VAX DEC/MMS
User's Guide

Order No. AA-P119B-TE

August 1984

This manual describes how to use DEC/MMS (Module Management System) for building software systems.

REVISION/UPDATE INFORMATION:	This revised document supersedes the VAX-11 DEC/MMS User's Guide (Order No. AA-P119A-TE).
OPERATING SYSTEM AND VERSION:	VAX/VMS Version 3.4 or later.
SOFTWARE VERSION:	DEC/MMS Version 2.0

digital equipment corporation · maynard, massachusetts

First Printing, March 1983
Revised, August 1984

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

The following are trademarks of Digital Equipment Corporation:

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

digital

ZK2594

HOW TO ORDER ADDITIONAL DOCUMENTATION

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

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

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

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

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

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

DIRECT MAIL ORDERS (CANADA)

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

DIRECT MAIL ORDERS (INTERNATIONAL)

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

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

Contents

Page

Preface vii

Chapter 1 Introduction to DEC/MMS

1.1 MMS and the Software Development Cycle 1-1
1.2 What MMS Does 1-2
1.3 How to Use MMS 1-4
1.4 How MMS Executes Commands 1-5
1.5 Process Requirements For Running MMS 1-5
1.6 Advantages of MMS Over DCL Command Procedures 1-6

Chapter 2 Creating a Description File

2.1 Dependency Rules 2-2
2.1.1 Dependency Rule Format 2-3
2.1.2 Source and Target Files 2-5
2.1.3 Action Lines 2-7
2.1.4 Examples 2-9
2.2 Built-In Rules 2-10
2.3 Macro Definitions 2-14
2.3.1 Format 2-15
2.3.2 Invoking Macros 2-15
2.3.3 Defining Macros on the Command Line 2-17
2.3.4 Default Macros 2-18
2.3.5 Using CLI Symbols as MMS Macros 2-19
2.4 Description File Example 2-20

Chapter 3 Advanced Description File Techniques

3.1 User-Defined Rules 3-1
3.2 Special Macros 3-2
3.3 Double Colon Dependencies 3-6
3.4 Directives 3-7
3.4.1 .IGNORE 3-8
3.4.2 .SILENT 3-10
3.4.3 .DEFAULT 3-11
3.4.4 .SUFFIXES 3-12
3.4.5 .INCLUDE 3-14
3.4.6 .FIRST 3-16
3.4.7 .LAST 3-17
3.4.8 .IFDEF, .ELSE, and .ENDIF 3-18

3.5 Action Line Prefixes 3-19
3.5.1 The Ignore Prefix (-) 3-20
3.5.2 The Silent Prefix (@) 3-20
3.6 Invoking MMS From a Description File 3-21

Chapter 4 Accessing Libraries With DEC/MMS

4.1 CREATING AND ACCESSING FILES IN VAX/VMS LIBRARIES 4-1
4.2 USING MMS WITH DEC/CMS 4-4
4.2.1 Using CMS Commands in a Description File 4-4
4.2.2 Automatic Access of CMS Elements from Dependency Rules 4-5
4.2.3 Explicit References to CMS Elements in Dependency Rules 4-6
4.2.4 Accessing Description Files in CMS Libraries 4-7
4.3 Accessing Forms in an FMS Library 4-8
4.4 Accessing Records in the CDD 4-8

Chapter 5 The MMS Command

/ACTION (D) 5-3
/CHECK_STATUS 5-4
/CMS 5-5
/DESCRIPTION (D) 5-6
/FROM_SOURCES 5-8
/HELP 5-9
/IDENTIFICATION 5-10
/IGNORE 5-11
/LOG 5-13
/MACRO 5-14
/OUTPUT 5-15
/OVERRIDE 5-16
/REVISE_DATE 5-17
/RULES (D) 5-18
/SKIP_INTERMEDIATE 5-19
/VERIFY (D) 5-21

Chapter 6 DEC/MMS Examples

6.1 Simple Uses of MMS 6-1
6.1.1 Checking Whether Files Are Up-to-Date 6-1
6.1.2 Using MMS to "Fetch and Build" 6-1
6.1.3 Using the /SKIP_INTERMEDIATE Qualifier 6-2
6.2 Description File Example 6-3
6.3 Gathering Statistics 6-4
6.3.1 Finding Out What Sources Are Missing 6-5
6.3.2 Creating a Checkpoint File 6-5
6.4 Using DCL Command Procedures in Description Files 6-7
6.5 Creating and Using Time Stamps 6-7
6.5.1 Creating a Time Stamp File Using DCL Symbols 6-8
6.5.2 Creating a Time Stamp File Using Included Files 6-9
6.6 Checking for Replacement of CMS Elements 6-11

6.7 Selectively Deleting Files 6-12
6.7.1 Creating a Command Procedure 6-12
6.7.2 Using a Macro Definition 6-13
6.8 Doing Parallel Processing 6-14

Appendix A DEC/MMS Built-In Features

Appendix B DEC/MMS and UNIX Make Comparisons

Appendix C DEC/MMS Messages

C.1 Message Format C-1
C.2 MMS Messages C-2

Glossary

Index

Figures

1-1 How MMS Builds a Software System 1-3
2-1 Dependencies of Files in a Software System 2-20

Tables

3-1 MMS Special Macros 3-3
3-2 MMS Directives 3-8
3-3 MMS Action Line Prefixes 3-20
A-1 The Suffixes Precedence List A-1
A-2 MMS Default Macros A-2
A-3 MMS Built-in Rules A-3
A-4 Built-in Rules for Library Files A-4
A-5 Built-in Rules for CMS Access A-5

Preface

Objective

This manual explains how to use DEC/MMS (Module Management System). It is a reference book with examples that illustrate both basic and advanced techniques.

Intended Audience

This manual is primarily intended for software engineers working on a wide range of software projects; however, others, such as managers and technical writers, can also use MMS. Those with UNIX[1] *make* experience should be able to use MMS with little trouble, since MMS is patterned after *make*.

DEC/MMS runs only on the VAX/VMS operating system Version 3.4 or later.

Structure of This Document

The *VAX DEC/MMS User's Guide* is divided into six chapters, three appendixes, and a glossary.

- Chapter 1, Introduction to DEC/MMS, describes how MMS automates the software development cycle, briefly explains how to use MMS, and describes how MMS executes commands.
- Chapter 2, Creating a Description File, presents the concepts of description files, built-in rules, and macro definitions.
- Chapter 3, Advanced Description File Techniques, describes techniques for using MMS as efficiently as possible.
- Chapter 4, Accessing Libraries With DEC/MMS, explains how MMS can process files stored in VAX/VMS, DEC/CMS, and VAX FMS libraries and records stored in the VAX Common Data Dictionary.
- Chapter 5, The MMS Command, presents the MMS command line format and contains detailed descriptions of all MMS qualifiers. The qualifier descriptions are listed alphabetically by qualifier name.
- Chapter 6, DEC/MMS Examples, illustrates MMS techniques.
- Appendix A, DEC/MMS Built-In Features, contains tables of MMS defaults, with explanatory information.

1. UNIX is a trademark of Bell Laboratories.

- Appendix B, DEC/MMS and UNIX *make* Differences, describes the differences between MMS and UNIX *make*.
- Appendix C, DEC/MMS Messages, lists and explains all MMS messages.
- The Glossary defines important terms.

Associated Documents

- The *VAX DEC/MMS Pocket Guide* (Order No. AV-P120B-TE) provides a concise summary of MMS rules and qualifiers.
- *Installing VAX DEC/MMS* (Order No. AA-P121B-TE) supplies the instructions for installing MMS on a VAX/VMS system.

Conventions Used in This Document

Convention	Meaning
[ ]	Square 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 item(s) 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 C.)
target	A term that appears in italics is defined in the Glossary.

Unless otherwise noted:

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

Summary of Technical Changes

This section summarizes the technical changes made to VAX DEC/MMS for Version 2.0:

- Better use of built-in rules and updated built-in macros.
- New directives .FIRST, .LAST, .IFDEF, .ELSE, and .ENDIF.
- New mnemonic synonyms for the MMS special macros.
- Three new qualifiers, /NODESCRIPTION, /FROM_SOURCES, and /HELP.
- Specification of a file that contains the default rules that MMS uses. This feature is provided by a parameter to the /RULES qualifier and the logical name MMS$RULES.
- Enhanced messages produced by the /LOG qualifier.
- Automatic access to description files and .INCLUDE files stored in VAX DEC/CMS libraries.
- Support for forms stored in VAX FMS (Forms Management System) libraries and records in the VAX CDD (Common Data Dictionary). You must have either VAX FMS or VAX CDD, Versions 2.1 or later, installed on your system.
- Support for wildcard characters in the specifications of VAX/VMS library modules.
- A new facility for managing subprocesses that allows quotas to be reduced. The minimum BYTLM quota has been reduced to 13000 if you use DEC/MMS recursively, and 8192 if you do not use DEC/MMS recursively.
- More error checking and more descriptive error messages.
- Printing of the erroneous line when a syntax error is detected in a description file.
- A new installation procedure using VMSINSTAL.

Chapter 1

Introduction to DEC/MMS

DEC/MMS (Module Management System) is a tool that automates and simplifies the building of software systems. MMS is useful for building both simple programs, which may have only one or two source files, and complex programs, which may consist of several source files, message files, and documentation. It can rebuild all the components in a system, or only those that have changed since the system was last built. And above all, MMS is also easy to use – with one command, you can build either a small or a large system.

If you have used the UNIX[1] *make* utility, you should easily be able to make the transition to MMS, since MMS is patterned after *make*. Appendix B describes the differences between UNIX *make* and MMS.

