of the file in your directory. If the file is newer, MMS will use it even though more recent generations of the element may exist in the library.

The /NOCMS qualifier directs MMS not to look automatically for sources in the current default CMS library; /NOCMS is the default. The /CMS and /NOCMS qualifiers are described in full in the Command Dictionary.

### 4.2.3 Explicit References to CMS Elements in Dependency Rules

The /CMS qualifier causes MMS to compare the times of a CMS element and a file in the specified directory, if both exist. You can also direct MMS to check only the CMS element by putting a tilde immediately after the source file name in a dependency rule. For example, the tilde in the following target or source line directs MMS to look for the source PROG.C in the current default CMS library:

```
PROG.OBJ : PROG.C~
```

If you use the tilde format to indicate CMS elements, you can specify only one element in a given dependency rule. You cannot specify a list of CMS elements if their file specifications are followed by tildes.

If the element is in a CMS library other than the current default library, you must type the library specification before the element name:

```
PROG.OBJ : [OTHER.CMS]PROG.C~
```

You may not be able to access elements in a CMS library that reside on a DECnet node other than your own.

### 4.2.4 Building the System Using CMS Elements

The following example demonstrates how to build your software system with CMS elements. Consider the following description file CMS_MMS.MMS:

**Example 4–1: Description File Using CMS Libraries**

```
$ TYPE CMS_MMS.MMS

  !
  ! Executable target
  !

❶   MAIN.EXE : MAIN.OBJ, SUB1.OBJ
      LINK $(MMS$SOURCE_LIST)

  !
  ! Object files and their sources
  !

❷   MAIN.OBJ : MAIN.PAS
  SUB1.OBJ : SUB1.PAS

  !
  ! Where the sources are stored
  !

❸ MAIN.PAS : DISK1:[SYSTEM2_LIB]MAIN.PAS~
  SUB1.PAS : DISK1:[SYSTEM2_LIB]SUB1.PAS~
```

❶ The executable target is stated.

❷ The objects or targets are stated.

❸ Each source code file has a target or source line. The element in the CMS library is the source.

You can build your system with the description file CMS_MMS.MMS as shown in Example 4–2.

## Example 4–2: Building a System from CMS Library Elements

```
❶ $ DIR/DATE=MODIFIED
Directory DISK1:[TEST]
CMS_MMS.MMS;1            30-JUL-1987 13:10

Total of 1 file.

❷ $ MMS/DESCRIPTION=CMS_MMS

  mms$cmslib :== 'f$logical("CMS$LIB")
  IF mms$cmslib .nes. "DISK1:[SYSTEM2_LIB]" THEN
  CMS SET LIBRARY DISK1:[SYSTEM2_LIB]
❸ CMS FETCH MAIN.PAS /GEN=1+ ""
 %CMS-S-FETCHED, generation 1 of element MAIN.PAS fetched
  IF mms$cmslib .EQS. "" THEN CMS SET LIBRARY 1234
  IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM2_LIB]"
      THEN CMS SET LIBRARY 'mms$cmslib'
❹ PASCAL /NOLIST/OBJECT=MAIN MAIN.PAS
 mms$cmslib :== 'f$logical("CMS$LIB")
 IF mms$cmslib .nes. "DISK1:[SYSTEM2_LIB]" THEN
   CMS SET LIBRARY DISK1:[SYSTEM2_LIB]
❺ CMS FETCH SUB1.PAS /GEN=1+ ""
 %CMS-S-FETCHED, generation 1 of element SUB1.PAS fetched
 IF mms$cmslib .EQS. "" THEN CMS SET LIBRARY 1234
 IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM2_LIB]"
   THEN CMS SET LIBRARY 'mms$cmslib'
❻ PASCAL /NOLIST/OBJECT=SUB1 SUB1.PAS
❼ LINK MAIN.OBJ, SUB1.OBJ

❽ $ DIR/DATE=MODIFIED

Directory DISK1:[TEST]

MAIN.EXE;1               30-JUL-1987 13:19
MAIN.OBJ;1               30-JUL-1987 13:19
MAIN.PAS;1               30-JUL-1987 13:10
SUB1.OBJ;1               30-JUL-1987 13:19
SUB1.PAS;1               30-JUL-1987 13:10
CMS_MMS.MMS;1            30-JUL-1987 13:10

Total of 6 files.
```

❶ The default directory contains only the software description.

❷ MMS is invoked using the CMS_MMS.MMS description file.

❸ MMS fetches the missing source code file MAIN.PAS from CMS. Note that MMS fetches the latest generation of each element by using the 1+ notation.

❹ MMS compiles the source code file MAIN.PAS.

❺ MMS fetches missing source code file SUB1.PAS from CMS.

❻ MMS compiles the source code file SUB1.PAS.

❼ MMS links the object files using the action line.

❽ The default directory has a complete software system.

MMS fetches sources from CMS if the source code file in your directory is missing or older. In this example, only the description file is in the default directory before you invoke MMS. There is no source code to compile in your default directory, but because the description file states the source of each source code file, MMS fetches the source code files from CMS.

Because of the MMS built-in rule for accessing CMS libraries, MMS generates more output when it fetches source files from CMS than when it builds a system with files from your directory. The extra actions reflected in the output are concerned with checking that the CMS library is set to the correct place before and after the fetch in case the source code is stored in more than one library.

Appendix C contains a full list of extensions that MMS must know to fetch files from a CMS library and to access other libraries.

## 4.2.5 Rebuilding the System Using CMS Libraries

Rebuilding a system from a CMS library is similar to all other system rebuilding in that MMS first checks that all parts of the system are present and up-to-date and then recreates executable files and object files that are old or missing. MMS also uses the CMS library as the source to update old and missing source code.

Example 4–3 is a real example of what can happen during the software development cycle. You reserve an element, fix a bug in the element, and test the fix before replacing the code element in the library. No one else has updated the library element since the last system build.

### Example 4–3: Rebuilding Using CMS Libraries

```
❶ $ DIR/DATE=MODIFIED

Directory DISK1:[TEST]

MAIN.EXE;1            30-JUL-1987 13:15
MAIN.OBJ;1            30-JUL-1987 13:13
MAIN.PAS;1            30-JUL-1987 13:10
SUB1.OBJ;1            30-JUL-1987 13:13
SUB1.PAS;1            30-JUL-1987 13:10
CMS_MMS.MMS;1         30-JUL-1987 13:10

Total of 6 files.

$ CMS SET LIBRARY DISK1:[SYSTEM2_LIB]
%CMS-I-LIBIS, CMS library is DISK1:[SYSTEM2_LIB]

❷ $ CMS RESERVE SUB1.PAS "Fixing a bug in stack handler"
%CMS-S-RESERVED, generation 1 of element SUB1.PAS reserved

❸ $ EDIT SUB1.PAS

$ PURGE
❹ $ DIR/DATE=MODIFIED

Directory DISK1:[TEST]

MAIN.EXE;1            30-JUL-1987 13:15
MAIN.OBJ;1            30-JUL-1987 13:13
MAIN.PAS;1            30-JUL-1987 13:10
SUB1.OBJ;1            30-JUL-1987 13:13
SUB1.PAS;3            30-JUL-1987 13:17
CMS_MMS.MMS;1         30-JUL-1987 13:10

Total of 6 files.

❺ $ MMS/DESCRIPTION=CMS_MMS

❻ PASCAL /NOLIST/OBJECT=SUB1 SUB1.PAS
  LINK MAIN.OBJ, SUB1.OBJ
```

❶ You have a complete, up-to-date system from a previous system build.

❷ You reserve a CMS element.

❸ You modify SUB1.PAS to fix a bug.

❹ One source code file SUB1.PAS is newer than its object SUB1.OBJ.

❺ You invoke MMS to rebuild the system.

❻ MMS rebuilds the system from files in your default directory fetching nothing from the library. MMS compiles SUB1.PAS and links MAIN.OBJ and SUB1.OBJ.

When you invoke MMS in Example 4–3, MMS does not fetch any files from the CMS library. Nothing in your directory is missing and nothing is older than its library element.

However, in the next example, you have a complete, up-to-date copy of the entire system in your directory. However, another person has updated one of the code files in the CMS library after you built your copy of the system. When you invoke MMS to see if your system is up-to-date, MMS detects the newer file in the CMS library and fetches the updated file from the library. It rebuilds the system using the newly fetched file. This demonstrates how MMS uses the CMS library as the source of your targets. Just as an object file is rebuilt if the code has changed, your file is updated if the library has changed.

For example,

```
❶ $  DIR/DATE=MODIFIED

Directory DISK1:[TEST]

MAIN.EXE;2            30-JUL-1987 13:15
MAIN.OBJ;1            30-JUL-1987 13:13
MAIN.PAS;1            30-JUL-1987 13:10
SUB1.OBJ;2            30-JUL-1987 13:13
SUB1.PAS;3            30-JUL-1987 13:10
CMS_MMS.MMS;1         30-JUL-1987 13:10

Total of 6 files.

❷ $ ! (Another user reserves, changes and replaces MAIN.PAS)

❸ $  MMS/DESCRIPTION=CMS_MMS

mms$cmslib :== 'f$logical("CMS$LIB")
IF mms$cmslib .nes. "DISK1:[SYSTEM2_LIB]" THEN
  CMS SET LIBRARY DISK1:[SYSTEM2_LIB]
❹ CMS FETCH MAIN.PAS /GEN=1+ ""
%CMS-I-FILEXISTS, file already exists, DISK1:[TEST]MAIN.PAS;2 created
%CMS-S-FETCHED, generation 2 of element MAIN.PAS fetched
IF mms$cmslib .EQS. "" THEN CMS SET LIBRARY 1234
IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM2_LIB]"
  THEN CMS SET LIBRARY 'mms$cmslib'
❺ PASCAL /NOLIST/OBJECT=MAIN MAIN.PAS
LINK MAIN.OBJ, SUB1.OBJ
```

❶ Your directory has a complete, up-to-date system.

❷ Another person updates a file in the project's CMS library.

❸ You invoke MMS to update your system.

❹ MMS fetches the newer source code in the library.

❺ MMS compiles and links the newer source code.

When creating the "official" release of a software product, you want to be sure that the release is built from code in the library, not from test code that you may have in your default directory. It is better to create a [RELEASE] directory that is used only for building the system from its CMS libraries. No other operations are performed in that directory.

## 4.2.6 Building a System from a Specified CMS Class

You can build your system from a specified CMS class. Building with a class specifier in CMS is identical to building from current generations in CMS.

In building a system from a specified CMS class, MMS still uses the CMS elements as the sources but it uses the designated class of generations, not necessarily the current generations. MMS allows you to find and modify old source code and recreate previous versions of your system by building from a specific CMS class.

MMS looks for the description file on the main line of descent unless you override the default macro CMSFLAGS. If you specify /MACRO="CMSFLAGS=/GENERATION=class-name", MMS instead uses the specified class. If MMS cannot find a description file in either your default directory or the CMS library, it aborts execution.

You can use a user-defined macro to control the class that MMS fetches. The user-defined macro is used as a qualifier to one of MMS's actions. For example, you define the macro CMSFLAGS, which contains the qualifiers MMS uses when it fetches an element from a CMS library. A /GENERATION qualifier on this macro causes MMS to fetch files from a certain class. Consider the command procedure BUILD_CLASS.COM and the description file SYSTEM3.MMS in Example 4–4.

### Example 4–4: Description File for Building from a CMS Class

```
❶ $  TYPE BUILD_CLASS.COM

$ !
$ ! Command procedure to build any CMS class of SYSTEM2.
$ !
$ ! Create the user defined macros we want
$ !
❷ $ INQUIRE CLASS "Enter name of class to build"
❸ $ CMSFLAGS = "/GENERATION=" + CLASS
$ !
$ ! Invoke MMS and direct it to use our macros
$ !
❹ $ MMS /DESCRIPTION=SYSTEM3 /OVERRIDE
$ !

❺ $ TYPE SYSTEM3.MMS
!
! Executable target
!
```

(continued on next page)

**Example 4–4 (Cont.): Description File for Building from a CMS Class**

```
MAIN.EXE : MAIN.OBJ, SUB1.OBJ, SUB2.OBJ
    LINK $(LINKFLAGS) $(MMS$SOURCE_LIST)
!
! Object files and their sources
!
MAIN.OBJ : MAIN.PAS
SUB1.OBJ : SUB1.PAS
SUB2.OBJ : SUB2.PAS
!
! Where the sources are stored
!
MAIN.PAS : DISK1:[SYSTEM3_LIB]MAIN.PAS~
SUB1.PAS : DISK1:[SYSTEM3_LIB]SUB1.PAS~
SUB2.PAS : DISK1:[SYSTEM3_LIB]SUB2.PAS~
```

❶ The command procedure invokes MMS.

❷ The command procedure prompts you for the name of CMS class to build.

❸ The CMSFLAGS macro (which is used by the built-in rule for fetching your source files) is set and the /GENERATION qualifier is added to the class name.

❹ The procedure invokes MMS with the /OVERRIDE qualifier so that your macro is used instead of the default.

❺ The description file lists where the source files are stored in the CMS library.

You can insert the MMS description file and its calling command procedure into the appropriate CMS class. In this way, the description file in a given class builds that class. As a system changes, you can continue to put a copy of the description file in each new class you create.

## 4.2.7 Building a System from a Previous Class

In this example, you build your system from a previous class in a directory that is empty except for the command procedure and the description file. MMS fetches files from CMS only if the library copy is newer than your copy. The files in the previous class are certain to be older than your code, so MMS does not fetch them unless you build the previous releases in a clean directory. For example, consider the system build shown in Example 4–5.

## Example 4–5: Building a System from a Previous CMS Class

```
❶ $ DIR/DATE=MODIFIED

Directory DISK1:[VERSION13]

BUILD_CLASS.COM;4      5-AUG-1987 13:17
SYSTEM3.MMS;3          5-AUG-1987 13:09

Total of 2 files.

❷$ @BUILD_CLASS

❸ Enter name of class to build: VERSION_1_3

  mms$cmslib :== 'f$logical("CMS$LIB")
  IF mms$cmslib .nes. "DISK1:[SYSTEM3_LIB]" THEN
    CMS SET LIBRARY DISK1:[SYSTEM3_LIB]
  %CMS-I-LIBIS, CMS library is DISK1:[SYSTEM3_LIB]
❹ CMS FETCH MAIN.PAS /GEN=VERSION_1_3 ""
  %CMS-S-FETCHED, generation 2 of element MAIN.PAS fetched
  IF mms$cmslib .EQS. "" THEN
    CMS SET LIBRARY 1234
  %CMS-E-NOREF, error referencing 1234
  -CMS-E-MUSTBEDIR, 1234 must be a directory specification
  %CMS-W-UNDEFLIB, CMS library is now undefined
  IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM3_LIB]"
    THEN CMS SET LIBRARY 'mms$cmslib'
  PASCAL /NOLIST/OBJECT=MAIN MAIN.PAS
  mms$cmslib :== 'f$logical("CMS$LIB")
  IF mms$cmslib .nes. "DISK1:[SYSTEM3_LIB]" THEN
    CMS SET LIBRARY DISK1:[SYSTEM3_LIB]
  %CMS-I-LIBIS, CMS library is DISK1:[SYSTEM3_LIB]
❹ CMS FETCH SUB1.PAS /GEN=VERSION_1_3 ""
  %CMS-S-FETCHED, generation 1 of element SUB1.PAS fetched
  IF mms$cmslib .EQS. "" THEN
    CMS SET LIBRARY 1234
  %CMS-E-NOREF, error referencing 1234
  -CMS-E-MUSTBEDIR, 1234 must be a directory specification
  %CMS-W-UNDEFLIB, CMS library is now undefined
  IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM3_LIB]"
    THEN CMS SET LIBRARY 'mms$cmslib'
  PASCAL /NOLIST/OBJECT=SUB1 SUB1.PAS
  mms$cmslib :== 'f$logical("CMS$LIB")
  IF mms$cmslib .nes. "DISK1:[SYSTEM3_LIB]" THEN
    CMS SET LIBRARY DISK1:[SYSTEM3_LIB]
  %CMS-I-LIBIS, CMS library is DISK1:[SYSTEM3_LIB]
```

(continued on next page)

**Example 4–5 (Cont.): Building a System from a Previous CMS Class**

```
❹ CMS FETCH SUB2.PAS /GEN=VERSION_1_3 ""
  %CMS-S-FETCHED, generation 3 of element SUB2.PAS fetched
  IF mms$cmslib .EQS. "" THEN
    CMS SET LIBRARY 1234
  %CMS-E-NOREF, error referencing 1234
  -CMS-E-MUSTBEDIR, 1234 must be a directory specification
  %CMS-W-UNDEFLIB, CMS library is now undefined
  IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM3_LIB]"
    THEN CMS SET LIBRARY 'mms$cmslib'
❺ PASCAL /NOLIST/OBJECT=SUB2 SUB2.PAS
  LINK /TRACE/NOMAP/EXEC=MAIN MAIN.OBJ, SUB1.OBJ, SUB2.OBJ

❻ $  DIR/DATE=MODIFIED

Directory DISK1:[VERSION13]

BUILD_CLASS.COM;4      5-AUG-1987 13:17
MAIN.EXE;1             5-AUG-1987 13:22
MAIN.OBJ;1             5-AUG-1987 13:21
MAIN.PAS;1            30-JUL-1987 13:22
SUB1.OBJ;1             5-AUG-1987 13:22
SUB1.PAS;1            30-JUL-1987 13:10
SUB2.OBJ;1             5-AUG-1987 13:22
SUB2.PAS;1             5-AUG-1987 13:12
SYSTEM3.MMS;3          5-AUG-1987 13:09

Total of 9 files.
```

❶ Your directory contains the command procedure and the description file.

❷ You invoke the command procedure.

❸ You enter the class name.

❹ MMS fetches all program code from the chosen class using the macro CMSFLAGS to override the default.

❺ The source code is compiled and linked normally.

❻ Your directory contains the complete system.

After you have built the previous version of your system, you can fix the bug you may have found. The following steps ensure a complete and accurate system:

- Reserve the CMS element generation with the bug.
- Edit the element to fix the bug.
- Replace the CMS element as an alternate line of descent.

- Replace the generation that was in the CMS class with the fixed version (CMS INSERT GENERATION /SUPERSEDE).

### Using Logical Names for CMS Library Specifications

When writing CMS library specifiers in a description file, you can use logical names instead of a hard-coded device and directory name. This allows the exact location of the library to change, and can make the description file easier to read. The command procedure that invokes MMS can set the logical name. (Or perhaps it is set in a group or system logical name table.)

## 4.2.8 Using the .INCLUDE Directive to Include CMS Files

You can also use the tilde format with the .INCLUDE directive to include files that are stored in the current default CMS library. For example:

```
.INCLUDE RULES~
```

This line in the description file directs MMS to fetch the file RULES.MMS from the current CMS library. (The .INCLUDE directive is discussed in Section 2.8.9.)

When a tilde occurs in your description file, MMS looks for the file in the current CMS library, even if you specify /NOCMS on the command line. However, if the CMS element is newer than the target in the dependency, the element is not fetched from its CMS library unless an action line directs CMS to fetch the source.

## 4.2.9 Using a User-Defined Rule to Access a Single CMS Element

The following example shows a user-defined rule for accessing a single-file CMS element.

```
.C~.OBJ :
    CMS FETCH $(MMS$CMS_ELEMENT) $(CMSFLAGS) $(CMSCOMMENT)
    $(CC) $(CFLAGS) $(MMS$CMS_ELEMENT)
```

This dependency rule tells MMS to do the following:

1. Fetch the .C source file from the current default CMS library, applying the qualifiers specified by the CMSFLAGS macro and writing to the CMS history file the remark specified by the CMSCOMMENT macro.
2. Run the C compiler on the file fetched from the CMS library, applying the qualifiers specified by the CFLAGS macro.

CMSFLAGS and CMSCOMMENT are default MMS macros.You can redefine them so that the same qualifiers or the same remarks are used for all accesses to CMS elements.

## 4.2.10 Accessing a CMS Element Not in the Default CMS Library

The next example shows how to access a CMS element that is not in the current default CMS library.

```
TEST.C : [OTHER.CMS]TEST.C~
     CMS SET LIBRARY [OTHER.CMS]
     CMS FETCH TEST.C "Auto fetch from MMS"
```

This dependency rule causes MMS to set the current default CMS library to [OTHER.CMS], fetch the element TEST.C, and write the specified remark to the CMS history file. (MMS does not reset the CMS library back to the default in this example. This action differs from that of the built-in rules for CMS element access. See Table C–9 in Appendix C.)

## 4.2.11 Accessing Description Files in CMS Libraries

If a description file does not exist in your default directory, and if you have defined a CMS library, you can request that MMS retrieve the description file from the CMS library by using the /CMS qualifier on the MMS command line. If the description file exists in your directory and is newer than the element in the CMS library, MMS uses the file in your directory.

If you know that the description file you want to use is stored in a CMS library, you can explicitly request MMS to use that file. When you use the /DESCRIPTION qualifier on the MMS command line, you can follow the name of the description file with a tilde character so that MMS automatically fetches the file from the current CMS library. For example:

```
$ MMS/DESCRIPTION=ALL~
```

This command directs MMS to fetch the description file ALL.MMS from the current CMS library.

If the file you specify with /DESCRIPTION does not exist in the current CMS library, MMS issues an error message.

## 4.3 Checking for Replacement of CMS Elements

If more than one programmer is working on a project, you may want to wait for someone else to replace an element in the project CMS library before you do a particular task. MMS can automatically check for element replacements at specified intervals by using the command procedure in the following example. Besides the command procedure, you also need a description file that tells MMS which element to look for and how to notify you when the element has been replaced. Such a description file might be named THERE.MMS and look like this:

```
THERE.TIM : NEEDED.FOR~   ! The name of the element
     IF "''F$SEARCH("THERE.TIM")'" .NES "" -
          THEN MAIL NL: 'F$GETJPI(" ","USERNAME")'-
     /SUBJECT="$(MMS$SOURCE) is back in the CMS library."
     SET DEFAULT 1234567890  ! Causes MMS to abort with $STATUS = failure
```

The command procedure CHECKCMS.COM that loops until the specified element is available in the CMS library is as follows:

```
$ CMS SET LIBRARY [LOUISE]       ! The CMS library
$ SET DEFAULT [LOUISE.WORK]      ! Your working directory
$ IF "''F$SEARCH("THERE.TIM")'" .EQS. "" THEN COPY NL: THERE.TIM
$ LOOP:
$ MMS/DESCRIPTION=THERE
$ IF .NOT. $STATUS THEN EXIT
$ WAIT 0:5   ! or some interval
$ GOTO LOOP
```

When submitted to the batch queue, this command procedure runs MMS, which checks to see whether the element in the CMS library is newer than THERE.TIM. If it is not (that is, if the element has not been replaced in the CMS library), $STATUS is 1, and MMS waits the specified interval before trying again. If the element has been replaced, the first bit in $STATUS is 0, and MMS mails you the message "NEEDED.FOR is back in the CMS library."

You can run this procedure in a subprocess (instead of submitting it to the batch queue) by typing the following command:

```
$ SPAWN/NOWAIT @CHECKCMS
```

## 4.4 Accessing Forms in an FMS Library

If VAX FMS is installed on your system, you can use MMS to access forms stored in FMS libraries. You should be familiar with FMS before reading this section.

To specify an FMS form in a dependency rule, use the same syntax as for files in VMS libraries. This syntax is explained in detail in Section 4.1. The file type .FLB after the library name informs MMS that the library contains FMS forms. The default file type for FMS forms is .FRM.

For example, consider the following dependency rule:

```
A.FLB(B) : B.FRM
    $(FMS) $(FMSFLAGS) A.FLB B.FRM
```

B.FRM is the source that updates the target B in the FMS library A.FLB. FMS and FMSFLAGS are default macros that invoke FMS with the /REPLACE qualifier.

MMS uses the insertion time of a form in an FMS library to determine whether a source is newer than the target. You cannot use the /REVISE_DATE qualifier with references to FMS forms. (See the Command Dictionary for a description of /REVISE_DATE.)

## 4.5 Accessing Records in the CDD

If the VAX Common Data Dictionary (CDD) is installed on your system, you can use MMS to access records stored in the CDD. You should be familiar with the Common Data Dictionary before reading this section.

In a dependency rule, you follow the path name of a CDD record description with the caret or an up-arrow character (^) to inform MMS that the source is stored in the CDD. For example:

```
A.OBJ : A.PAS, CDD$TOP.B.C.D.E^       ! CDD record referred to in A.PAS
    PASCAL A.PAS
```

MMS uses the CDD path specification to find the source and check its revision time against that of the target, A.OBJ. In this example, A.OBJ resides in your current directory.