1.1 MMS and the Software Development Cycle

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

- Think about the problem and create a design
- Write code based on the design
- Build the system
- Test the software

MMS automates the build 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 DEC/CMS library). Each programmer usually

1. UNIX is a trademark of Bell Laboratories.

has copies of the sources, which he or she edits and then replaces. MMS can simplify this procedure in the following ways:

1. 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 updates them. Thus, the entire system is not rebuilt, only those components whose sources have been modified.

2. If you rebuild a complete system without regard to which components do not need to be updated, you waste disk space. MMS has a /SKIP_INTERMEDIATE qualifier that avoids unnecessary building of intermediate files, thus saving space. (The /SKIP_INTERMEDIATE qualifier is described in Chapter 5.)

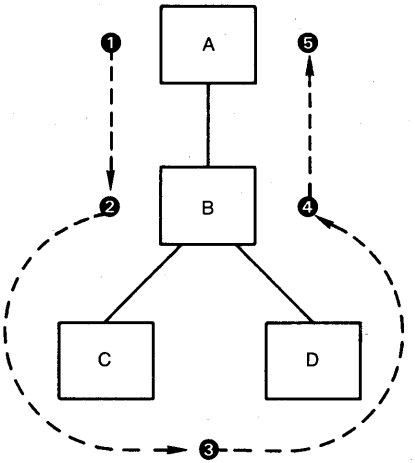
3. You can use MMS first to build and test modules locally, and then against the modules in the source directory (or library). Thus, you can test modules before you replace them and be sure that they work properly. If you use MMS in conjunction with DEC/CMS, you can be sure that the replaced modules do not conflict with the edits another programmer may have made. In fact, you can use these two tools together to ensure that only your changes (and not another programmer's) are included in the revised module.

DEC/CMS (Code Management System) is a tool that helps manage a project's files by storing them in a library, tracking changes, and monitoring access to the library that contains the files. Section 4.2 describes in detail how MMS and CMS can work together.

1.2 What MMS Does

MMS controls the efficient building of a software system by determining which components in the system have changed and then updating, or creating new versions of, only those files that depend on the changed components.

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. (For example, A might be an .EXE file, B might be an .OBJ file, and C and D might be .C or definition files.) 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.)



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 times of Components C and D against that of B.

❹ If the times of Components C and/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 to B because it is already up-to-date.

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

ZK-1090-82

Figure 1-1: How MMS Builds a Software System

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

1.3 How to Use MMS

When you use MMS to build your software system, you usually perform two steps:

1. Create a *description file*
2. Invoke MMS

The description file contains rules that describe how the components of your system are related, and the CLI commands that MMS is to use in building them. Once you have created your description file, you can use it every time you invoke MMS to build your system. In most cases, you will need a description file; however, if your system is very simple, MMS can build it even if you have no description file. (Description files are explained in detail in Chapter 2.)

When you invoke MMS, it looks in your current directory for a description file called DESCRIP.MMS, unless you have specified the /NODESCRIPTION qualifier to indicate that no description file is to be used. 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.

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.

Suppose that you have a source file named PROG.COB, which you want to use to create an executable file, PROG.EXE, and suppose that the intermediate file PROG.OBJ does not exist. All you have to do is type the MMS command with the /NODESCRIPTION qualifier (to indicate that no description file is to be used); then specify the executable file as the target, as shown in the following example:

```
$ MMS/NODESCRIPTION PROG.EXE
```

Because MMS does not use a description file in this example, it must rely on its default rules, or *built-in rules*, to build PROG.EXE. MMS knows that an .EXE file results from an .OBJ file, so it looks for PROG.OBJ in the current directory. Since PROG.OBJ does not exist, MMS uses another built-in rule that builds an .OBJ file from a .COB file and looks for PROG.COB. If PROG.COB is newer than PROG.EXE, MMS determines that PROG.EXE is not up-to-date. To make it up-to-date, MMS invokes the COBOL compiler to compile PROG.COB

and then invokes the VAX Linker to link PROG.OBJ, thus creating a new version of PROG.EXE. (Built-in rules are described in Chapter 2, and the /NODESCRIPTION qualifier is described in Chapter 5.)

Before you read further in this manual, you may want to see for yourself how MMS works. The following example shows how you might use MMS to build the file CHAPTER1.MEM using DIGITAL Standard Runoff (DSR) and the input file CHAPTER1.RNO:

```
$ MMS/NODESCRIPTION CHAPTER1.MEM
RUNOFF /OUTPUT=CHAPTER1 CHAPTER1.RNO
DIGITAL Standard Runoff Version V2.0-014: No errors detected
12 pages written to "USER$:[ANDERSON]CHAPTER1.MEM;1"
$
```

In this example, MMS applies the built-in rule that instructs it to use DSR when building .MEM files. By default, MMS attempts to locate the source by looking in the working directory for a file with the same name as the target and with the extension .RNO.

1.4 How MMS Executes Commands

MMS runs two processes as it updates a target. The first process, your current process, executes MMS. The second process, a spawned subprocess, executes the commands 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 the subprocess executes an action (such as a DCL command), the parent process waits until it is notified that the subprocess has finished executing commands. Therefore, if you monitor the parent process, do not be alarmed to find it idle.

If you invoke MMS from a description file with the $(MMS) reserved macro (see Section 3.6), another subprocess is used to execute the new invocation of MMS. The original subprocess is treated as a parent process for the subsequent MMS execution.

1.5 Process Requirements for Running MMS

When a subprocess is created, the VAX/VMS operating system automatically assigns it a portion of the quotas established for your main process. Some of

these quotas are shared among the main process and all its subprocesses; that is, the sum of these quotas cannot exceed the total originally assigned to the main process.

In some cases, the subprocess MMS creates to execute actions may not have sufficient BYTLM, ASTLM, PRCLM, and FILLM quotas. These quotas determine the number of bytes of buffered I/O and the number of ASTs, subprocesses, and open files your process can have. If MMS exceeds one or more of these quotas when it begins processing your description file, it issues an error message that tells you the current quotas and the minimum needed to continue processing. The quotas suggested in the error message are the minimum required to execute MMS, the subprocess that executes action lines, and one recursive invocation of MMS using the $(MMS) reserved macro. You must then ask your system manager to increase your process's quotas by the difference between the current quotas and the minimum required by MMS. If your quotas are higher than the minimum, MMS will be able to execute faster and more efficiently.

1.6 Advantages of MMS Over DCL Command Procedures

You can write DCL command procedures that perform functions similar to MMS functions, but the major advantage of MMS is the ease with which it can rebuild a software system. Writing a command procedure that determines whether one file is more recent than another (or whether one file exists at all in relation to another file) is not an easy task; but making such a command procedure straightforward requires your time and effort, and also requires considerable processing time. MMS is designed to rebuild only the parts of a software system whose sources have been modified since the last system build, and it does so faster and more easily than a comparable command procedure could. For example, the following MMS rule:

```
A : B
     ACTION
```

means, in DCL terms:

```
If (A is older than B) or
     (A does not exist but B exists)
THEN
     DO ACTION
```

As you can see, this relationship is expressed much more easily by an MMS rule than by DCL commands.

MMS has other advantages over DCL command procedures:

1. The order in which MMS processes the rules in a description file does not depend on the position of the rules in the description file. In DCL command procedures, the order of command execution is determined by the position of the command in the procedure or by a directive to transfer control to a specified location.
2. You can leave out some steps of an MMS description file because MMS built-in rules define certain commonly used actions. You cannot leave out any steps of a DCL command procedure.
3. MMS allows the "top-down" breakdown of a task. Thus, you can use mnemonic names at the beginning of a description file to specify the order in which tasks must be accomplished, and then specify the actions to accomplish those tasks further on in the description file. In DCL, you cannot break down a task into sub-tasks in a clean manner; you must specify the actions as you go along, scattering commands and structural information throughout the command procedure. With MMS, the structural information is all in one place, giving a clear representation of the system.

Chapter 2

Creating a Description File

A description file contains the information MMS needs to build your software system. It has two principal uses:

- To instruct MMS how to build your system
- To document the relationships among the various components of your system

Since the description file is simply an ASCII text file, you can create and modify it with any text editor. Once you have created the description file, you need issue only a simple MMS command to update your system. (The MMS command is described in detail in Chapter 5.)

The description file describes the process of building a single target. However, the process of building one file may produce several other files. For example, to build a new version of a compiler, MMS might need to recompile all the source files to produce object files, an image file, a message file, and an installation document.

A description file can contain five kinds of information:

- Dependency rules
- Comments (optional)
- Macro definitions (optional)
- Directives (optional)
- User-defined rules (optional)

This chapter describes dependency rules, comments, and macro definitions. Directives and user-defined rules are described in Chapter 3.

As explained in Chapter 1, MMS by default looks for a description file called DESCRIP.MMS, and then for one called MAKEFILE. If you need several

description files for different MMS applications, one of the following methods will help you keep better track of your description files:

- Create a separate subdirectory for each MMS application and store a description file named DESCRIP.MMS in each one.
- Give each description file a meaningful name and then specify the correct file each time you invoke MMS. You can use the /DESCRIPTION qualifier on the MMS command line to supply the name of a description file; the default file type is .MMS. Examples of the /DESCRIPTION qualifier appear throughout this manual. A complete discussion of /DESCRIPTION is contained in Chapter 5.

2.1 Dependency Rules

A description file always contains *dependency rules*. These rules describe the relationships among the files in a software system and specify the actions MMS will perform in building an up-to-date version of the system. Dependency rules indicate how files depend on, or are affected by, other files.

For example, if the file TEST.OBJ results from compiling TEST.BAS with the BASIC compiler, then TEST.OBJ depends upon TEST.BAS, and the action required to update TEST.OBJ is the command BASIC TEST. To put it in MMS terms, TEST.OBJ is the *target* that MMS will build from the *source*, TEST.BAS, by executing the *action line* BASIC TEST. You express such a relationship to MMS by writing a dependency rule.