The CDD maintains a history list that includes the date and time that a CDD record was accessed and an optional remark that you supply to document the access. To insert a remark in the CDD history list when MMS accesses a CDD record, you can use the /AUDIT qualifier after the caret in the CDD record specification. The /AUDIT qualifier is followed by a quoted string that contains the remark that is to be inserted in the CDD history file. For example:

```
A.OBJ : A.PAS, CDD$TOP.B.C.D.E^/AUDIT="Accessed by MMS to update A"
    PASCAL A
```

MMS writes the remark that follows the /AUDIT qualifier into the CDD history list for the specified record.

When used with the CDD record specification, the /AUDIT qualifier must follow the caret character; separate the qualifier from the remark with an equal sign or colon. You cannot use /AUDIT on the MMS command line.

MMS also provides the default macro CDDFLAGS. This macro is defined to be the null string, but it can be redefined so that the same remark is written to the history file for all accesses to CDD records. For example, you could set up your description file as follows:

```
CDDFLAGS = /AUDIT="Record accessed by MMS"

A.OBJ : A.PAS, CDD$TOP.B.C.D.E^
     PASCAL A

Q.OBJ : Q.PAS, CDD$TOP.L.M.N.O^
     PASCAL Q

V.OBJ : V.PAS, CDD$TOP.W.X.Y.Z^
     PASCAL V
```

When MMS accesses one of these sources from the CDD, it writes the string that is the value of CDDFLAGS into the history file.

The following restrictions apply to CDD access:

- You cannot access CDD records that reside on a different DECnet node than your own.
- You cannot use the /REVISE_DATE qualifier with references to CDD records. (See the Command Dictionary for a description of /REVISE_DATE.)
- /AUDIT information is not examined during CDD node comparisons.
- The /NOACTION qualifier has no effect on the /AUDIT qualifier. That is, if you have suppressed the execution of action lines with the /NOACTION qualifier, the remark you supplied with /AUDIT is still written to the CDD history file.

## 4.6 Accessing Files in an SCA Library

When you use MMS with the /SCA_LIBRARY qualifier, MMS allows you to generate an SCA library during the build process. Example 4–6 demonstrates how to use MMS with the /SCA_LIBRARY qualifier.

## Example 4–6: Using MMS with the /SCA_LIBRARY Qualifier

```
$ SET DEFAULT  [SYSTEM1]
❶ $ SCA CREATE LIB [.SCALIB]
%SCA-S-NEWLIB, SCA Library created in DISK1$:[SYSTEM1.SCALIB]
%SCA-S-LIB, your SCA Library is DISK1$:[SYSTEM1.SCALIB]
$
❷ $ TYPE SCA.MMS
PROG.EXE : PROG.OBJ

PROG.OBJ : PROG.C
$
❸ $  TYPE  PROG.C
main ()
     {
      int total;

      total = 2 + 2;
     }
$
❹ $  MMS/SCA_LIBRARY/DESCRIPTION=SCA PROG.EXE
CC /NOLIST/OBJECT=PROG/ANALYSIS_DATA=PROG PROG.C
mms$scalib = F$TRNLNM( "SCA$LIBRARY")
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "SCA$LIBRARY:" .NES.-
 "SCA$LIBRARY:" THEN mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "SCA$LIBRARY:" .NES.-
 "SCA$LIBRARY:" .AND. mms$scalib .NES. "SCA$LIBRARY:" THEN-
  mms$scasetlib = 3
IF F$SEARCH("SCA$LIBRARY:SCA$EVENT.DAT") .EQS. "" THEN-
 mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN SCA CREATE LIBRARY SCA$LIBRARY:
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN SCA SET LIBRARY SCA$LIBRARY:
❺ SCA LOAD PROG
%SCA-S-LOADED, module PROG loaded
%SCA-S-COUNT, 1 module loaded (1 new, 0 replaced)
IF mms$scasetlib THEN SCA SET LIBRARY 'mms$scalib'
LINK /TRACE/NOMAP/EXEC=PROG PROG.OBJ
$
```

❶ Set default to your system directory, and initialize the SCA library. A side effect of initializing the library is that the logical name SCA$LIBRARY is now defined.

❷ The MMS description file describing the system dependencies.

❸ The source code file that MMS uses to compile and link our target and sources.

❹ MMS/SCA_LIBRARY is invoked with the target PROG.EXE specified.

❺ SCA loads the SCA data file into the SCA library.

# Command Dictionary

This Command Dictionary describes the elements of the MMS command line, defines the syntax rules for entering commands, and describes the MMS command and its qualifiers.

### Command Format

The MMS command has the following format:

```
MMS [/qualifier . . . ] [target, . . . ]
```

## Parameters

### qualifier

An MMS qualifier.

### target

The name of a target, which can be either a VMS file specification, a library specification enclosed in quotes, or a mnemonic name.

Unless you use the /NODESCRIPTION qualifier on the command line, you need not type the qualifiers and targets you want to use. MMS assumes default qualifiers and updates the first target in the description file whenever you type the MMS command.

### Qualifiers

The MMS command qualifiers modify the command and can be placed anywhere on the command line after the MMS command. The MMS command activates the following default qualifiers:

/ACTION
/NOCHECK_STATUS
/NOCMS
/DESCRIPTION=DESCRIP.MMS or /DESCRIPTION=MAKEFILE.
/NOIGNORE
/NOLOG
/OUTPUT=SYS$OUTPUT
/NOOVERRIDE
/NOREVISE_DATE
/RULES
/NOSCA_LIBRARY
/NOSKIP_INTERMEDIATE
/VERIFY

You can abbreviate all MMS qualifiers and their parameters. However, you must be sure that the abbreviations are unique, so they will not be confused with other CLI qualifiers. If you type an ambiguous abbreviation, the CLI issues an error message.

You can continue an MMS command to the next line by using the DCL continuation character, a hyphen (-), as the last character on the command line.

The MMS qualifiers are described in alphabetic order, and that description notes whether the qualifier affects the behavior of MMS, the execution of action lines, or both. The notation (D) following a qualifier indicates the default form.

# MMS

The command MMS invokes the VAX DEC/Module Management System. By default, it searches for the description file *descrip.mms* or *makefile.* in the current directory and looks to the first action line for the target. To use MMS, your directory must contain one of these files.

## Format

**MMS** *[/qualifier . . . ] [target, . . . ]*

| Command Qualifiers | Defaults |
|---|---|
| /ACTION | /NOACTION |
| /CHECK_STATUS | /NOCHECK_STATUS |
| /CMS | /NOCMS |
| /DESCRIPTION | /NODESCRIPTION |
| /FROM_SOURCES | |
| /HELP | |
| /IDENTIFICATION | |
| /IGNORE | /NOIGNORE |
| /LOG | /NOLOG |
| /MACRO | |
| /OUTPUT | |
| /OVERRIDE | /NOOVERRIDE |
| /REVISE_DATE | /NOREVISE_DATE |
| /RULES | /NORULES |
| /SCA_LIBRARY | /NOSCA_LIBRARY |
| /SKIP_INTERMEDIATE | /NOSKIP_INTERMEDIATE |
| /VERIFY | /NOVERIFY |

## Command Parameters

***/qualifier***
An MMS qualifier.

***target***
The name of a target, which can be either a VMS file specification or a logical name.

## Command Qualifiers

### /ACTION (D)
### /NOACTION

The /ACTION and /NOACTION qualifiers control whether MMS executes the action lines in a description file. These qualifiers affect only the execution of action lines, not the behavior of MMS. The /ACTION qualifier directs MMS to execute action lines.

The /NOACTION qualifier directs MMS not to execute action lines, but still to write them to the output file. (The output file can be either SYS$OUTPUT or the file specified by the /OUTPUT qualifier.) /NOACTION is useful for determining what actions MMS would have executed had the system actually been built. You can also use /NOACTION in combination with the /OUTPUT qualifier to generate a command procedure (refer to the description of the /OUTPUT qualifier).

/NOACTION overrides the Silent action line prefix (@) described in Section 2.7.6. Note that the $(MMS) reserved macro is executed even if you specify /NOACTION. Therefore, you can see what actions MMS would have executed in the subprocess. See Section 3.3 for information about the $(MMS) macro.

The /NOACTION qualifier does not affect the /AUDIT qualifier that you can provide with references to CDD records. That is, if you have suppressed the execution of action lines with the /NOACTION qualifier, the remark you supplied with /AUDIT is still written to the CDD history file. The /AUDIT qualifier and the use of CDD records with MMS is described in Section 4.5.

### /CHECK_STATUS
### /NOCHECK_STATUS (D)

The /CHECK_STATUS and /NOCHECK_STATUS qualifiers control whether MMS returns a value in the symbol MMS$STATUS, instead of updating a target. This symbol contains the status of the last action line executed by MMS. These qualifiers affect both the execution of action lines and the behavior of MMS.

/CHECK_STATUS directs MMS to check whether a target is up–to–date by determining whether any actions would be executed if MMS/ACTION were specified. MMS issues an informational message and sets MMS$STATUS to 1 if no actions would be executed (that is, if the target is up–to–date). If the target needs to be updated, MMS sets the MMS$STATUS value to 0.

The /CHECK_STATUS qualifier has precedence over both the /ACTION and /REVISE_DATE qualifiers if they appear on the same command line. In these cases, only /CHECK_STATUS is processed.

The /NOCHECK_STATUS qualifier directs MMS to process the description file as it normally would, executing action lines if necessary.

### /CMS
### /NOCMS (D)

If CMS is installed on your system, you can use the /CMS and /NOCMS qualifiers to control whether MMS looks for source files, description files, and included files in the current default CMS library, as well as in the specified directories. See Section 4.2 for information on using MMS to access elements in CMS libraries. These qualifiers affect both the execution of action lines and the behavior of MMS.

The /CMS qualifier directs MMS to look for source files in the current default CMS library and in the specified directories. If the source in the CMS library is newer, it is fetched from there. If the source in the CMS library is older, MMS uses the source in the specified directory rather than fetch it from the CMS library. /CMS also directs MMS to look in the current default CMS library for a description file and any files included with the .INCLUDE directive. If MMS cannot find a description file in either the specified directory or the current default CMS library, it aborts execution. (The .INCLUDE directive is described in Section 2.8.9.)

The /CMS qualifier also directs MMS to apply CMS built–in rules where appropriate. (See Table C–9 for a table of CMS built–in rules.)

The /NOCMS qualifier directs MMS not to look in the current default CMS library for source files, description files, or included files. However, if any file specifications in the description file are followed by a tilde ( ~ ) to indicate specific CMS elements, MMS looks for the files in the CMS library even if /NOCMS is in effect.

If you specify /NOCMS or the combination /CMS/NORULES, and the sources do not exist in the specified directory, MMS aborts execution.

### /DESCRIPTION (D)
### /NODESCRIPTION

The /DESCRIPTION and /NODESCRIPTION qualifiers control whether MMS looks for a description file to update the target. These qualifiers affect the behavior of MMS but not the execution of action lines.

```
$ MMS/DESCRIPTION=filespec . . .
$ MMS/NODESCRIPTION target
```

The *filespec* is a VMS file specification or a logical name that identifies the description file. The default file type is .MMS. If a tilde ( ~ ) follows the file specification, MMS fetches the description file from the default CMS library even if the description file exists in the default directory. The *target* is a VMS file specification or a mnemonic name that designates the target to be built.

When you specify more than one description file, separate the file specifications with either commas (,) or plus signs (+) and enclose them in parentheses or quotation marks

If you use commas, the description files are processed separately and the list of files must be enclosed in parentheses:

```
$ MMS/DESCRIPTION=(A, B)
```

If you use plus signs, the description files are concatenated and processed as one file. The list of files must be enclosed in quotation marks.

```
$ MMS/DESCRIPTION="A + B"
```

The following command line directs MMS to process A.MMS and B.MMS as one file, and CLEANUP.MMS as another. Since you are specifying essentially two description files here, you must use commas to separate the file specifications.

```
$ MMS/DESCRIPTION=("A + B", CLEANUP)
```

In this case, there are two default targets: the first one in either A.MMS or B.MMS (depending on the contents of the two files) and the second one in CLEANUP.MMS.

If you specify a list of description files in parentheses and a list of targets, the rules for updating all the listed targets must occur in all the listed description files. Consider the following command:

```
$ MMS/DESC=(A,B) X,Y,Z
```

In this case, the rules for updating X, Y, and Z must appear in both description files, A.MMS and B.MMS.

If you specify a concatenated list of description files and a list of targets, the rules for updating all the listed targets must occur in the concatenated description file. Consider the following command:

```
$ MMS/DESC="A + B" X,Y,Z
```

In this case, the description file formed by the concatenation of A.MMS and B.MMS must contain the rules for updating X, Y, and Z.

If you specify /DESCRIPTION without the name of a description file, MMS looks for the default description file DESCRIP.MMS. If it cannot locate that file, it looks for MAKEFILE.; if it cannot find MAKEFILE., it issues an error message and aborts execution.

If you do not specify /DESCRIPTION, MMS looks first for DESCRIP.MMS. If it cannot locate that file, it looks for one called MAKEFILE.. If that search also fails, MMS issues an error message and aborts execution.

The /NODESCRIPTION qualifier directs MMS to ignore all description files and to build the target specified on the command line. When you use the /NODESCRIPTION qualifier, MMS does not automatically look for DESCRIP.MMS and MAKEFILE. You must specify a target on the command line; otherwise, MMS does not know what target to build.

### /FROM_SOURCES

The /FROM_SOURCES qualifier directs MMS to build a target from its sources, regardless of whether the target is already up to date. This qualifier affects the execution of action lines and the behavior of MMS.

When you specify /FROM_SOURCES on the command line, MMS does not compare the revision times of the specified sources and target. Instead, it executes the action lines in the description file necessary to update the target. The /FROM_SOURCES qualifier is useful when you want to guarantee that an entire system is rebuilt, perhaps for an internal release.

If you specify /CMS/FROM_SOURCES qualifiers on the MMS command line, MMS uses the sources found in the default CMS library. If you do not use /CMS, MMS locates the sources in the specified directory.

The /FROM_SOURCES qualifier overrides the /SKIP_INTERMEDIATE qualifier.

### /HELP

The /HELP qualifier allows you to obtain information about MMS and its qualifiers. This qualifier affects the behavior of MMS but not the execution of action lines.

```
$ MMS/HELP["topic"]
```

The *topic* is a MMS topic on which you want information.

The /HELP qualifier displays on your terminal the information in the HELP library specific to MMS. If you use the /HELP qualifier alone, you are presented with general information about MMS and a list of its qualifiers and other topics on which more detailed information is available. To see the information about one of these topics, follow the /HELP qualifier with an equal sign (=) and the topic. The topic must be enclosed in quotation marks.

### /IDENTIFICATION

The /IDENTIFICATION qualifier directs MMS to print an informational message with the version number of the MMS image and the copyright date. This qualifier affects the behavior of MMS but not the execution of action lines.

The /IDENTIFICATION qualifier provides you with the version number and copyright date of the MMS image you are running. When you use /IDENTIFICATION, MMS does not process any description files or qualifiers; it simply prints an informational message on your screen. You should include the version number and copyright date when you submit Software Performance Reports (SPRs) about MMS.

### /IGNORE
### /NOIGNORE (D)

The /IGNORE qualifier specifies the severity levels of errors that MMS should ignore when it executes action lines. The parameters correspond to the DCL severity levels W, E, and F. /NOIGNORE directs MMS to abort execution when it finds any error. These qualifiers affect the execution of action lines but not the behavior of MMS. The format of this qualifier is as follows:

```
$ MMS/IGNORE=[WARNING]|ERROR |FATAL

$ MMS/NOIGNORE
```

The *WARNING* directs MMS to ignore W errors and continue processing, but to abort execution when it finds either an E or an F error. If you specify /IGNORE without parameters, WARNING is the default. The *ERROR* string directs MMS to ignore both W and E errors, but to abort execution when it finds an F error. The *FATAL* directs MMS to ignore all errors, and to continue processing the description file. This parameter is equivalent to the .IGNORE directive.

The errors that MMS ignores when you specify /IGNORE are those errors generated by the execution of action lines, rather than MMS errors. /IGNORE does not stop MMS error messages from being generated or displayed. Informational messages are always displayed, regardless of any use of the /IGNORE qualifier.

You should be careful about executing MMS with the /IGNORE qualifier. If errors occur during processing, the target may be updated but still contain errors of which you will not be aware.

The .IGNORE directive and the Ignore action line prefix are similar to the /IGNORE=FATAL qualifier. Instead of typing them on the command line, however, you include them in the description file. Sections 2.8.1 and 2.7.5 describe the Ignore action line prefix and the .IGNORE directive in detail.

If you want to override the .IGNORE directive contained in a description file, you must type the /IGNORE[=WARNING], /IGNORE=ERROR, or /IGNORE=FATAL qualifier explicitly on the MMS command line. You cannot override the Ignore action line prefix from the MMS command line.

### /LOG
### /NOLOG (D)

The /LOG and /NOLOG qualifiers control whether MMS displays on your terminal informational messages about its findings and assumptions as it processes the description file. These qualifiers affect the behavior of MMS but not the execution of action lines.

The /LOG qualifier directs MMS to write all informational messages to your terminal screen while it processes the description file. The /LOG qualifier is useful for debugging your description files. These messages indicate what MMS finds and what it assumes as it processes the description file. You should include these messages with any Software Performance Reports (SPRs) that you submit about MMS. To save these messages in a file, type the following:

```
$ DEFINE SYS$OUTPUT MYFILE.LOG
$ MMS/LOG
$ DEASSIGN SYS$OUTPUT
```

The /NOLOG qualifier directs MMS not to display informational messages about its assumptions while it processes the description file.

However, if you specify /NOLOG/CHECK_STATUS on the same command line, MMS does display the informational message that reports the value of MMS$STATUS. (Refer to the description of /CHECK_STATUS for more information about MMS$STATUS.)

### /MACRO

The /MACRO qualifier directs MMS to add to or override the macro definitions in the description file. This qualifier affects the behavior of MMS but not the execution of action lines. The syntax of this qualifier is as follows:

```
MMS/MACRO=filespec |"macro", . . .
```

The *filespec* is a VMS file specification or a logical name that identifies a file of macro definitions. The default file type is .MMS. The *macro* string is a macro definition enclosed in quotation marks. Use the same format as for macro definitions in description files, that is, **name = string**.

The /MACRO qualifier allows you to specify a macro definition on the MMS command line. It also allows you to specify a file of macro definitions that you want to use in your description file. Section 2.4.2 gives a detailed discussion of the use of macros.

You can define macros in three locations:

- In a description file
- In a macro definitions file
- On the command line

To specify more than one macro definition on the MMS command line, enclose the list of macros in parentheses. For example:

```
$ MMS/MACRO=("A=MAC1", "B=MAC2")
```

You can also specify both a macro definition and a file on the same command line:

```
$ MMS/MACRO=("A=MAC1", MACROS)
```

### /OUTPUT

The /OUTPUT qualifier directs MMS to write action lines and output to the specified file. Error messages preceded by "%MMS" are not written to this output file, but instead are written to SYS$ERROR. This qualifier affects the behavior of MMS but not the execution of action lines.

```
$ MMS/OUTPUT=filespec
```

The *filespec* is a VMS file specification or a logical name that identifies the output file. The default file type is .LOG.

If you specify the /NOVERIFY qualifier on the same MMS command line with /OUTPUT, MMS does not write action lines to the output file.

If you specify /OUTPUT and your command-line interpreter is DCL, MMS automatically prefixes a dollar sign ($) to any action line that does not begin with one. This technique allows you to use the file generated by /OUTPUT as a DCL command procedure.

If you do not specify the /OUTPUT qualifier on the MMS command line, MMS writes all action lines, messages, and output to SYS$OUTPUT.

### /OVERRIDE
### /NOOVERRIDE (D)

The /OVERRIDE and /NOOVERRIDE qualifiers control the order in which MMS applies definitions when it processes macros. These qualifiers affect the behavior of MMS but not the execution of action lines.

The /OVERRIDE qualifier directs MMS to override the macro definitions in the description file with CLI symbol definitions. To find the macro definitions that should have precedence, MMS looks at symbols defined by the CLI assignment statement, scanning the CLI symbol table for the body of the macro. If the body of the macro is not in the CLI symbol table, MMS substitutes a null string for all invocations of the macro.

The /OVERRIDE qualifier imposes the following order of application when MMS processes macro definitions:

1. Command-line
2. CLI symbol
3. Description file
4. Built-in

Once MMS finds a definition for a macro, it does not search those locations farther down the list for more definitions. If MMS finds more than one definition in the same location (such as on a command line), it uses the last definition it processed, unless the location is a description file. MMS issues an error message if a macro is defined more than once in a description file.

The /NOOVERRIDE qualifier imposes the following order, which is the default hierarchy:

1. Command–line
2. Description file
3. Built–in
4. CLI symbol

### /REVISE_DATE
### /NOREVISE_DATE (D)

The /REVISE_DATE and /NOREVISE_DATE qualifiers control whether MMS changes only the revision dates of all targets that need updating, rather than actually performing the update. These qualifiers affect the behavior of MMS, not the execution of action lines.

The /REVISE_DATE qualifier directs MMS to change only the revision dates of any target that needs updating; it does not direct MMS to execute the action lines that actually do the updating. If any files are missing, /REVISE_DATE causes MMS to create them. If MMS cannot create a missing file, or if it cannot update the revision date of an existing file, it issues an error message.

The /REVISE_DATE qualifier is useful for reducing the number of superfluous compilations—for example, when only a comment line was changed in a required file. However, /REVISE_DATE can defeat the purpose of using MMS, so you should use this qualifier with caution.

As it changes the revision times, MMS writes the name of the revised files to the output file (either SYS$OUTPUT or the file specified by the /OUTPUT qualifier). If you specify /REVISE_DATE/NOVERIFY, the names of revised files are suppressed. (Section 2.8.2 describes the .SILENT directive.)

Unless you specify a target on the command line, the /REVISE_DATE qualifier causes the first target in the description file and its sources to be revised by MMS. If you specify multiple targets on the command line, those targets and their sources are revised. /REVISE_DATE does not change the value of MMS$STATUS (see Section 2.7 for information about MMS$STATUS).

The /REVISE_DATE qualifier has precedence over the /ACTION qualifier if they both appear on the same command line. In that case, only /REVISE_DATE is processed.

The /NOREVISE_DATE qualifier directs MMS to build the system by updating targets as necessary (as long as the /CHECK_STATUS qualifier does not appear on the same command line).

### /RULES (D)
### /NORULES

The /RULES and /NORULES qualifiers control whether MMS applies built-in rules and the suffixes precedence list when it builds a system. These qualifiers affect the behavior of MMS but not the execution of action lines.

```
$ MMS/RULES[=filespec]
```

```
$ MMS/NORULES
```

*Filespec* is a VMS file specification or a logical name that identifies the file of default rules that MMS is to use.

The /RULES qualifier directs MMS to use built-in rules and the suffixes precedence list. If you supply a file specification with /RULES, MMS reads the specified file and uses the rules and suffixes list it contains as the built-in rules. The rules in this file replace the built-in rules that MMS normally uses. If you specify /RULES without a file specification, MMS translates the logical name MMS$RULES to find the file of built-in rules to use. If MMS$RULES is not defined, MMS uses its own built-in rules. Therefore, if you want to replace MMS's built-in rules with default rules of your own, you have two choices:

- You can create a file of your rules and specify the file with the /RULES qualifier on the MMS command line. The file specified with /RULES has precedence over the file represented by MMS$RULES.
- You can assign the logical name MMS$RULES to the file specification of your rules file.

The /NORULES qualifier directs MMS not to use its built-in rules or the suffixes precedence list. It also prevents MMS from applying user-defined rules and default macros.

This qualifier is useful when the description file makes explicit all actions MMS should take in building a system. When you specify /NORULES, MMS applies only the dependency rules contained in the description file.

### /SCA_LIBRARY[=library–name]
### /NOSCA_LIBRARY (D)

The /SCA_LIBRARY qualifier controls whether MMS generates an SCA library during the build process.

```
$ MMS/SCA_LIBRARY[=library--name]

$ MMS/NOSCA_LIBRARY
```

When you specify a library name with the /SCA_LIBRARY qualifier, MMS defines the macro $(SCALIBRARY) to be that library name. If you use /SCA_LIBRARY without specifying a library name, SCA$LIBRARY is the value of $(SCALIBRARY). If the /SCA_LIBRARY qualifier is not specified, /NOSCA_LIBRARY is the default.

The macro $(SCA) is defined to be SCA regardless of the setting of the /SCA_LIBRARY qualifier.

The macro $(MMSQUALIFIERS) contains the setting of the /SCA_LIBRARY qualifier.

If the /SCA_LIBRARY qualifier is specified, macros and built-in rules for BASIC, BLISS-32, C, FORTRAN, PASCAL, and PLI change. Table C–2 lists the macros that change when MMS is used with the /SCA_LIBRARY qualifier.