For each dependency rule, MMS uses the last revision dates of the target and the source to determine whether the target should be updated. The revision date indicates the date and time the file was most recently changed. (You can find out this information for yourself by using the DCL command DIRECTORY/DATE = MODIFIED.)

If none of the sources have later revision dates than the target, MMS does not perform the action specified in the dependency rule. If, however, any of the sources have been modified since that target was last updated, or if the target does not exist, MMS performs the action that updates or creates the target.

NOTE

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

1. If sources are being stored in a library and more than one person is accessing a given source, someone else may replace that source in the library after you have invoked MMS but before MMS has checked the source's revision time. Thus, when MMS does check the time, a source newer than the corresponding target will exist in the library, which may cause MMS to execute an action to update the target when you did not expect the target to be out-of-date.
2. 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's solely because the clock on the source's node is slower.

These conditions are not likely to occur very often, but you should be aware that they are possible.

2.1.1 Dependency Rule Format

A dependency rule has the following format:

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

target, source

A VAX/VMS file specification or a *mnemonic name* (Section 2.1.2 describes mnemonic names). If you are using DECnet, the file specifications for the target and source can include node information.

comment

A string of text, introduced by an exclamation point (!), that documents the description file.

action line

A command-language command that MMS will use to update the target. You can specify any number of action lines for a target.

You must begin a target/source line in column 1 of the line, and you must include at least one space or tab on either side of the colon that separates the source from the target. This requirement is necessary so that MMS does not interpret the colon as part of a VAX/VMS file specification.

By default, MMS expects to find the source and target files in the current default directory. However, it can process files in other directories if you provide a complete file specification.

You can use a logical name for a source or a target; however, if you do so, you must supply the action lines that update the target because MMS relies on the source and target file types to apply built-in rules. Section 2.2 describes how MMS uses built-in rules.

An action line is positioned below the corresponding target/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/source line. You can omit the action line and MMS will use built-in rules to update the target if it can (see Section 2.2).

The following example shows the dependency rule for making TEST.OBJ:

```
TEST.OBJ : TEST.BAS    ! TEST.OBJ is updated by TEST.BAS
     BASIC TEST        ! Creates or updates the .OBJ file
```

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:

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

The hyphen means that the next line is treated as part of the current line. In the example, the second and third lines are continuations of the target/source line. Note, though, that a comment can appear after a continuation character without affecting the processing of the description file. 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.

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. 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. In that case, MMS searches the description file for the dependency rule associated with the specified target.

MMS must check all sources before it updates a target, since sources may themselves be targets with sources of their own in other dependency rules. Therefore, MMS updates all sources and their dependencies before updating the main target.

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

2.1.2 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, in the following dependency:

```
[DELANO]A.OBJ :
     PASCAL/DEBUG [DELANO]A.PAS
```

MMS executes the PASCAL command only if A.OBJ does not exist in the directory [DELANO].

As MMS checks the revision times of targets and sources, it builds a list of times used to allow it to decide 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 VAX/VMS.) All targets and sources that are not files are assigned this revision time.

To specify multiple targets and sources, you separate them with commas, spaces, or a combination of both. However, the specification of multiple targets is actually just a shorthand notation that MMS expands into separate dependencies before it executes the action lines. For example, suppose you include the following dependency rule in a description file:

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

Although no action line is specified, MMS can use built-in rules to determine what action is needed to update KERNEL.OBJ and DRIVER.OBJ. (Built-in rules are explained in Section 2.2.) Thus, MMS expands the previous rule to the following two:

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

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

Built-in rules also allow MMS to determine that KERNEL.C and DRIVER.C, which were not specified in the original dependency, are sources for KERNEL.OBJ AND DRIVER.OBJ, respectively. The original dependency rule therefore expands to two action lines, resulting in two compilations if both targets need to be updated.

Sometimes, however, 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:

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

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

Since the second dependency in the description file expands to two action lines, MMS executes the command BASIC/LIST A.BAS twice and produces two .OBJ files and two .LIS files.

You can use a mnemonic name to represent a target; you can also use a mnemonic name to represent a source, provided that the source is a target in another dependency rule. 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, since MMS by default builds only one target. If you actually need to update several targets, you can make them sources in a dependency rule where the target is a mnemonic name. For example:

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

A.EXE : A.OBJ
     LINK A.OBJ

B.EXE : B.OBJ
     LINK B.OBJ
```

The target (in this case, NEW_SYSTEM) is considered updated when MMS has executed the action line or lines that follow it. This technique guarantees that both A.EXE and B.EXE are updated, if necessary.

The following partial example shows the use of mnemonic names as both targets and sources:

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

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

PRINT : 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
```

MMS updates the target ALL by updating the two sources, PROG.EXE and PRINT, which are themselves targets in subsequent dependency rules.

2.1.3 Action Lines

Sometimes a target requires a series of actions to update it. In that case, you may follow a target/source line in the description file with one or more action lines, as in the following example:

```
RESULTS.DIF : ACCOUNTS.EXE, BENCHMARK.DAT
     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

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

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.) It then runs the program, runs the DIFFERENCES utility to compare the program's output with a master file, and displays the results of the comparison.

When you run MMS, all action lines, along with any comments that you specified on action lines in the description file, are written to SYS$OUTPUT or to the file specified by the /OUTPUT qualifier on the MMS command line. (The /OUTPUT qualifier is described in Chapter 5.) If you were to run MMS using the previous example as the description file BALANCE.MMS, you would see the following output:

```
$  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$:[ALISON]RESULTS.DIF1-
    USER$:[ALISON]ACCOUNTS.DAT;19-
    USER$:[ALISON]BENCHMARK.DAT;27
$
```

Action lines are subject to certain restrictions, as follows:

- An action line may not receive data from SYS$INPUT. For example, an action line cannot 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 may spawn a subprocess only by using the $(MMS) reserved macro (see Section 3.6). The DCL command SPAWN is not allowed in an action line.
- An action line may not use the DCL commands SET VERIFY or SET ON. You can use these commands in a command procedure that you invoke from an action line; however, if you use SET VERIFY you must be sure to issue SET NOVERIFY before the command procedure ends.

MMS uses a special symbol, MMS$STATUS, to record the return status of the last action line it executed. If the value of MMS$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, you can 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.

2.1.4 Examples

The following example shows a simple dependency rule:

```
NEWTESTS.OBJ : TEST1.BAS,TEST2.BAS
     BASIC/OBJECT=NEWTESTS TEST1+TEST2
```

This dependency rule states that the target, NEWTESTS.OBJ, depends on two sources, TEST1.BAS and TEST2.BAS. If either source has a later revision date than NEWTESTS.OBJ, MMS executes the action line and updates NEWTESTS.OBJ.

Most description files contain more than just one dependency rule. Sometimes a source in a dependency may itself depend on other files in the system. In that case, the source is a target in another dependency rule, so the description file for building the system must contain several rules. The dependency rules in the next example describe the structure of a system called MYSYSTEM.EXE.

```
USER$:[PROJECT]MYSYSTEM.EXE : TESTS.OBJ KERNEL.OBJ DRIVER.OBJ
     LINK/EXE=USER$:[PROJECT]MYSYSTEM  TESTS.OBJ,-
          KERNEL.OBJ,DRIVER.OBJ

TESTS.OBJ : TEST1.BAS, TEST2.BAS
     BASIC/OBJECT=TESTS TEST1+TEST2

KERNEL.OBJ : KERNEL.BAS
     BASIC KERNEL

DRIVER.OBJ : DRIVER.BAS
     BASIC DRIVER
```

The first dependency rule defines the top level of the system. That is, the target MYSYSTEM.EXE in the directory USER$:[PROJECT] is composed of the sources TESTS.OBJ, KERNEL.OBJ, and DRIVER.OBJ. The second dependency rule tells MMS that TESTS.OBJ is itself a target and that it depends on TEST1.BAS and TEST2.BAS.

In summary, MMS dependency rules allow you to describe all of the file relationships and processing needed to build a software system. However, for building complex systems (such as a compiler), the number of dependency rules may make the description file quite large. In such cases, instead of explicitly describing all dependencies, you may want to shorten the description file by relying on MMS built-in rules, which are described in the next section.

2.2 Built-in Rules

When writing a description file, you can explicitly state dependencies and actions, or you can abbreviate them by taking advantage of MMS *built-in rules.*

Built-in rules allow MMS to assume dependencies that are not stated in the description file and to perform actions necessary to update the target. MMS applies built-in rules (when they exist) in addition to any dependencies and action lines you supply in the description file. By using built-in rules, you can write shorter description files and allow MMS to do most of the work for you. A complete list of the MMS built-in rules is in Table A-3 in Appendix A.

MMS attempts to use its built-in rules only when you omit the action line or the source, or both, 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 make this relationship explicit, as in the following dependency rule:

```
MOD3.OBJ : MOD3.FOR
     FORTRAN MOD3.FOR
```

Or you can rely on MMS built-in rules by eliminating the action line, leaving only:

```
MOD3.OBJ : MOD3.FOR
```

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

If you omit the action line, MMS tries to locate the source in the directory you specified. If the source exists there, MMS uses the built-in rule that describes how it should use that source to update the target. If the source does not exist, MMS issues an error message and aborts execution.

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. In the previous example, since the target's file name is MOD3, MMS assumes that the source's file name is also MOD3. And since MMS knows that .OBJ files depend on .FOR files with the same file name, you can abbreviate the previous dependency rule even further to:

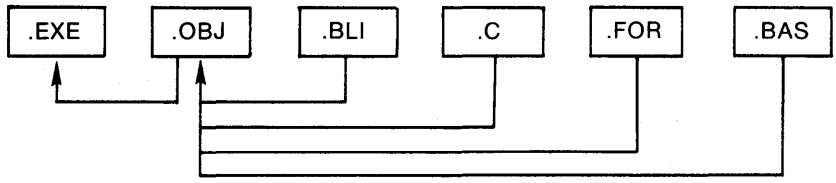
```
MOD3.OBJ :
```

MMS automatically looks for MOD3.FOR and uses it to build MOD3.OBJ.

To determine the source's file type, MMS checks its *suffixes precedence list*, which lists all the file types it recognizes, arranged in a predetermined order. MMS then uses the built-in rules (and user-defined rules, if any exist) to determine how the various types of files can be generated from the known rules. (User-defined rules are explained in Section 3.1.) Suppose the suffixes precedence list contains the following file types:

```
.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 could be represented as follows:



ZK-1664-84

The arrows in this figure indicate rules known to MMS. For example, a known rule specifies how an .EXE file is made from an .OBJ file; no other rule exists (in this figure) for making .EXE files. Similarly, rules exist to direct MMS how to make an .OBJ file from a .BLI file, a .C file, a .FOR file, and a .BAS file. Since .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 included in Table A-1; you can alter the order of the suffixes precedence list, as described in Section 3.4.4.)