**NOTE**

You may find it preferable to defer the loading of modules into the SCA library until after all the compilations have been completed. In this case you should define the default rules for compilation in your description file to be the same as the default rule provided by MMS when /NOSCA_LIBRARY is specified. You should also include a .LAST directive, which then loads the SCA database. For example,

```
.LAST :
   $(SCA) SET LIBRARY $(SCALIBRARY)
   $(SCA) LOAD *
```

### /SKIP_INTERMEDIATE
### /NOSKIP_INTERMEDIATE (D)

The /SKIP_INTERMEDIATE and /NOSKIP_INTERMEDIATE qualifiers control whether MMS builds intermediate source/target files. These qualifiers affect the behavior of MMS, not the execution of action lines.

The /SKIP_INTERMEDIATE qualifier directs MMS to determine whether the target is up to date without rebuilding intermediate files unless they need to be updated. If MMS cannot find some intermediate files, it skips over them as though they already existed. Suppose, for example, that you have a .C file and an .EXE file, but no .OBJ file, and the time of the .EXE file is more recent than that of the .C file. /SKIP_INTERMEDIATE directs MMS not to build the .OBJ file and not to rebuild the .EXE file because the target is already up-to-date with regard to its nearest source. Using /SKIP_INTERMEDIATE saves time and disk space.

The /NOSKIP_INTERMEDIATE qualifier directs MMS to make sure that all intermediate source files exist and are up-to-date. If any intermediate source files do not exist, MMS builds them.

When you use the /SKIP_INTERMEDIATE qualifier, be aware that certain MMS actions (such as invoking the VAX Linker) require all sources to be present. Other actions (such as invoking the VMS librarian) may operate correctly only on the sources used to update the current target.

MMS dependencies cannot distinguish between a situation in which all sources must be present for MMS to perform the specified action and a situation in which only one of the specified sources may be required.

The following example shows a situation in which all of the sources must be present for MMS to perform the action:

```
PROG.OBJ : PROG.C, DEFS.H
     CC PROG
```

PROG.C and DEFS.H do not depend on each other, but both must be present for MMS to build PROG.OBJ. If one source has changed, and you specify /SKIP_INTERMEDIATE, MMS does not verify that the other source is present in the directory; therefore, it cannot build the target. You will not encounter a situation such as this one if you use /NOSKIP_INTERMEDIATE (the default).

### /VERIFY (D)
### /NOVERIFY

The /VERIFY and /NOVERIFY qualifiers control whether MMS displays action lines before executing them. These qualifiers affect the behavior of MMS, not the execution of action lines.

The /VERIFY qualifier directs MMS to display each action line before executing it. MMS writes action lines either to SYS$OUTPUT or into the file specified by the /OUTPUT qualifier.

If you specify the /REVISE_DATE qualifier on the same command line, the /VERIFY qualifier causes MMS to display the names of files whose dates have been revised.

The /NOVERIFY qualifier suppresses the display (but not the execution) of action lines. Any error messages generated by the execution of action lines continue to be displayed. If you specify /REVISE_DATE/NOVERIFY on the same command line, the names of files whose dates have been revised are not displayed.

The behavior of the /NOVERIFY qualifier is identical to that of the .SILENT directive and the Silent action line prefix (see Sections 2.8.2 and 2.7.6, respectively). If a description file contains the .SILENT directive, but you want to override it, you must type the /VERIFY qualifier explicitly on the MMS command line. You cannot override the Silent action line prefix from the MMS command line.

# Appendix A

# MMS Messages

This appendix lists the MMS messages. These messages are listed alphabetically by identifier, and when necessary are explained. Where possible, suggestions for actions needed to recover from errors are included.

MMS messages are displayed on the current output device. If you are running MMS interactively, this device is a terminal. If you are running MMS in batch mode, messages are written into the log file.

## A.1 Message Format

The general format of messages displayed by the VMS operating system is the following:

```
%FACILITY-L-IDENT, text
```

MMS messages range in purpose from confirming the successful completion of your last MMS command to notifying you of an error that caused the last command to be terminated.

The severity level of a message indicates the general nature of the message. MMS messages have one of three severity levels: I, W, or F. These severity levels indicate the following:

- Informational (I) messages indicate MMS's actions during the process of building the system. The display of many of these messages can be controlled by the /LOG qualifier on the command line. Other informational messages are displayed regardless of whether you specified /LOG.
- Warning (W) messages indicate that MMS has encountered a minor error. If the error occurred during the execution of an action line, processing stops unless you specified the /IGNORE=FATAL, /IGNORE=ERROR, or /IGNORE=[WARNING] qualifiers on the command line.

- Fatal (F) messages indicate that MMS is about to terminate because of a problem that prevents it from continuing any further. Processing of the command stops.

MMS does not generate Success (S) or Error (E) messages.

For some error messages, the recommended action is to submit a Software Performance Report (SPR). See the *VAX DEC/Module Management System Installation Guide* for information on submitting SPRs.

## A.2 MMS Messages

This section lists all MMS messages along with brief descriptions and recommended user actions. A term enclosed in single quotation marks is variable information.

ABORT, For target 'target name,' CLI returned abort status: %X'status.'

**Explanation:** Execution of an action line in the description file returned a fatal or warning error. By default, MMS aborts execution.

**User Action:** Correct the error in the action line.

BADTARG, Specified target 'target name' does not exist in the description file.

**Explanation:** You specified a target on the command line that does not exist in your description file.

**User Action:** Correct the command line or the target specification in the description file.

CDDACCERR, CDD access error on path 'path specification.'

**Explanation:** The VAX Common Data Dictionary (CDD) signaled an error while attempting to access the path specified in your description file.

**User Action:** Verify the path specification and correct the description file.

CDDNOAUD, CDD audit string not found.

**Explanation:** You used the /AUDIT qualifier with a CDD record specification, but you did not supply a remark to be included in the CDD audit history file.

**User Action:** Edit the description file to remove the /AUDIT qualifier or to include a remark with it.

CDDNOTIME, CDD path 'name' has no time attribute.

**Explanation:** The CDD path specification in your description file is not associated with a revision time. Therefore, MMS cannot determine whether the CDD record is newer than your target.

**User Action:** You cannot use a CDD record that is not associated with a revision time. Correct the description file to specify a different CDD record.

CDDPRIERR, Prior severe CDD error has occurred.

**Explanation:** An error occurred earlier in the processing of your description file when MMS tried to access a CDD record. This message is preceded by one of MMS's other error messages that pertain to the CDD and by a message from the CDD itself to help you locate the error.

**User Action:** Correct the condition that caused the first error and try again.

CLPHELP, Please type HELP MMS for help on DEC/MMS.

**Explanation:** For some reason MMS cannot access the help library from the /HELP qualifier on the MMS command.

**User Action:** Type the DCL command HELP MMS instead.

CMSABORT, Aborted with CMS errors.

**Explanation:** One or more errors were returned by Callable CMS and MMS cannot continue processing.

**User Action:** The CMS message printed after %MMS-W-CMSCALL will describe what caused the problem. Refer to the *Guide to DEC/Code Management System* for more information.

CMSBADGEN, Illegal generation 'value' specified in description file.

**Explanation:** The generation value specified by the /GENERATION qualifier is not valid.

**User Action:** Correct the generation value or the CMS library.

CMSBADLIB, There is a problem with the specified CMS library 'library name.'

**Explanation:** MMS is unable to access the specified CMS library.

**User Action:** Correct the CMS library or the description file.

CMSBADTIM, Invalid time field in CMS history file for file 'filespec.'

**Explanation:** The time field in the history portion of the file in the element is in a nonstandard format.

**User Action:** Reserve, then replace the CMS element that contains the specified file. If this element belongs to a specified CMS class, perform the steps necessary to replace the newly created generation of that element into that CMS class.

CMSCALL, Callable CMS has returned an error.

**Explanation:** Callable CMS, used in conjunction with CMS Version 2 libraries, has returned an error to MMS. The specific error is printed on the next line.

**User Action:** Refer to the *Guide to DEC/Code Management System* for more information on the CMS error returned.

CMSNOCLAS, Specified class name 'name' not found in CMS library 'library name.'

**Explanation:** MMS could not find the given class name (specified with /GENERATION in the CMS library.)

**User Action:** Correct the CMS library or the description file.

CMSNOELE, Element 'element name' not found in CMS library.

**Explanation:** MMS could not find the specified element in the CMS library.

**User Action:** Correct the CMS library or the description file.

CMSNOFIL, File filespec not found in CMS library.

**Explanation:** MMS could not find the specified file in the CMS library.

**User Action:** Correct the CMS library or the description file.

CMSNOGEN, No generation value specified.

**Explanation:** You did not specify a value with the /GENERATION qualifier.

**User Action:** Add the value to the /GENERATION qualifier.

CMSNOLIB, Your default CMS library is undefined.

**Explanation:** You do not have a CMS library defined but you used /CMS on the command line or a tilde ( ~ ) in the description file.

**User Action:** Define a CMS library.

CMSNOV2SUP, DEC/CMS is installed without DEC/CMS Version 2.0 support.

**Explanation:** You are trying to access a source in a CMS Version 2.0 library, but MMS was installed without CMS Version 2.0 support.

**User Action:** CMS Version 2 must already be installed on your system before you install MMS if you want access to CMS Version 2.0 libraries.

CMSPROBLEM, Problem with CMS control file 'filespec.'

**Explanation:** The specified control file is either missing or has been opened by another user without using CMS.

**User Action:** Check to see whether the file is in the specified CMS library. If it is, make sure it is closed and try running MMS again. If the file is not in the CMS library, correct the library.

CMSPROCED, Proceeding with CMS library access.

**Explanation:** MMS is now accessing the specified CMS library.

**User Action:** None. This is an informational message that appears after the CMSWAIT message when MMS finally succeeds in accessing the library.

CMSWAIT, CMS library 'library name' is in use. Please wait . . .

**Explanation:** The specified CMS library is currently being accessed by another user. This message is printed at 4-second intervals until access is successful.

**User Action:** Wait until MMS succeeds in accessing the CMS library.

DRVBADPARSE, Parser detected a fatal syntax error in the description file.

**Explanation:** The description file contains a syntax error. MMS did not attempt to build the system.

**User Action:** Correct the erroneous line in the description file.

DRVDEPFIL, Using description file 'filespec.'

**Explanation:** MMS is using the specified description file to build the system.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

DRVFMSSUP, DEC/MMS is installed with support for VAX FMS.

**Explanation:** You can access forms stored in VAX FMS libraries with this version of MMS.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

DRVINSQUO, Your process needs a 'quota name' of at least 'value,' current value is 'value.'

**Explanation:** At least one of the process quotas set by your system manager has been exceeded, and the remaining process quotas at the time MMS was invoked were insufficient to run MMS reliably. The BYTLM value relates to the buffered I/O byte count quota; the ASTLM value relates to the AST limit of your process; the PRCLM value relates to the subprocess limit of your process; and the FILLM value relates to the open file limit of your process. You can obtain information about your process-specific parameters by typing the DCL command SHOW PROCESS/QUOTA.

**User Action:** Request that your system manager increase process quotas.

DRVNOFMSSUP, DEC/MMS is installed without support for VAX FMS.

**Explanation:** You cannot access forms stored in VAX FMS libraries because you did not install FMS before you installed MMS on your system.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

DRVOUTFIL, Using output file 'filespec.'

**Explanation:** MMS is writing all action lines and their resulting output to the specified output file. Note that messages preceded by "%MMS" are not written into this file, but to SYS$ERROR.

**User Action:** None. This is an informational message. It appears only if you have invoked MMS with the /LOG qualifier.

DRVPARSERR, Parser error: 'message' in file 'filename,' line 'number.'

**Explanation:** The MMS parser failed, for the reason explained in the message text.

**User Action:** Correct the erroneous action line in the description file.

DRVQUALIF, Using non-defaulted qualifiers 'qualifier name.'

**Explanation:** MMS is processing your description file using the specified qualifiers. These qualifiers, which are not enabled by default, correspond to the value of the $(MMSQUALIFIERS) reserved macro.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

DRVRULFIL, Using rules file 'filespec.'

**Explanation:** MMS is reading its default rules from the specified rules file.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

DRVSUBCLI, Using 'CLI name' for the subprocess CLI.

**Explanation:** MMS is using the specified CLI to execute the subprocess.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

EXEBADREAD, Could not read command output from subprocess.

**Explanation:** The MMS main process was unable to read the results of the action lines executed by the subprocess.

**User Action:** This message indicates a problem with your system, resulting possibly from insufficient quotas or a mailbox problem. Check with your system manager.

EXEBADWRT, Could not write command line to subprocess.

**Explanation:** The MMS main process was unable to send an action line to the subprocess for execution.

**User Action:** This message indicates a problem with your system, resulting possibly from insufficient quotas or a mailbox problem. Check with your system manager.