Suppose that your description file directs MMS to update a target called MOD3.EXE. If you specify no sources or action lines, MMS assumes that the source and the target have the same file name, which in this example is MOD3. Since a rule exists to build an .EXE file from an .OBJ file, MMS looks in your directory (or in the directory you specified in the description file) for a file named MOD3.OBJ. If such a file exists, and if it has a revision time later than the target's, MMS applies the known rule to update the target.

If MMS cannot locate a source with the proper file name and with the first file type in the suffixes list that can update the target, it proceeds to the next file type that is connected to the target type by a known rule. In this example, however, no other rules exist for making .EXE files. If MMS checks all possible source types and cannot find a file with the same name as the target, it does not abandon the attempt to update the target. Instead, it tries to build a file that could then be used as the target's source. For example, if MMS failed to find

MOD3.OBJ, it tries to build MOD3.OBJ and then use it as the source that updates MOD3.EXE. To build this new target, MMS works through the list of file types, looking in the specified directory for a source with the correct file name and a file type that corresponds in turn to each file type that can update the target.

For example, the figure shows that .OBJ files can be built from .BLI, .C, .FOR, and .BAS files; so, 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. Thus, 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.

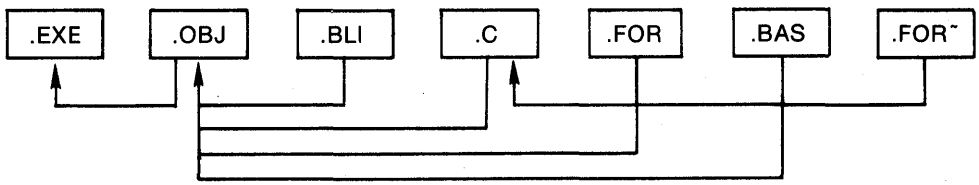
Suppose that MMS finally matches MOD3.OBJ with MOD3.FOR and locates MOD3.FOR in your directory. It then applies the built-in rule that instructs it to update the target MOD3.OBJ from the source MOD3.FOR by using the action line FORTRAN MOD3.FOR. This procedure explains why a dependency rule as brief as the following:

```
MOD3.OBJ :
```

equates to the full dependency rule:

```
MOD3.OBJ : 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 non-existent sources. For example, if MMS does not locate a file that can update MOD3.OBJ, it starts with the first possible source for MOD3.OBJ and tries to build that. In this case, it would try to build MOD3.BLI, since the suffixes list gives .BLI files the highest priority among the file types that update .OBJ files. Since there is no known rule for building .BLI files (in our example), MMS next considers .C. Suppose you have added a rule that directs MMS to build a .FOR file by fetching it from a CMS library. (Section 4.2 explains how to specify CMS elements in description files.) The relationship between the rules and the file types might look like this:



ZK-1665-84

(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. Suppose that 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. Thus the simple dependency

```
MOD3.EXE :
```

could result in the following sequence of actions:

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

MOD3.OBJ : MOD3.FOR
     FORTRAN/OBJ=MOD3 MOD3.FOR

MOD3.FOR : MOD3.FOR~
     CMS FETCH MOD3.FOR~
```

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 what seems to be the correct source for updating a target, it does not immediately apply the corresponding known rule. Instead, it checks to see whether the source itself needs updating before it can be used to update the original target. To do this, MMS repeats the process of trying to find 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 that does not take advantage of MMS built-in rules:

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

MOD1.OBJ : MOD1.C
     CC MOD1.C

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

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

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

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

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

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

The first dependency rule lists the object files and states that PROG.EXE is constructed by executing the DCL command LINK. 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. The third dependency rule says that MOD2.OBJ also depends on DEFDIR:DEFS1.H. The rule for building MOD1.OBJ need not be specified because a built-in rule directs MMS to build it from MOD1.C.

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 3.1 describes when and how to define new rules.

2.3 Macro Definitions

A *macro* is a name that represents a character string. You define macros at the beginning of your description file or on the command line that invokes MMS. You can then use the macro names in the description file in place of the strings. For example, you might discover that your description file uses the same file name over and over, or that you have several action lines that invoke a compiler with the same set of qualifiers. If you define a macro to represent the file name or the list of qualifiers, you can use the macro name throughout your description file. Then, if you want to use a different file name or need to change the qualifiers, you can edit only the macro definition and leave the rest of the description file as it is.

The following sections describe how to define and use MMS macros. Besides your own macros, you can also use default macros, discussed in Section 2.3.4, and special macros, which MMS defines for you, described in Section 3.2. Finally, you can also treat CLI symbols as macros (see Section 2.3.5).

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

1. Command-line definitions
2. Description file definitions

3. Built-in definitions
4. CLI symbol definitions

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.

You can change the order in which MMS applies macro definitions by using the /OVERRIDE qualifier (see Chapter 5).

2.3.1 Format

A macro definition has the following format:

```
name = string
```

name

The name of the macro. A macro name can consist of any characters except a space, a tab, a carriage return, an equal sign, the sequence $( ), and control characters. A macro name can be as long as you like.

string

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; however, when the macro is expanded it is considered as one line.

You must begin a macro definition in column 1 of the line. You can place macro definitions anywhere in the description file; however, it is a good idea to place all macro definitions at the beginning of the description file so that you can easily find and edit them.

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 2.3).

2.3.2 Invoking Macros

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

```
$(name)
```

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

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:

```
TWO = /DEBUG

ONE = /LIST $(TWO)
```

The macro invocation $(TWO) is expanded to /DEBUG because TWO has already been defined. If the positions of the macro definitions were reversed, TWO 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. That is, 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:

```
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
```

2.3.3 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"... }

filespec

A VAX/VMS file specification or a logical name for a file that contains only macro definitions. The default file type is .MMS.

"macro"

A macro definition enclosed in quotation marks. Use the same format that you would use to define a macro in a description file; that is, **name = string.** If you need to specify more than one macro, you must separate the macros with commas and enclose the list in parentheses.

The /MACRO qualifier is described in detail in Chapter 5.

Suppose you want to build a new program, called TEST1.EXE, using the same description file with which you built TESTS.EXE (as shown in the example in Section 2.3.2). To redefine FNAME and override the macro definition in the description file, type the following:

```
$ MMS/DESC=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/DESC=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 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 Chapter 5 for descriptions of these qualifiers.)

2.3.4 Default Macros

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. Table A-2 in Appendix A contains the MMS default macros.

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 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 following very simple dependency rule:

```
PROG.OBJ :
```

You may, though, want to redefine a default macro, perhaps so that you can use different qualifiers. The following example redefines CFLAGS:

```
CFLAGS = /LIST

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

MOD1.OBJ :

MOD2.OBJ : DEFDIR:DEFS1.H

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

MMS interprets the description file as the following:

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

MOD1.OBJ : MOD1.C
     CC/LIST MOD1.C

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

MOD3.OBJ : 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 by typing:

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

MMS then interprets the description file as the following:

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

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

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

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

2.3.5 Using CLI Symbols as MMS Macros

If you have defined CLI symbols before you invoke MMS, you can use them as MMS macros in your description file or on the MMS command line. For example, you could define a CLI symbol called CCQUALS as follows:

```
$ CCQUALS :== /LIST/DEBUG
```

Then in your description file you could use the syntax for a macro invocation to refer to this symbol:

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

When MMS processes the description file, it replaces the macro with the list of qualifiers specified in the CLI symbol definition.

To find the macro definition, 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.

By default, the CLI symbol table is the last place MMS looks for macro definitions. To give CLI symbols a higher precedence, you can use the /OVERRIDE qualifier, as described in Chapter 5.

If an action line in your description file changes the value of a CLI symbol, this change is not reflected in any subsequent use of that symbol as a macro in the description file. MMS expands the symbol in the parent process that is running the MMS image, while the action lines are executed in a subprocess. Because the CLI symbol tables for the process and the subprocess are not related, the action lines are not affected by the changed symbol.

2.4 Description File Example

Figure 2-1 illustrates a system that consists of an executable file called PROG.EXE, several object and source files, and two definition files. Files at a higher level depend upon files at a lower level.

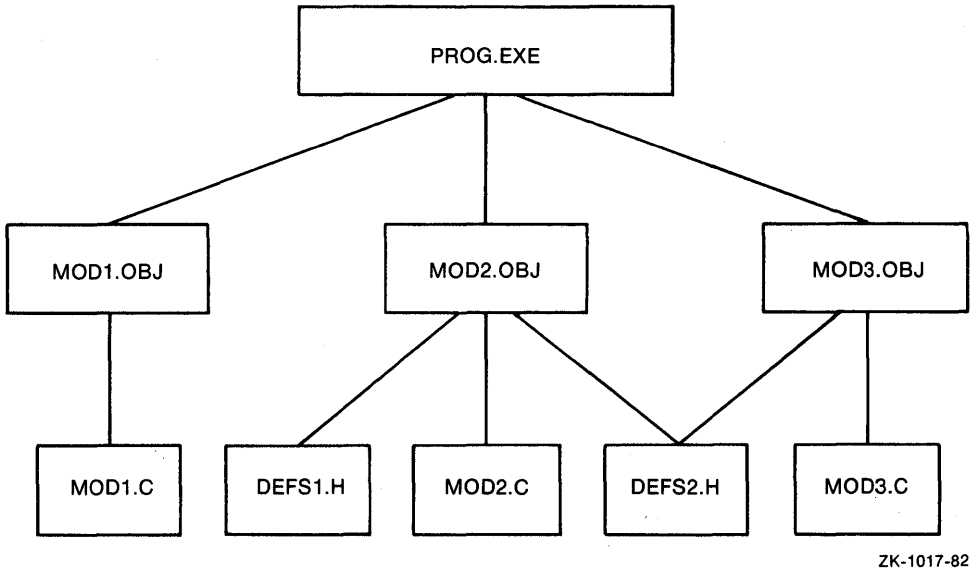


Figure 2-1: Dependencies of Files in a Software System

PROG.EXE is made by linking together three files: MOD1.OBJ, MOD2.OBJ, and MOD3.OBJ. All of the object files are made by compiling their respective .C source files: MOD1.C, MOD2.C, and MOD3.C. Some of these sources include definition files; for example, MOD2.C refers to both DEFS1.H and DEFS2.H, and MOD3.C refers to DEFS2.H.

PROG.EXE depends upon the object, source, and definition files. If any of them have changed since PROG.EXE was last linked, PROG.EXE is no longer up-to-date. Similarly, the object files depend upon their respective sources, as well as upon the definition files in two cases.

Note that although MOD2.C and MOD3.C refer to the definition files, they do not depend upon them. Therefore, if either of the two definition files was changed, MOD2.OBJ would be have to be updated, but no change would be needed to MOD2.C. If only DEFS2.H were changed, both MOD2.OBJ and MOD3.OBJ would need to be updated.

The following example shows how to state the dependencies shown in Figure 2-1 in a description file, CPROG.MMS.

```
PROG.EXE : MOD1.OBJ, MOD2.OBJ, MOD3.OBJ     ! PROG depends on three .OBJ files
     LINK/EXEC=PROG MOD1.OBJ, -              ! LINK object files
          MOD2.OBJ, -                        ! to produce PROG.EXE
          MOD3.OBJ

MOD1.OBJ :   ! Use built-in rules to compile MOD1.C and produce MOD1.OBJ

MOD2.OBJ : DEFDIR:DEFS1.H  ! Use built-in rules to compile MOD2.C

MOD2.OBJ, MOD3.OBJ : DEFDIR:DEFS2.H   ! Use built-in rules to compile MOD3.C
```

Suppose that you want to update the system and the current directory contains the following files:

```
$ DIR/DATE=MOD

Directory USER$:[LOUISE]

CPROG.MMS              22-JAN-1984 09:42
DEFS1.H                22-JAN-1984 10:20
DEFS2.H                22-JAN-1984 10:00
MOD1.C                 22-JAN-1984 10:25
MOD1.OBJ               22-JAN-1984 10:30
MOD2.C                 22-JAN-1984 10:05
MOD2.OBJ               22-JAN-1984 10:10
MOD3.C                 22-JAN-1984 10:15
```

Note that the file MOD3.OBJ does not exist in the directory. To build the system, type:

```
$ MMS/DESC=CPROG
```

MMS builds the system and writes the following action lines to SYS$OUTPUT as they are executed:

```
CC /NOLIST/OBJECT=MOD2 MOD2.C
CC /NOLIST/OBJECT=MOD3 MOD3.C
LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
```

The description file in this example illustrates the MMS build process:

1. Since no target is specified on the command line, MMS attempts to update the first target in the description file (PROG.EXE).
2. MMS finds that all the sources for PROG.EXE are themselves targets in subsequent dependency rules.
3. MMS scans each subsequent dependency rule and, by comparing the revision dates and times, determines whether the target should be updated:
   - If any of the sources have changed since the target was last updated, MMS executes the action line to update the target. For example, in the third dependency rule, MMS determines that the definition file DEFS1.H is newer than the target MOD2.OBJ. Therefore, MMS updates MOD2.OBJ. In the fourth dependency rule, MMS determines that MOD3.C has recently been revised, but that the target MOD3.OBJ does not exist in the current directory. Therefore, MMS creates the target, MOD3.OBJ.
   - If none of the sources for a specific dependency have changed since the target was last updated, MMS does not execute the action line, and the target is left unchanged. For example, in the second dependency rule, the target MOD1.OBJ is more recent than its source, so MMS does not update MOD1.OBJ.
4. After MMS updates all the sources, it updates the main target, PROG.EXE, by executing its action line.

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:

- User-defined rules
- Special macros
- Double colon dependencies
- Directives
- Action line prefixes
- Invoking MMS from a description file

3.1 User-defined Rules

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

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

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

.SRC

The file type of the source.

.TAR

The file type of the target.

comment

A string of text, introduced by an exclamation point (!), that documents the description file.

action line

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.

You can continue any line in a user-defined rule onto the next line by ending it with a hyphen (-). A comment can appear on any line in the user-defined rule.

Suppose a new language has been added to your system. The command for compiling programs written in this language is NEWLANG; the files have a type of .NEW. If you want MMS to build .OBJ targets automatically from .NEW sources, you must first add .NEW to the suffixes precedence list. (Section 3.4.4 describes how to extend the suffixes precedence list.)

Then, you can put the following user-defined rule in your description file:

```
.NEW.OBJ
     NEWLANG $(MMS$SOURCE)
```

MMS$SOURCE is a special macro that MMS expands to the name of the source (see Section 3.2). MMS interprets this rule in the following way: make an .OBJ target from a .NEW source by executing the NEWLANG command.

3.2 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 may not use a special macro on a target/source line in a description file. You may not redefine a special macro.

Table 3-1 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 3-1: 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	$%	Expands to the name of a module in a VAX/VMS library and its file name, including the file type (see Section 4.1).
MMS$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).

More information on the special macros that relate to DEC/CMS can be found in Section 4.2.

NOTE

The characters $*, $%, and $? always denote special macros. If an action line contains these characters 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. Suppose your description file contains the following dependency:

```
[ALDEN]MOD2.OBJ : [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
```

A good use of the MMS$CHANGED_LIST special macro is to get listings of files that have changed since the last time the system was built. For example:

```
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 insures that PRINT.FLG has approximately the same revision time as PROG.EXE. Therefore, 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. Suppose your description file contains the following dependency rule:

```
PROG.EXE : 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)
```

Assume that your directory contains the following entries:

```
$ DIR/DATE=MODIFIED

Directory USER$:[MICHAELS]

MOD1.OBJ                02-DEC-1983 13:50
MOD2.OBJ                02-DEC-1983 09:22
MOD3.OBJ                02-DEC-1983 14:06
PROG.EXE                02-DEC-1983 11:47
$
```

Since 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.

The MMS$TARGET_NAME special macro expands to the target name or the file name (without the file type) of the target being updated. MMS$TARGET_NAME can be useful when the rule specifies an output file with the same name as the source being processed, but with a file type other than the one MMS expects by default. For example, you could create your own rule for building .MEM files from .TXT sources, as shown here:

```
.TXT.MEM :
     RUNOFF/OUTPUT=$(MMS$TARGET_NAME).MEM $(MMS$SOURCE)
```

This rule directs MMS to use DIGITAL Standard Runoff (DSR) to build .MEM files from any .TXT files.

You can then use this rule in a description file to build a document:

```
.SUFFIXES .TXT

.TXT.MEM :
     RUNOFF/LOG/OUTPUT=$(MMS$TARGET_NAME).MEM $(MMS$SOURCE)

MEMO.MEM : PART1.MEM, PART2.MEM
     COPY/LOG PART1.MEM MEMO.MEM
     APPEND/LOG PART2.MEM MEMO.MEM

PART1.MEM :

PART2.MEM :
```

The first line in this description file adds .TXT to the list of suffixes recognized by MMS. (Section 3.4.4 describes the .SUFFIXES directive in detail.) Since you have defined the rule that MMS will use to update .MEM files from .TXT files, you can omit the sources and the action lines from the last two dependency rules.

If both .MEM files in this example need to be updated, the following output appears on your screen when you run MMS:

```
$  MMS MEMO.MEM
RUNOFF/LOG/OUTPUT=PART1.MEM PART1.TXT
DIGITAL Standard Runoff Version V2.0-014: No errors detected
6 pages written to "USER$:[HENRY]PART1.MEM;1"
RUNOFF/LOG/OUTPUT=PART2.MEM PART2.TXT
DIGITAL Standard Runoff Version V2.0-014: No errors detected
19 pages written to "USER$:[HENRY]PART2.MEM;2"
COPY/LOG PART1.MEM MEMO.MEM
%COPY-S-COPIED, USER$:[HENRY]PART1.MEM;1 copied to USER$:[HENRY]MEMO.MEM;1 (24 b
locks)
APPEND/LOG PART2.MEM MEMO.MEM
%APPEND-S-APPENDED, USER$:[HENRY]PART2.MEM;2 appended to USER$:[HENRY]MEMO.MEM;1
 (1252 records)
$
```

3.3 Double Colon Dependencies

In writing MMS dependency rules, you are allowed to 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.

Sometimes, however, you want MMS to take different actions depending on which sources have changed. For example, in the previous dependency rules, MMS was supposed to execute the PRINT command if DEFS1.DEF had changed and the PASCAL command if MOD2.PAS had changed. For such cases,

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. The double colon 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 could 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 MOD2.PAS is newer than MOD2.OBJ, both action lines will be executed.

Suppose you use MMS to maintain a library of object files. This library, called UTIL.LIB, 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. The syntax for using MMS to access files in VAX/VMS libraries is described in Section 4.1.

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

3.4 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 3-2 lists the directives and their functions.

Table 3-2: 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 SYS$OUTPUT or 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.

The following sections describe the directives in detail.

3.4.1 .IGNORE

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 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
```