EXECANTWAKE, Could not wake up main process.

**Explanation:** After executing an action line, the subprocess was unable to wake up the main process.

**User Action:** This message indicates a problem with your system, resulting possibly from insufficient quotas or a mailbox problem. Check with your system manager.

EXEDELPROC, Subprocess terminated abnormally.

**Explanation:** The subprocess terminated unexpectedly, possibly because you used illegal commands like STOP or LOGOUT in your description file or because the subprocess was stopped by another process.

**User Action:** Remove any invalid commands from the description file.

EXEDELSES, Cleanup of subprocess %X'value' failed.

**Explanation:** The $DELPRC system service could not delete the subprocess that was executing action lines.

**User Action:** Type the DCL command SHOW SYSTEM/SUB to determine whether the subprocess still exists. If it does, type the STOP command to delete it: STOP/ID='value'. If the subprocess does not still exist and this message was preceded by the message %MMS-F-EXEDELPROC, the subprocess was probably deleted by a user command such as LOGOUT. If this is the case, remove the offending command from the description file.

EXENCRE, Could not create subprocess for executing action lines.

**Explanation:** MMS could not create the subprocess for executing action lines.

**User Action:** Check your quotas, and raise them if necessary. This message could also indicate a system problem; you should notify your system manager.

EXENEF, Unable to allocate event flag.

**Explanation:** MMS was unable to allocate an event flag that allows the MMS main process to communicate with the subprocess.

**User Action:** This message indicates a system problem. Check with your system manager.

EXENOAST, Could not enable AST.

**Explanation:** MMS could not enable an AST that allows the main MMS process to send input to the subprocess and the subprocess to send output back to the main process.

**User Action:** This message indicates a system problem. Check with your system manager.

EXENOMBX, Unable to create mailbox for communicating with subprocess.

**Explanation:** MMS could not create a mailbox for the MMS main process to use in communicating with the subprocess.

**User Action:** This message indicates a problem with your process's creation of mailboxes. Check with your system manager.

EXEPROCID, PID of created subprocess is %X'value.'

**Explanation:** The process ID of your subprocess is the value specified in the message.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

EXESTRING, Quoted string must be under 'value' characters.

**Explanation:** A quoted string in an action line exceeded the maximum length allowed.

**User Action:** Correct the action line in the description file.

EXETOOBIG, Command too large. Maximum length is 'value' characters.

**Explanation:** The command on an action line exceeded the maximum command length allowed.

**User Action:** Correct the command in the description file.

FMSNOSUPP, DEC/MMS is installed without VAX FMS support.

**Explanation:** Your description file specifies a form in an FMS library but you installed MMS without FMS support.

**User Action:** VAX FMS must already be installed on your system before you install MMS if you want access to FMS forms.

FMSNOWILD, Wild cards are not allowed for VAX FMS library access.

**Explanation:** You cannot use a wild card character in the specification of an FMS form.

**User Action:** Correct the description file to specify the forms you want MMS to access.

GFBTYPEMIX, Illegal single/double colon rule mix for 'item' in line 'number.'

**Explanation:** The item named was specified as a target in both a single colon and a double colon dependency rule.

**User Action:** Choose the rule you want for the build and make the description file consistent with respect to this target.

GMBADMOD, Missing left parenthesis in library specification 'filespec.'

**Explanation:** A library specification is missing a left parenthesis.

**User Action:** Insert the missing parenthesis.

GMTIMFND, Time for 'filespec' is 'time.'

**Explanation:** The specified time is the latest revision time MMS found for the specified file.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

GWKBEGWLK, Starting the build at target 'target name.'

**Explanation:** MMS will start its build process by trying to update the specified target.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

GWKCANT, MMS cannot update target 'target name.'

**Explanation:** MMS cannot update the specified target because neither the description file nor the built-in rules indicate how to do it. Because you instructed MMS to ignore severe errors (using either .IGNORE or /IGNORE=FATAL), processing of the description file continues.

**User Action:** Revise the description file. Either remove the dependency on the target or describe how to update the target.

GWKCONECT, Target 'target name' found in circular dependency.

**Explanation:** The specified targets are involved in a circular dependency; that is, a source depends on its target. This message is always issued after the GWKLOOP message, which indicates the target for which a circular dependency was detected in the description file.

**User Action:** Revise the description file to remove circular dependencies.

GWKCURRNT, Target 'target name' is already up-to-date.

**Explanation:** MMS has not updated the specified target because it is already up-to-date.

**User Action:** None. This is an informational message.

GWKEXESTS, Status of executed command is %X'condition code.'

**Explanation:** MMS has executed a CLI command in an attempt to update a target. The resulting condition code of the command is displayed in this message, and MMS attempts to decode its text in the following message line. If the next message line is blank, MMS cannot decode the message.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

GWKHSHOVER, Internal Hashtable Overflow.

**Explanation:** This is an MMS internal error.

**User Action:** Collect as much information as possible and submit a Software Performance Report (SPR).

GWKLOOP, Circular dependency detected at target 'target name.'

**Explanation:** The specified target is indirectly its own source. That is, you are asking MMS to make a target from the target itself, which is not legal. The ensuing GWKCONECT messages specify all relevant targets involved in the circular dependency.

**User Action:** Revise the description file to remove circular dependencies.

GWKNEEDUPD, An update is required for target 'name.'

**Explanation:** This message is issued when /CHECK_STATUS is specified.

**User Action:** None. This is an informational message.

GWKNOACTS, Actions to update 'target name' are unknown.

**Explanation:** MMS cannot determine what actions to take in updating the specified target. This message may indicate a problem with the .SUFFIXES list or with your user-defined rules. There may be no built-in rule or user-defined rule for MMS to use. The file types in the user-defined rule might not be in the .SUFFIXES list, or they might be in the wrong order.

**User Action:** Revise the description file. Specify the actions needed to update the target.

GWKNOPRN, There are no known sources for the current target 'target name.'

**Explanation:** MMS has found no sources for the current target.

**User Action:** Create a source file that can update the target.

GWKNOREV, Cannot update modification time for file 'filespec.'

**Explanation:** MMS is unable to modify the revision time of the specified file, as directed by the /REVISE_DATE qualifier on the command line, because an error occurred. A possible reason for the error is that the file's protection prohibited write access.

**User Action:** Correct the file protection so that write access is allowed.

GWKOLDNOD, Target 'target name' is older than 'source names.'

**Explanation:** The specified target is older than the specified sources, so MMS will update it.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

GWKUPDONE, Completed update for target 'target name.'

**Explanation:** MMS has updated the specified target.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

GWKUPDTIM, Updating modification time for file 'filespec.'

**Explanation:** MMS is changing the revision time of the specified file, as directed by the /REVISE_DATE qualifier.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier, in addition to /REVISE_DATE.

GWKWILLEX, MMS will try executing action line to update target 'target name.'

**Explanation:** MMS will execute the action line to update the current target for one of the following reasons: at least one source may be more recent than the target, or the target may have no sources.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

IDENT, DEC/MMS 'version' COPYRIGHT (C) DIGITAL EQUIPMENT CORPORATION 'date'

**Explanation:** This message provides the version number and copyright date of the MMS image installed on your system. You should include this information with any Software Performance Reports (SPRs) that you submit about MMS.

**User Action:** None. This is an informational message that appears only if you have invoked MMS with the /IDENTIFICATION qualifier.

INTERNERR, Internal MMS Error. Please Report Error #'number.'

**Explanation:** An MMS internal component failed.

**User Action:** Collect as much information as possible and submit a Software Performance Report (SPR).

LBRNOELEM, Illegal library element is specified in 'filespec.'

**Explanation:** You used incorrect syntax to specify a library module.

**User Action:** Correct the module syntax in the description file.

LEXFILELOOP, Included file 'filespec' is already open.

**Explanation:** An included file is itself included at some deeper level. If undetected, this situation would cause an infinite sequence of included files.

**User Action:** Remove the second level of inclusion in the file.

LEXIFERR, Encountered .ENDIF without matching .IFDEF.

**Explanation:** MMS found an .ENDIF directive in your description file but no corresponding .IFDEF directive.

**User Action:** Correct the description file to remove the .ENDIF directive or add the necessary .IFDEF.

LEXILLNAME, Specified target name 'target' on line 'number' is illegal.

**Explanation:** You used incorrect syntax to indicate the target on the specified line number of the description file.

**User Action:** Correct the description file.

LEXUNEXEND, Continuation character found at end of file.

**Explanation:** MMS found a hyphen (-) or a backslash (\) continuation character at the end of the description file.

**User Action:** Revise the description file. Delete the continuation character or add another line.

LFSBADFP, Cannot find source for target 'filespec.'

**Explanation:** MMS cannot process an invalid file specification. This error can occur if you specified an undefined logical name as the target.

**User Action:** Correct the syntax of the file specification and invoke MMS again.

MBBADMODE, Unknown mode parameter 'mode number.'

**Explanation:** An internal MMS component failed.

**User Action:** Collect as much information as possible and submit a Software Performance Report (SPR).

MBREDEFILL, Illegal attempt to redefine macro 'macro name.'

**Explanation:** You attempted to redefine the specified macro in the description file. You cannot define the same macro twice in one description file. The attempt is ignored, and the original definition applies.

**User Action:** If you want to redefine a macro, you must use the /MACRO qualifier on the MMS command line.

NOACCESS, Unable to access file 'file'.

**Explanation:** This message is followed by one or more messages describing why the file could not be accessed.

**User Action:** Modify the file protection of the inaccessible file.

NOLIBSPECDBL, Library module specifications not allowed as targets in double colon rules: 'filespec'.

**Explanation:** You used a library module specification as a target in a double colon dependency rule.

**User Action:** Rewrite the dependency as a single colon dependency using the library module specification or use only the library file name in your double colon dependency rule. You can write the preferred single colon syntax by using library module specifications. For example:

```
UTIL(MOD1) : MOD1.OBJ
   LIBR UTIL.LIB MOD1.OBJ

UTIL(MOD2) : MOD2.OBJ
   LIBR UTIL.LIB MOD2.OBJ
```

The following dependency rule is correct for the double colon use:

```
UTIL.OLB :: MOD1.OBJ
    LIBR UTIL.LIB MOD1.OBJ

UTIL.OLB :: MOD2.OBJ
    LIBR UTIL.LIB MOD2.OBJ
```

NOMACFIL, Cannot open macro file 'filespec.'

**Explanation:** You specified either an illegal or a nonexistent file in the command line macro definitions.

**User Action:** Create the file, or correct the file specification.

NOOUTFIL, Cannot open output file 'filespec.'

**Explanation:** MMS failed to create the output file.

**User Action:** Verify that the file specification is legal, check your disk quota, or check the protection of an existing file of the same name as the output file.

NOSTATUS, Unable to set MMS$STATUS to 'value.'

**Explanation:** MMS received an error from VMS when trying to set the symbol MMS$STATUS. This error may occur if you have exceeded the available space for symbols defined by your process, or if symbol scope is set to noglobal.

**User Action:** Either remove some of your symbols or have the system manager change the SYSGEN parameter CLISYMTBL or set symbol scope to global.

NOTARGET, No target specified.

**Explanation:** You did not specify a target for MMS to build.

**User Action:** Specify a target on the MMS command line, or correct the description file so that it specifies a target.

UTLALLOCFAIL, Failed to allocate memory for dynamic data structures.

**Explanation:** An MMS call to obtain more virtual memory failed. Either your description file is too large or a system service failed unexpectedly.

**User Action:** Try trimming your description file. If this fails, consult your system manager about increasing the size of virtual address space available to your system processes. If this fails, submit a Software Performance Report (SPR).

UTLBADMAC, Unterminated macro name 'string.'

**Explanation:** The character combination $( was encountered without a matching closing parenthesis. As a result, on the line that contains the offending macro, all characters to the right of the $( are ignored.

**User Action:** Correct the erroneous line.

UTLUNDERFLOW, Deallocation of unallocated space.

**Explanation:** This is an internal MMS error.

**User Action:** Collect as much information as possible and submit a Software Performance Report (SPR).

# Appendix B

# MMS and UNIX *make* Comparisons

This appendix briefly compares the characteristics of MMS and UNIX *make* Version 7. It is designed to ease the transition to MMS for users already familiar with *make*.