Suppose that CHAPTER3.RNO contains DSR errors. When you run MMS with this description file (BOOK.MMS), the following lines 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$:[MICHAELS]CHAPTER3.RNO;1"
%RUNOFF-W-CJL, Can't justify line
        on output page 2; on input line 52 of page 1 of file "USER$:[MICHAELS]CHAPTER3.RNO;1"
%RUNOFF-W-TFE, Too few end commands
        on output page 3; on input line 77 of page 1 of file "USER$:[MICHAELS]CHAPTER3.RNO;1"
%RUNOFF-W-BMS, Bad margin specification: ".lm70
        on output page 4; on input line 102 of page 1 of file "USER$:[MICHAELS]CHAPTER3.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$:[MICHAELS]CHAPTER3.RNO;1"
DIGITAL Standard Runoff Version 2.0-014: 5 diagnostic messages reported
10 pages written to "USER$:[MICHAELS]CHAPTER3.MEM;1"
RUNOFF CHAPTER4
DIGITAL Standard Runoff Version V2.0-014: No errors detected
13 pages written to "USER$:[MICHAELS]CHAPTER4.MEM;1"
COPY/LOG CHAPTER1.RNO BOOK.MEM
%COPY-S-COPIED, DOCD$:[MICHAELS]CHAPTER1.MEM;1 copied to DOCD$:[MICHAELS]BOOK.MEM;1 (35 blocks)
APPEND/LOG CHAPTER2.MEM BOOK.MEM
%APPEND-S-APPENDED, DOCD$:[MICHAELS]CHAPTER2.MEM;1 appended to DOCD$:[MICHAELS]BOOK.MEM;1 (1452 records
APPEND/LOG CHAPTER3.MEM BOOK.MEM
%APPEND-S-APPENDED, DOCD$:[MICHAELS]CHAPTER3.MEM;1 appended to DOCD$:[MICHAELS]BOOK.MEM;1 (1508 records
APPEND/LOG CHAPTER4.MEM BOOK.MEM
%APPEND-S-APPENDED, DOCD$:[MICHAELS]CHAPTER4.MEM;1 appended to DOCD$:[MICHAELS]BOOK.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 but still contain errors of which you will not be aware.

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 Chapter 5 for more information on the /IGNORE qualifier.)

3.4.2 .SILENT

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 /NOACTION). /OUTPUT and /NOACTION are described in Chapter 5).

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 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. (See Chapter 5 for more information on the /VERIFY qualifier.)

3.4.3 .DEFAULT

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...
```

action line

A command-language command that MMS will execute by default. You may specify as many action lines as you like.

The .DEFAULT directive can be useful when, for example, you are developing a system that contains inoperative parts. You want MMS to process the operating portions and inform you about the inoperative parts. Assume that only one module (TEST.D) has been written in the system described by the following description file:

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

```
$ MMS/DESC=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 .DEFAULT in this way, you are reminded when you invoke MMS of the modules you have not yet implemented.

Another situation in which .DEFAULT might be useful is to copy files from one directory to another, as shown in this 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. Since these dependency rules have no action lines and MMS finds no built-in or user-defined rules to 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 3.2.)

Unlike some directives, .DEFAULT cannot be changed or overridden from the MMS command line.

3.4.4 .SUFFIXES

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 to the existing list, or disable recognition of all file types. MMS uses the suffixes precedence list to determine the order in which it should look 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 2.2 contains a detailed discussion of how the suffixes precedence list and MMS built-in rules work together.

The .SUFFIXES directive has the following format:

.SUFFIXES [file types list]

file types list

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.

When you want to change the precedence of suffixes, you must first clear the precedence list with the simple directive .SUFFIXES. Then you can specify .SUFFIXES followed by a list of file types to enable built-in and user-defined rules for the specified suffixes. Once you set up a new list of suffixes, MMS will recognize only the specified file types.

When you specify a new list, you must be careful to list the file types of targets before the file types of sources. For example, consider the new suffixes list here:

```
.SUFFIXES

.SUFFIXES .MEM .OBJ .RNO .BLI
```

When MMS tries to apply a built-in rule to update an .OBJ target from a .BLI source, it will begin searching the precedence list with the first suffix following that of the current target from which an .OBJ file can be built. In this case, MMS begins searching at .BLI and finds that it can use a built-in rule to update the .OBJ file. However, in the next example, the positions of the two suffixes are reversed:

```
.SUFFIXES

.SUFFIXES .BLI .RNO .OBJ .MEM
```

Because MMS begins its search with .MEM, it will not be able to locate the built-in rule that updates an .OBJ target from a .BLI source. Therefore, it will never find .BLI in the list because .BLI comes before .OBJ.

Suppose you want to redefine the suffixes precedence list so that .PLI files have precedence over .C files. You can do so by putting the following lines in your description file:

```
.SUFFIXES

.SUFFIXES .EXE .OBJ .PLI .C
```

The first use of .SUFFIXES completely clears the suffixes precedence list and disables all built-in rules. The next use specifies the revised order of file type precedence (.PLI before .C) and enables built-in rules for only the specified suffixes.

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
```

With the .SUFFIXES directive, you can also add a new file type to the suffixes precedence list. If you want the new file type to be added at the end of the precedence list, you can simply specify it after .SUFFIXES, as in the following:

```
.SUFFIXES .NEW
```

In this case, however, you must be sure not to clear the suffixes list first because you still want the rest of the suffixes to be recognized.

If you want to give precedence to .NEW over some or all of the established file types, you must first clear the suffixes precedence list and then redefine the precedence in the order you want:

```
.SUFFIXES

.SUFFIXES .EXE .OLB .OBJ .NEW .BLI
```

The first use of .SUFFIXES completely clears the suffixes precedence list and disables all built-in rules. The second use redefines the suffixes precedence list so that .NEW files have precedence over .BLI files. It also enables the built-in rules for the specified suffixes only.

After you have added a new file type to the suffixes precedence list, you can then create a user-defined rule to instruct MMS how it can update a target from a source with the new file type. If you want MMS to use the command NEWLANG to update .OBJ targets from .NEW sources, you can put the following lines in your description file:

```
.SUFFIXES

.SUFFIXES .EXE .OLB .OBJ .NEW .BLI

.NEW.OBJ
     NEWLANG $(MMS$SOURCE)
```

When MMS processes the description file, it expands the special macro MMS$SOURCE to the name of the source and updates the corresponding .OBJ target with the NEWLANG command. Section 3.1 explains how to define your own rules to supplement MMS built-in rules.

3.4.5 .INCLUDE

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

filespec

A VAX/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. For example, suppose you create a description file of user-defined rules, MAKEINDEX.MMS, that generate an index using an indexing program called INDEX:

```
.SUFFIXES

.SUFFIXES .MEX .RNX .BRN .RNO

INDEXQUALS = /INDEX/NOOUTPUT

.RNX.MEX
     RUNOFF/LOG $(MMS$SOURCE)                ! Format the index

.BRN.RNX
     INDEX $(MMS$SOURCE)    ! Process intermediate file with INDEX

.RNO.BRN
     RUNOFF $(INDEXQUALS) $(MMS$SOURCE)     ! Make an intermediate file
```

Subsequently, a description file (CHAPTER1.MMS) that wants to take advantage of these rules can simply include MAKEINDEX.MMS as follows:

```
.INCLUDE MAKEINDEX

CHAPTER1.MEX : CHAPTER1.RNX     ! Index formatted for output

CHAPTER1.RNX : CHAPTER1.BRN     ! Intermediate file makes an index

CHAPTER1.BRN : CHAPTER1.RNO     ! Input file updates intermediate file
```

When you invoke MMS to run this example, the following output appears on your screen (or in your output log file):

```
$ MMS/DESC=CHAPTER1
RUNOFF /INDEX/NOOUTPUT CHAPTER1.RNO     ! Make an intermediate file
INDEX CHAPTER1.BRN     ! Process intermediate file with INDEX
RUNOFF/LOG CHAPTER1.RNX                 ! Format the index
RUNOFF Version V2.0-014: No errors detected
3 pages written to "USER$:[AUSTEN]CHAPTER1.MEX;1"
$
```

Included files may themselves include files, up to a depth of sixteen or the maximum open file limit for your current process (as indicated by the FILLIM quota). 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.

3.4.6 .FIRST

The .FIRST directive tells MMS to execute certain action lines before it executes the action lines that update the target.

The .FIRST directive has the following format:

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

action line

A command-language command that MMS will execute 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
```

3.4.7 .LAST

The .LAST directive tells MMS to execute certain action lines after it has executed the action lines that update the target.

The .LAST directive has the following format:

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

action line

A command-language command that MMS will execute after it updates the target. 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. The following output might be produced on your screen when you use the description file shown here:

```
$ MMS/DESC=ADESC
FORTRAN/LIST=[GREGORY.LISTINGS]A.LIS          /OBJECT=[GREGORY.OBJECTS]A.FC
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)
```

3.4.8 .IFDEF, .ELSE, and .ENDIF

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

The name of the macro being tested.

description file line

Zero or more action lines that are valid in a description file.

.IFDEF 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.

For example, suppose your description file contains the following lines:

```
.IFDEF VAX

A.OBJ : A.BLI
     BLISS A

.ENDIF

.IFDEF PDP11

A.OBJ : A.BLI
     BLISS/PDP11 A

.ENDIF
```

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

```
$ MMS/DESC=BLIPROG/MACRO="VAX=CURRENT"
BLISS A
$
```

Since the command line defines the macro VAX, the command BLISS 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 will skip 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.BLI
     BLISS A
.ENDIF
A.EXE : A.OBJ
     LINK A.OBJ

.ELSE
A.OBJ : A.BLI
     BLISS/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.

3.5 Action Line Prefixes

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

The two action line prefixes are described in Table 3-3.

Table 3-3: MMS Action Line Prefixes

Prefix	Function
- (Ignore)	Causes MMS to ignore errors generated by the action line on which the prefix appears.
@ (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 non-blank 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; they must be separated from the rest of the action line with at least one space or tab. The following example shows the use of both prefixes:

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

Note that 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.

3.5.1 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
```

3.5.2 The Silent Prefix (@)

The Silent action line prefix (@) directs MMS that the action line on which the prefix appears should not be written to SYS$OUTPUT or to the file specified by the /OUTPUT qualifier. This prefix 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 then returns to the [WORKING] directory. Since the Silent prefix suppresses the action lines, MMS can do its work silently and then display "Cleanup done" when the work is done.

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

MMS assumes that an at sign (@) followed by a space signifies the Silent prefix. If you want to invoke a command procedure from an action line, you must not type a space between the at sign and the name of the command procedure.

3.6 Invoking MMS from a Description File

You can invoke MMS from a description file while MMS is updating a target. The second invocation of MMS will run as a spawned subprocess that inherits any existing symbol definitions. To invoke MMS from within a description file, specify the reserved macro $(MMS) on an action line where you want MMS to be invoked again.

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 Chapter 5.) Thus, the MMS subprocess is created but no other actions are performed.

NOTE

When you spawn an MMS subprocess, you may exceed the quotas assigned to your process. See Section 1.5 for a discussion of how MMS execution can be affected by subprocess quotas.

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 Chapter 5.

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/DESC=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/DESC=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.

Chapter 4

Accessing Libraries With DEC/MMS

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

- VAX/VMS libraries created with the LIBRARY utility
- DEC/CMS (Code Management System) libraries
- VAX FMS (Forms Management System) libraries
- VAX CDD (Common Data Dictionary)

4.1 Creating and Accessing Files in VAX/VMS Libraries

MMS can access files that are contained in VAX/VMS libraries; in addition, you can use MMS to create library files using certain built-in rules. The built-in rules that MMS uses to create library files are listed in Table A-4 in Appendix A. These rules tell MMS to create the specified library if one does not already exist, and then 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.

To specify that a source or target in a dependency rule is a module in a VAX/VMS library, use the following format avoiding any spaces or tabs that cause problems in processing:

```
library (module[=filespec],...)
```

library

A VAX/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.

module

The name of the module in the library.

filespec

A VAX/VMS file specification that corresponds to the module in the library. The default file type depends on the file type of the library. You can use a logical name for the name of the module, but if you do so, you must 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$STRLEN = STRLEN.OBJ) designates the module C$STRLEN in library CRTLIB.OLB; C$STRLEN is found in the file named STRLEN.OBJ.

To specify more than one module in the same library, you have three options:

- 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$STRIND = STRIND.OBJ).
- You can use the VMS * wildcard character. For example, the specification CRTLIB(C$*) directs MMS to look for all modules in the library CRTLIB.OLB whose names begin with the characters C$.
- You can use the VMS % wildcard character. For example, the specification CRTLIB(C$STR%) directs MMS to look for all modules in CRTLIB.OLB whose names begin with the characters C$STR followed by only one character.

You can use a complete library specification when you need to access library modules whose names are not valid VAX/VMS file specifications (for example, C$STRLEN). However, you do not always need to make the specification complete; 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, though, you must supply it. 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
```

If you use a logical name as the directory name of an object library, that logical name must not translate to another logical name. This restriction is necessary because the built-in rules that MMS uses to update libraries are defined in terms of the DCL function F$SEARCH, which translates only one level of logical name.

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).

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.

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$LIBRARY:CRTLIB/LIB, USER$:[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 DEC/CMS

If VAX DEC/CMS (Code Management System) is installed on your system, you can use MMS to access elements in CMS libraries. You should be familiar with CMS before you read this section.

MMS provides default macros and special macros tailored for use with CMS. Table A-2 in Appendix A lists the default macros, and Table 3-3 in Chapter 3 describes the special macros.

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/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 A-5 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, suppose your description file 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 Chapter 5.

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/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~
```

Note, however, that you cannot access elements in a CMS library that resides on a different DECnet node than your own.

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 3.4.5.)

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.

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.

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 A-5 in Appendix A.)

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 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.

4.2.4 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.

MMS looks for the description file on the main line of descent unless you override the default macro CMSFLAGS. If you specify /MACRO = "CMSFLAGS = /GEN = 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.

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 will automatically fetch 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 Accessing Forms in an FMS Library

If VAX FMS (Forms Management System) 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, you use the same syntax as for files in VAX/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, in 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 Chapter 5 for a description of /REVISE_DATE.)

4.4 Accessing Records in the CDD

If the VAX CDD (Common Data Dictionary) 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 a 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. /AUDIT 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.

The /AUDIT qualifier must follow the caret character in the CDD record specification. You separate the qualifier from the remark with an equal sign. You cannot use /AUDIT on the MMS command line.

MMS also provides the default macro CDDFLAGS. This macro is initially defined to be the null string, but you can redefine it 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 Chapter 5 for a description of /REVISE_DATE.)
- The /NOACTION qualifier does not affect 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.

Chapter 5

The MMS Command

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 VAX/VMS file specification 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. The following command line:

```
$ MMS
```

activates the following default qualifiers:

/ACTION
/NOCHECK_STATUS
/NOCMS
/DESCRIPTION = DESCRIP.MMS or /DESCRIPTION = MAKEFILE.
/NOIGNORE
/NOLOG
/OUTPUT = SYS$OUTPUT
/NOOVERRIDE
/NOREVISE_DATE
/RULES
/NOSKIP_INTERMEDIATE
/VERIFY

You can abbreviate all MMS qualifiers and their parameters. However, you must be sure that the abbreviations are unique, so that 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 rest of this chapter describes the MMS qualifiers in alphabetic order and 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.

/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.

Format

$ MMS/[NO]ACTION

Description

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 /OUTPUT).

/NOACTION overrides the Silent action line prefix (@) (described in Section 3.5.2).

NOTE

The $(MMS) reserved macro is executed even if you specify /NOACTION. Thus, you can see what actions MMS would have executed in the subprocess. See Section 3.6 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 4.4.

/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.

Format

$ MMS/[NO]CHECK_STATUS

Description

/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 DEC/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.

Format

$ MMS/[NO]CMS

Description

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 3.4.5.)

The /CMS qualifier also directs MMS to apply CMS built-in rules where appropriate. (See Table A-5 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 tildes (~) 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.

Format

$ MMS/DESCRIPTION = filespec...
$ MMS/NODESCRIPTION target

Parameters

filespec

A VAX/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.

target

A VAX/VMS file specification or a mnemonic name that designates the target to be built.

Description

To specify more than one description file, you can separate the file specifications with either commas (,) or plus signs (+). 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 and must not be surrounded by parentheses:

```
$ 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. That is, in the following command

```
$ MMS/DESC=(A,B) X,Y,Z
```

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. That is, in the following command

```
$ MMS/DESC="A + B" X,Y,Z
```

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 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 build the target specified on the command line. In that case, MMS does not automatically look for DESCRIP.MMS and MAKEFILE. If you use the /NODESCRIPTION qualifier, 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.

Format

$ MMS/FROM_SOURCES

Description

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.

Format

$ MMS/HELP[= "topic"]

Command Parameters

topic

An MMS topic on which you want information.

Description

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, not the execution of action lines.

Format

$ MMS/IDENTIFICATION

Description

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 level(s) 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.

Format

```
$ MMS/IGNORE = {[WARNING] ¦ ERROR ¦ FATAL}
$ MMS/NOIGNORE
```

Parameters

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.

ERROR

Directs MMS to ignore both "W" and "E" errors, but to abort execution when it finds an "F" error.

FATAL

Directs MMS to ignore all errors, and to continue processing the description file. This parameter is equivalent to the .IGNORE directive.

Description

/IGNORE is equivalent to /IGNORE = WARNING.

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 3.4.1 and 3.5.1 describe the .IGNORE directive and the Ignore action line prefix in detail.

The /NOIGNORE qualifier directs MMS to abort execution when it finds any error.

If you want to override the .IGNORE directive contained in a description file, you must type the /IGNORE[= WARNING], /IGNORE = ERROR, or /NOIGNORE 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, not the execution of action lines.

Format

$ MMS/[NO]LOG

Description

The /LOG qualifier directs MMS to write all informational messages to your terminal screen while it processes the description file. 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. The /LOG qualifier is useful for debugging your description files.

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, not the execution of action lines.

Format

$ MMS/MACRO = { filespec ¦ "macro",... }

Parameters

filespec

A VAX/VMS file specification or a logical name that identifies a file of macro definitions. The default file type is .MMS.

"macro"

A macro definition enclosed in quotation marks. Use the same format as for macro definitions in description files, that is, **name = string**.

Description

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 2.3 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 to SYS$ERROR. This qualifier affects the behavior of MMS, not the execution of action lines.

Format

$ MMS/OUTPUT = filespec

Parameters

filespec

A VAX/VMS file specification or a logical name that identifies the output file. The default file type is .LOG.

Description

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, not the execution of action lines.

Format

$ MMS/[NO]OVERRIDE

Description

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.

/OVERRIDE imposes the following order of application when MMS processes macro definitions:

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

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.