Because VMS and UNIX are very different operating systems, certain system-imposed changes were necessary to provide the features of *make* on VMS. The experienced user of *make* will notice the following differences:

- In the absence of a /DESCRIPTION or /NODESCRIPTION qualifier, MMS looks first for the description file DESCRIP.MMS. It looks for MAKEFILE. only if it cannot locate DESCRIP.MMS.
- In the target or source line of a dependency rule, there must be at least one space or tab on either side of the colon or double colon that separates the list of targets from the list of sources. The space or tab prevents MMS from trying to interpret the colon or colons as part of a VMS file specification.
- MMS allows you to use commas as well as spaces to separate the elements in a list of targets or sources.
- MMS allows either a number sign (#) or an exclamation point (!) to be used as a comment character. On target or source lines, as well as on blank lines that separate dependency rules, the number sign and the exclamation point can be used interchangeably; however, on action lines, only the exclamation point may be used to indicate a comment.
- In MMS, subprocesses are not executed independently of one another. Therefore, it is possible to define logical names, change directories, and in general manipulate the subprocess environment at will.
- The dummy target .PRECIOUS, found in *make*, is not implemented in MMS.
- When invoking a macro in MMS, you must enclose the macro name in parentheses. That is, $(A) is a legal invocation of an MMS macro, but $A is not.

- MMS action lines may begin with either a space or a tab, and MMS assumes that any line that begins with a space or tab is an action line (unless the preceding line ends with a continuation character).
- MMS has different built–in rules from those of *make*. See Table C–6 for the format and contents of MMS built–in rules.
- MMS requires you to separate the Silent (@) and Ignore (-) action line prefixes from the rest of the action line by at least one space.
- In a description file, the line continuation character can be either a hyphen (-) or a backslash (\). On the MMS command line, only the hyphen is legal.
- In the specification of a VMS library module, you can use the question mark (?) wildcard character as a synonym for the percent sign (%) wildcard.
- MMS allows an optional format for dependency rules:

```
PROG.OBJ DEPENDS_ON PROG.C

UTIL.LIB ADDITIONALLY_DEPENDS_ON MOD1.OBJ
```

  In this format. DEPENDS_ON is used in place of the colon and ADDITIONALLY_DEPENDS_ON is used in place of the double colon.

For compatibility with *make*, MMS provides alternative formats for dependency rules, user–defined rules, and directives, and recognizes 2–character abbreviations for special macros. The experienced user of *make* will recognize the following *make* features in MMS:

- MMS allows the following alternative format for dependency rules:

```
target . . .   [source . . . ] ; [action line . . . ]
```

  In this format, the only legal comment character is an exclamation point (!). You cannot use the Ignore (-) action line prefix with this format because the hyphen is interpreted as a line continuation character.
- MMS allows the following alternative format for user–defined rules:

```
.SRC.TAR : ; action line . . .
```

  In this format, you must include at least one space or tab on each side of the colon and the semicolon to prevent MMS from trying to interpret the rule as a file specification. You cannot use the Ignore (-) action line prefix with this format because the hyphen is interpreted as a line continuation character.
- The name of a directive can be followed by a colon. For example, you can specify either .SILENT or .SILENT : in a description file.

- The period preceding the .INCLUDE directive is optional.
- MMS special macros can be abbreviated to two characters; see Table C–3.

# Appendix C

# DEC/MMS Built–In Features

This appendix contains tables of certain MMS built–in features, namely, the suffixes precedence list, the built–in rules, and the default macros. Chapter 2 contains detailed information about how these three features work together in MMS.

The tables in this appendix are arranged as follows:

- Table C–1 lists the default macros.
- Table C–2 lists the changed default macros when you use the /SCA_ LIBRARY qualifier.
- Table C–3 lists the special macros.
- Table C–4 contains the suffixes precedence list.
- Table C–5 lists and describes the directives used in a description file.
- Table C–6 contains the standard built–in rules.
- Table C–7 describes the built–in rules for accessing VMS libraries.
- Table C–8 lists the built–in rules that change when you use the /SCA_ LIBRARY qualifier.
- Table C–9 includes the built–in rules for accessing CMS library elements.

For information on using MMS to create and access elements in VMS libraries, see Section 4.1; in CMS libraries, Section 4.2.

## C.1 MMS Default Macros

MMS uses default macros to build your system if none are specified or redefined.

**Table C–1: MMS Default Macros**

| Macro | Definition |
|---|---|
| ANLFLAGS | /OUTPUT=$(MMS$TARGET_NAME).ANL |
| AS | MACRO |
| BASIC | BASIC |
| BASFLAGS[1] | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| BLISS | BLISS |
| BLISS16 | BLISS/PDP11 |
| BFLAGS[1] | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| BLIBFLAGS[1] | /NOLIST |
| CC | CC |
| CFLAGS[1] | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| CDDFLAGS | null string |
| CLDFLAGS | null string |
| CMS | CMS |
| CMSCOMMENT | null string |
| CMSFLAGS | /GEN=$(MMS$CMS_GEN) |
| COBOL | COBOL |
| COBFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| CORAL | CORAL |
| CORFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| DIBOL | DIBOL |
| DBLFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| FMS | FMS |
| FMSFLAGS | /REPLACE |
| FORT | FORTRAN |

[1]This default macro changes when you use the /SCA_LIBRARY qualifier.

(continued on next page)

**Table C–1 (Cont.): MMS Default Macros**

| Macro | Definition |
|---|---|
| FFLAGS[1] | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| LIBR | LIBRARY |
| LIBRFLAGS | /REPLACE |
| LINK | LINK |
| LINKFLAGS | /TRACE/NOMAP/EXEC=$(MMS$TARGET_NAME).EXE |
| MACRO | MACRO |
| MFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| MSGFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| PASCAL | PASCAL |
| PENVFLAGS | /NOLIST |
| PFLAGS[1] | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| PLI | PLI |
| PLIFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| RPG | RPG |
| RPGFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ |
| RUNOFF | RUNOFF |
| RFLAGS | /OUTPUT=$(MMS$TARGET_NAME).OBJ |
| SCA | SCA |
| SCAFLAGS | /LOG |
| SCALIBRARY[1] | Not defined |

[1]This default macro changes when you use the /SCA_LIBRARY qualifier.

## C.2 Default Macro Changes with /SCA_LIBRARY

**Table C–2: /SCA_LIBRARY Default Macros**

| Macro | Definition |
|---|---|
| SCA | SCA |
| SCALIBRARY | library name from /SCA_LIBRARY qualifier |
| BASFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ/ANALYSIS_DATA=$(MMS$TARGET_NAME).ANA |
| BLIBFLAGS | /NOLIST/ANALYSIS_DATA=$(MMS$TARGET_NAME).ANA |
| BFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ/ANALYSIS_DATA=$(MMS$TARGET_NAME).ANA |
| CFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ/ANALYSIS_DATA=$(MMS$TARGET_NAME).ANA |
| FFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ/ANALYSIS_DATA=$(MMS$TARGET_NAME).ANA |
| PFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ/ANALYSIS_DATA=$(MMS$TARGET_NAME).ANA |
| PLIFLAGS | /NOLIST/OBJECT=$(MMS$TARGET_NAME).OBJ/ANALYSIS_DATA=$(MMS$TARGET_NAME).ANA |

## C.3 MMS Special Macros

Table C–3 lists the MMS special macros and describes their functions. The table also lists a symbol that you can use as an abbreviation for each macro.

**Table C–3: MMS Special Macros**

| Macro | Symbol | Meaning |
|---|---|---|
| MMS$TARGET | $@ | Expands to the mnemonic name or the complete file specification of the target currently being updated. |
| MMS$TARGET_NAME | $* | Expands to the mnemonic name or the file name (excluding the file type) of the target being updated. The device, directory, and node information are included. |
| MMS$SOURCE | $ < | Expands to the source file specification. |
| MMS$SOURCE_LIST | $+ | Expands to a comma list of the full file specifications of all sources specified in this dependency rule, including any sources implied by built–in rules. |
| MMS$CHANGED_LIST | $? | Expands to a comma list of the full file specifications of all sources that have changed since the target was updated, including any sources implied by built–in rules. |
| MMS$LIB_ELEMENT | $% | Expands to the name of a module in a VMS library and its file name, including the file type (see Section 4.1). |
| MMS$CMS_ELEMENT | $ < | Expands to the implicit CMS element specification (if the source file is a CMS element). |
| MMS$CMS_GEN | $& | Expands to the CMS generation specified by the source file (if the source is a CMS element). |
| MMS$CMS_LIBRARY | $@ | Expands to the CMS library specification (if the source is a CMS element). |

## C.4 MMS Suffixes Precedence List

**Table C–4: The Suffixes Precedence List**

| | |
|---|---|
| .SUFFIXES | .ANL .EXE .OLB .MLB .HLB .TLB .FLB .OBJ .BLI .B32 .C .COB .FOR .BAS .B16 .PLI .PEN .PAS .MAC .MAR .CLD .MSG .COR .DBL .RPG .MEM .RNO .HLP .RNH .L32 .REQ .R32 .L16 .R16 .TXT .H .FRM .MMS .DDL .COM .DAT .OPT .ANL~[1] .BAS~ .BLI~ .B32~ .B16~ .C~ .CLD~ .COB~ .COM~ .COR~ .DAT~ .DDL~ .FOR~ .FRM~ .HLP~ .H~ .MAC~ .MAR~ .MMS~ .DBL~ .MSG~ .OPT~ .PAS~ .PLI~ .REQ~ .R32~ .R16~ .RNH~ .RNO~ .RPG~ .TXT~ |

[1] A tilde ( ~ ) after a file type indicates that the file is in a CMS library. See Section 4.2 for information on using MMS to access CMS elements.

## C.5 MMS Directives

**Table C–5: MMS Directives**

| Directive | Function |
|---|---|
| .IGNORE | Causes MMS to ignore all errors generated by all action lines and to continue processing the description file. |
| .SILENT | Suppresses the writing of all action lines to the output file (whether to SYS$OUTPUT or to the file specified by the /OUTPUT qualifier). |
| .DEFAULT | Indicates actions to be performed if MMS built–in rules or user–defined rules do not specify how to update a target. |
| .SUFFIXES | Clears, adds to, or redefines the suffixes precedence list. |
| .INCLUDE | Includes the specified file in the description file. |
| .FIRST | Indicates actions to be performed before MMS has executed any action lines to update the target. |
| .LAST | Indicates actions to be performed after MMS has executed all the action lines that update the target. |
| .IFDEF | Causes subsequent lines of a description file to be processed only if the specified macro is defined. |
| .ELSE | Causes subsequent lines of a description file to be processed if the specified macro for the .IFDEF directive is undefined. |
| .ENDIF | Terminates the set of lines in the description file whose processing is controlled by .IFDEF or .ELSE. |

## C.6 MMS Built-In Rules

**Table C–6: MMS Built-In Rules**

| Source | Target | Action |
|---|---|---|
| .BAS[1] | .OBJ | $(BASIC) $(BASFLAGS) $(MMS$SOURCE) |
| .BLI[1] | .OBJ | $(BLISS) $(BFLAGS) $(MMS$SOURCE) |
| .B16[1] | .OBJ | $(BLISS16) $(BFLAGS) $(MMS$SOURCE) |
| .B32[1] | .OBJ | $(BLISS) $(BFLAGS) $(MMS$SOURCE) |
| .C[1] | .OBJ | $(CC) $(CFLAGS) $(MMS$SOURCE) |
| .CLD | .OBJ | SET COMMAND /OBJECT=$(MMS$TARGET_NAME) |
| | | $(CLDFLAGS) $(MMS$SOURCE) |
| .COB | .OBJ | $(COBOL) $(COBFLAGS) $(MMS$SOURCE) |
| .COR | .OBJ | $(CORAL) $(CORFLAGS) $(MMS$SOURCE) |
| .DBL | .OBJ | $(DIBOL) $(DBLFLAGS) $(MMS$SOURCE) |
| .EXE | .ANL | ANALYZE/IMAGE $(ANLFLAGS) $(MMS$SOURCE) |
| .FOR[1] | .OBJ | $(FORT) $(FFLAGS) $(MMS$SOURCE) |
| .MAC | .OBJ | $(MACRO) $(MFLAGS) $(MMS$SOURCE) |
| .MAR | .OBJ | $(MACRO) $(MFLAGS) $(MMS$SOURCE) |
| .MSG | .OBJ | MESSAGE $(MSGFLAGS) $(MMS$SOURCE) |
| .OBJ | .EXE | $(LINK) $(LINKFLAGS) $(MMS$SOURCE) |
| .OBJ | .ANL | ANALYZE/OBJECT $(ANLFLAGS) $(MMS$SOURCE) |
| .PAS[1] | .OBJ | $(PASCAL) $(PFLAGS) $(MMS$SOURCE) |
| .PAS | .PEN | $(PASCAL) /ENVIRON=$(MMS$TARGET) |
| | | $(PENVFLAGS) $(MMS$SOURCE) |
| .PLI[1] | .OBJ | $(PLI) $(PLIFLAGS) $(MMS$SOURCE) |
| .RNH | .HLP | $(RUNOFF) $(RFLAGS) $(MMS$SOURCE) |
| .RNO | .MEM | $(RUNOFF) $(RFLAGS) $(MMS$SOURCE) |

[1]The use of the /SCA_LIBRARY qualifier changes some of these built–in rules. See Table C–8 for a list of rules changes.

(continued on next page)

**Table C–6 (Cont.): MMS Built-In Rules**

| Source | Target | Action |
|---|---|---|
| .REQ[1] | .L32 | $(BLISS) /LIBRARY=$(MMS$TARGET) |
| | | $(BLIBFLAGS) $(MMS$SOURCE) |
| .RPG | .OBJ | $(RPG) $(RPGFLAGS) $(MMS$SOURCE) |
| .R16[1] | .L16 | $(BLISS16) /LIBRARY=$(MMS$TARGET) |
| | | $(BFLAGS) $(MMS$SOURCE) |
| .R32[1] | .L32 | $(BLISS) /LIBRARY=$(MMS$TARGET) |
| | | $(BFLAGS) $(MMS$SOURCE) |

[1]The use of the /SCA_LIBRARY qualifier changes some of these built-in rules. See Table C–8 for a list of rules changes.

## C.7 Built-In Rules for Library Files

**Table C–7: Built-In Rules for Library Files**

```
.HLP.HLB
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
        THEN $(LIBR)/CREATE/HELP $(MMS$TARGET)
     $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
```

**To build a macro library:**

```
.MAR.MLB
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
        THEN $(LIBR)/CREATE/MAC $(MMS$TARGET)
     $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)

.MAC.MLB
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
        THEN $(LIBR)/CREATE/MAC $(MMS$TARGET)
     $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
```

(continued on next page)

## Table C–7 (Cont.): Built-In Rules for Library Files

**To build an object library:**

```
.OBJ.OLB
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
        THEN $(LIBR)/CREATE $(MMS$TARGET)
     $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)


.TXT.TLB
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
        THEN $(LIBR)/CREATE/TEXT $(MMS$TARGET)
     $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
```

**To build an FMS library:**

```
.FRM.FLB
     IF "''F$SEARCH("$(MMS$TARGET)")'" .NES. "" -
        THEN $(FMS)/LIBRARY $(FMSFLAGS) $(MMS$TARGET) $(MMS$SOURCE)

     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
        THEN $(FMS)/LIBRARY/CREATE $(MMS$TARGET) $(MMS$SOURCE)
```

# C.8 MMS Built–In Rules for /SCA_LIBRARY Qualifier

**Table C–8: Changes to Built-In Rules When Using the /SCA_LIBRARY Qualifier**

```
.BAS.OBJ :
     $(BASIC) $(BASFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN-
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND.-
          mms$scalib .NES. "$(SCALIBRARY)" THEN  mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'


.BLI.OBJ :
     $(BLISS) $(BFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
          mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
```

(continued on next page)

**Table C–8 (Cont.): Changes to Built-In Rules When Using the /SCA_LIBRARY Qualifier**

```
.B32.OBJ :
     $(BLISS) $(BFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
          mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'


.C.OBJ :
     $(CC) $(CFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
          mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS)
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS)
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
```

(continued on next page)

**Table C–8 (Cont.): Changes to Built-In Rules When Using the /SCA_LIBRARY Qualifier**

```
.FOR.OBJ :
     $(FORT) $(FFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
          mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'


.PAS.OBJ :
     $(PASCAL) $(PFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
          mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
```

(continued on next page)

**Table C–8 (Cont.): Changes to Built-In Rules When Using the /SCA_LIBRARY Qualifier**

```
.PLI.OBJ :
     $(PLI) $(PLIFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
          mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'


.REQ.L32 :
     $(BLISS)/LIBRARY=$(MMS$TARGET) $(BLIBFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
          mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
```

(continued on next page)

**Table C–8 (Cont.): Changes to Built-In Rules When Using the /SCA_LIBRARY Qualifier**

```
.R32.L32 :
     $(BLISS)/LIBRARY=$(MMS$TARGET) $(BLIBFLAGS) $(MMS$SOURCE)
     mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
     mms$scasetlib = 0
     IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
          mms$scasetlib = 2
     IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
          mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
     IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
          mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
     IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
          $(SCALIBRARY)
     $(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
     IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
```

## C.9 MMS Built-In Rules for CMS Access

The following table lists the rules for accessing specific libraries when using MMS.

### Table C–9: Built-In Rules for CMS Access

**CMS Rules**

```
.ANL~.ANL :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).ANL -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.BAS~.BAS :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
         $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).BAS -
         $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
         THEN $(CMS) SET LIBRARY 'mms$cmslib'


.BLI~.BLI :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
         $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).BLI -
         $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
         THEN $(CMS) SET LIBRARY 'mms$cmslib'
```

(continued on next page)

## Table C–9 (Cont.): Built-In Rules for CMS Access

**CMS Rules**

```
.B16~.B16 :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
         $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).B16 -
         $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
         THEN $(CMS) SET LIBRARY 'mms$cmslib'


.B32~.B32 :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
         $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).B32 -
         $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
         THEN $(CMS) SET LIBRARY 'mms$cmslib'


.C~.C :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).C -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.CLD~.CLD :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).CLD -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'
```

(continued on next page)

## Table C–9 (Cont.): Built-In Rules for CMS Access

**CMS Rules**

```
.COB~.COB :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).COB -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.COR~.COR :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).COR -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.COM~.COM :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).COM -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.DAT~.DAT :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).DAT -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'
```

(continued on next page)

## Table C–9 (Cont.): Built-In Rules for CMS Access

**CMS Rules**

```
.DBL~.DBL :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).DBL -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.DDL~.DDL :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).DDL -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.FOR~.FOR :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).FOR -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.FRM~.FRM :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).FRM -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'
```

(continued on next page)

## Table C–9 (Cont.): Built-In Rules for CMS Access

**CMS Rules**

```
.H~.H :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).H -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.HLP~.HLP :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).HLP -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.MAC~.MAC :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MAC -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.MAR~.MAR :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MAR -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'
```

(continued on next page)

## Table C–9 (Cont.): Built-In Rules for CMS Access

**CMS Rules**

```
.MMS~.MMS :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MMS -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.MSG~.MSG :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MSG -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.OPT~.OPT :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).OPT -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.PAS~.PAS :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).PAS -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'
```

(continued on next page)

## Table C–9 (Cont.): Built-In Rules for CMS Access

**CMS Rules**

```
.PLI~.PLI :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).PLI -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.REQ~.REQ :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).REQ -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.RNH~.RNH :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RNH -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.RNO~.RNO :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RNO -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'
```

(continued on next page)

**Table C–9 (Cont.): Built-In Rules for CMS Access**

**CMS Rules**

```
.R16~.R16 :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).R16 -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.R32~.R32 :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).R32 -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.RPG~.RPG :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RPG -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'


.TXT~.TXT :
        mms$cmslib :== 'f$trnlnm("CMS$LIB")
        IF mms$cmslib .nes. "$(MMS$CMS_LIBRARY)" THEN $(CMS) SET LIBRARY -
          $(MMS$CMS_LIBRARY)
        $(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).TXT -
          $(CMSFLAGS) $(CMSCOMMENT)
        IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "$(MMS$CMS_LIBRARY)" -
          THEN $(CMS) SET LIBRARY 'mms$cmslib'
```

A tilde ( ~ ) after a file type indicates that the file is in a CMS library.

# Glossary

**action**

A command-language command that MMS executes to update a target. (See also *action line.*)

**action line**

The part of the dependency rule that contains the commands that tell MMS how to use the source or sources to update the target. (See also *dependency rule.*)

**action line prefix**

A special character placed at the beginning of an action line that influences how MMS executes the action. (See also *action line.*)

**built-in rule**

A command that MMS uses when the description file does not explicitly describe the processing needed to update a target.

**default macro**

A name that represents a character string that defines commonly used operations. MMS built-in rules are expressed in terms of default macros. (See also *macro* and *built-in rule.*)

**dependency rule**

The description of a relationship between a target and one or more sources, and the action or actions required to update the target. Dependency rules are contained in the description file. (See also *description file.*)

**description file**

A text file that contains the dependency rules, comments, macros, and directives that MMS uses to build your system. (See also *dependency rule, macro,* and *directive.*)

**directive**

A word that specifies an action for the processing of the description file.

**library module specification**

A VMS file specification for a module in a library.

**macro**

A name that represents a character string. The string is substituted for the name in a dependency rule.

**macro invocation**

The execution of the macro that replaces the macro name with its definition.

**mnemonic name**

A name that identifies the purpose of a sequence of related actions. It can be used as either a source or a target in the description file. (See also *source* and *target*.)

**source**

In a dependency rule, an entity that is used to create or to update the target. A source can be a VMS file specification or a mnemonic name. (See also *dependency rule* and *mnemonic name*.)

**special macro**

A name that represents a character string that expands to source or target names in the dependency currently being processed. A special macro is used instead of a target or source file specification when writing general user-defined rules. (See also *target, macro, dependency rule*.)

**suffixes precedence list**

A list of file types to which MMS refers when it needs to know which source file can update the specified target.

**target**

In a description file, the entity that is to be updated. A target can be a VMS file specification or a mnemonic name. (See also *mnemonic name*.)

**user-defined rule**

A rule that the user specifies in the description file to add to and/or replace MMS built-in rules. (See also *built-in rule*.)

# Index

## A



## B



## C

# D

## E



## F

## N



## O



## P



## Q

## R



## S

## T



## U



## V



## W

# How to Order Additional Documentation

## Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

## Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

## Telephone and Direct Mail Orders

| Your Location | Call | Contact |
|---|---|---|
| Continental USA, Alaska, or Hawaii | 800-DIGITAL | Digital Equipment Corporation<br>P.O. Box CS2008<br>Nashua, New Hampshire 03061 |
| Puerto Rico | 809-754-7575 | Local DIGITAL subsidiary |
| Canada | 800-267-6215 | Digital Equipment of Canada<br>Attn: DECdirect Operations KAO2/2<br>P.O. Box 13000<br>100 Herzberg Road<br>Kanata, Ontario, Canada K2K 2A6 |
| International | ———— | Local DIGITAL subsidiary or approved distributor |
| Internal[1] | ———— | SDC Order Processing - WMO/E15<br>*or*<br>Software Distribution Center<br>Digital Equipment Corporation<br>Westminster, Massachusetts 01473 |

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

# Reader's Comments

**Guide to VAX DEC/Module Management System**
**Order number: AA–P119D–TE**

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

| **I rate this manual's:** | Excellent | Good | Fair | Poor |
|---|---|---|---|---|
| Accuracy (software works as manual says) | ☐ | ☐ | ☐ | ☐ |
| Completeness (enough information) | ☐ | ☐ | ☐ | ☐ |
| Clarity (easy to understand) | ☐ | ☐ | ☐ | ☐ |
| Organization (structure of subject matter) | ☐ | ☐ | ☐ | ☐ |
| Figures (useful) | ☐ | ☐ | ☐ | ☐ |
| Examples (useful) | ☐ | ☐ | ☐ | ☐ |
| Index (ability to find topic) | ☐ | ☐ | ☐ | ☐ |
| Page layout (easy to find information) | ☐ | ☐ | ☐ | ☐ |

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

| Page | Description |
|---|---|
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

-- 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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here ----------

Cut Along Dotted Line

# Reader's Comments

**Guide to VAX DEC/Module Management System**
**Order number: AA–P119D–TE**

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

| **I rate this manual's:** | Excellent | Good | Fair | Poor |
|---|---|---|---|---|
| Accuracy (software works as manual says) | ☐ | ☐ | ☐ | ☐ |
| Completeness (enough information) | ☐ | ☐ | ☐ | ☐ |
| Clarity (easy to understand) | ☐ | ☐ | ☐ | ☐ |
| Organization (structure of subject matter) | ☐ | ☐ | ☐ | ☐ |
| Figures (useful) | ☐ | ☐ | ☐ | ☐ |
| Examples (useful) | ☐ | ☐ | ☐ | ☐ |
| Index (ability to find topic) | ☐ | ☐ | ☐ | ☐ |
| Page layout (easy to find information) | ☐ | ☐ | ☐ | ☐ |

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

| Page | Description |
|---|---|
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

– 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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

– Do Not Tear - Fold Here ----------

Cut Along Dotted Line