/NOOVERRIDE imposes the following order, which is the default hierarchy:

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

/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.

Format

$ MMS/[NO]REVISE_DATE

Description

The /REVISE_DATE qualifier directs MMS to change only the revision dates of any target that needs updating, but not 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 3.4.2 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$STATUS (see Section 2.1.3 for information about MMS$STATUS).

/REVISE_DATE 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, not the execution of action lines.

Format

$ MMS/[NO]RULES[= filespec]

Command Parameters

filespec

A VAX/VMS file specification or a logical name that identifies the file of default rules that MMS is to use.

Description

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.

/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.

Format

$ MMS/[NO]SKIP_INTERMEDIATE

Description

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 therefore 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.

Restrictions

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 VAX/VMS librarian) may operate safely only on the sources used to update the current target.

MMS dependencies cannot distinguish between situations in which all sources must be present for MMS to perform the specified action and situations 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.

Format

$ MMS/[NO]VERIFY

Description

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 3.4.2 and 3.5.2, 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.

Chapter 6

DEC/MMS Examples

This chapter contains many examples that should help you use MMS creatively and effectively. It assumes that you understand MMS concepts and that you are familiar with DCL. This chapter also suggests approaches to some specific tasks for which you might not have considered using MMS. You can use these approaches as they appear in this chapter, or you can modify them for other purposes.

6.1 Simple Uses of MMS

The following sections describe some simple tasks for which you can use MMS. Some of these tasks do not even require a description file.

6.1.1 Checking Whether Files Are Up-to-Date

You can use MMS to check whether a file is up-to-date with respect to its source(s), without even using a description file. Suppose you have defined a CMS library that contains the source file TEST.BLI. The target, TEST.EXE, resides in your default directory. To see whether TEST.EXE is up-to-date, you need type only the following command line:

```
$ MMS/CMS/SKIP/CHECK TEST.EXE
```

MMS checks the time of TEST.BLI in the CMS library and reports whether TEST.EXE is up-to-date with respect to TEST.BLI. (The /SKIP qualifier ensures that the outcome of /CHECK is unaffected if any intermediate files do not exist.)

6.1.2 Using MMS to "Fetch and Build"

Suppose you have a CMS library that contains DSR (DIGITAL Standard Runoff) sources (which have a file type of .RNO) and you want to create a .MEM file from one of those sources. Instead of fetching the .RNO file from the CMS library and running DSR to create the .MEM file, you can simply use the /CMS qualifier and specify the .MEM file on the command line.

For example, suppose you want to create a memo, and the source MEMO.RNO exists in your CMS library. To use MMS to build MEMO.MEM, you can type the following command line:

```
$ MMS/CMS MEMO.MEM
```

MMS fetches MEMO.RNO from the CMS library, runs DSR, and creates MEMO.MEM. No description file is necessary, but if your directory does contain one of the default description files (DESCRIP.MMS or MAKEFILE.) and the description file specifies the same .MEM file as a target, MMS will process that description file.

6.1.3 Using the /SKIP_INTERMEDIATE Qualifier

You can use the /SKIP_INTERMEDIATE qualifier when you want to update a system, but do not want to rebuild all the intermediate files (such as .OBJ files). For example, suppose your directory contains the following files:

```
$ DIRECTORY/COL=1

Directory USER$:[MARK]

A.BLI
DESCRIP.MMS
MYPROG.EXE
MYPROG.OLB
```

The description file for building MYPROG.EXE could look like this:

```
MYPROG.EXE : MYPROG.OLB(A,B,C)
     LINK/EXE=MYPROG MYPROG/LIB/INCLUDE=A
```

Suppose that you have fetched A.BLI from a CMS library and edited it. To update the system to reflect the changes to A.BLI, type this command:

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

MMS updates MYPROG.EXE and also creates A.OBJ, since A.BLI is newer than the corresponding module in MYPROG.OLB. However, MMS does not create any other .OBJ files (such as B.OBJ and C.OBJ) because you specified /SKIP on the command line. Had you not specified /SKIP, MMS would have fetched B.BLI and C.BLI, compiled them, and added the .OBJ files to MYPROG.OLB, even though they did not need to be updated.

You may want to delete intermediate files once the main target is updated. See Section 6.7 for an example of selectively deleting files.

6.2 Description File Example

The description file in the next example shows how MMS can be used to perform the following functions:

- Define macros
- Describe dependencies
- Print out the source 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 this description file explain the dependency rules.

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

MOD2.OBJ : DEFS1.H

MOD2.OBJ, MOD3.OBJ : 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.

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

```
! 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 $(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:

```
$ MMS CLEANUP
```

To print all the source files, type:

```
$ MMS PRINT
```

To print all recently changed source files, type:

```
$ MMS PRINTNEW
```

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

```
$ MMS CROSSREF
```

To check the portability of source code, type:

```
$ MMS PORTABLE
```

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

6.3 Gathering Statistics

You can use MMS to gather statistics about your files. The examples in the following sections describe some methods for gathering certain kinds of statistics.

6.3.1 Finding Out What Sources Are Missing

Suppose that you have stored the sources for a particular software system in a source directory or CMS library, and you want to make sure that all the sources you need are there. To get a list of any missing files, you could put a default action such as the following in your description file, using the .DEFAULT directive:

```
.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.

6.3.2 Creating a Checkpoint File

You can use MMS to create a checkpoint file that indicates when MMS finished building a target. For example, suppose that your directory contains the source files TEST1.C, TEST2.C, and TEST3.C. You want MMS to create .EXE files from each of these sources and also 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. It also creates a file called CHECK.PNT, which 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 : FOO.PNT BAR.PNT BAS.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. Also, the .PNT files never really exist. They are simply a trick to 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-1984 14:48
To:      MICHAELS
Subj: Build summary of MAIN_TARGET ending at 21-FEB-1984 14:48:06.85

Completed build of TEST1.EXE at 21-FEB-1984 14:47:32.99
Completed build of TEST2.EXE at 21-FEB-1984 14:47:49.65
Completed build of TEST3.EXE at 21-FEB-1984 14:48:06.33
```

6.4 Using DCL Command Procedures in Description Files

You can use DCL command procedures in conjunction with MMS. For example, you could write a command procedure that loops until a given file becomes available, and invoke that command procedure in your description file.

Suppose your command procedure is called GETFILE.COM, and contains the following lines:

```
$ LABEL:
$ IF "''F$SEARCH(" 'P1' ")'" .NES. "" THEN GOTO DONE
$ WAIT +:'P2'
$ GOTO LABEL
$ DONE:
```

You could use this command procedure when you start MMS in a batch job.

The description file that invokes GETFILE.COM might look like the following:

```
GET_NEXT_INFO :
     MAIL NL: $(MY_PROC)/SUBJECT="string"
     @GETFILE ANSWER.IN 15
     @ANSWER.IN
```

NOTE

Be sure that 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.

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.

6.5 Creating and Using Time Stamps

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

The following sections describe two methods of creating and using time stamps.

6.5.1 Creating a Time Stamp File Using DCL Symbols

The following example 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 may 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.

6.5.2 Creating a Time Stamp File Using Included Files

Suppose you have the following directories and files:

```
[DIR1] contains FILE1.X

[DIR2] contains FILE2.Y

[DIR3] contains FILE3.Z
```

You want MMS to build a file reporting 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)
```

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

1. If no changes were made to the files, MMS prints "No changes detected," as instructed in the REPORT_CHANGE action line.
2. 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 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"
```

6.6 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: $(MY_PROC) -
     /SUBJECT="$(MMS$SOURCE) is back in the CMS library."
     SET DEFAULT 1234567890 ! Causes MMS to abort with $STATUS = failure
```

The command procedure that loops until the specified element is available in the CMS library looks like the following:

```
$ 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 will wait the specified interval before trying again. If the element has been replaced, the first bit in $STATUS is 0, and MMS will mail 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 @FILENAME
```

FILENAME is the name of the command procedure.

6.7 Selectively Deleting Files

Suppose you have just updated your system, and now you want to delete the intermediate files from your working directory. Or you want intermediate files to be deleted automatically after an MMS build. Three ways of accomplishing this task with MMS are:

1. Create a command procedure.
2. Use a macro definition.
3. Use the .LAST directive.

The first two methods are described in the following sections. The .LAST directive is described in Section 3.4.7.

6.7.1 Creating a Command Procedure

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 just be extremely careful that you are deleting only those files you are sure 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 DELETE
```

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

6.7.2 Using a Macro Definition

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

1. Use a macro definition on the MMS command line.
2. 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 for 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
```

6.8 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.

Appendix A

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 A-1 contains the suffixes precedence list.
- Table A-2 lists the default macros.
- Table A-3 contains the standard built-in rules.
- Table A-4 describes the built-in rules for accessing VAX/VMS libraries.
- Table A-5 includes the built-in rules for accessing CMS library elements.

For information on using MMS to create and access elements in VAX/VMS libraries, see Section 4.1; in CMS libraries, Section 4.2.

Table A-1: 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~ .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~

NOTE

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.

Table A-2: MMS Default Macros

Macro	Definition
ANLFLAGS	/OUTPUT = $(MMS$TARGET_NAME)
AS	MACRO
BASIC	BASIC
BASFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
BLISS	BLISS
BLISS16	BLISS/PDP11
BFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
BLIBFLAGS	/NOLIST
CC	CC
CFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
CDDFLAGS	null string
CLDFLAGS	null string
CMS	CMS
CMSCOMMENT	null string
CMSFLAGS	/GEN = $(MMS$CMS_GEN)
COBOL	COBOL
COBFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
CORAL	CORAL
CORFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
DIBOL	DIBOL
DBLFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
FMS	FMS
FMSFLAGS	/REPLACE
FORT	FORTRAN
FFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
LIBR	LIBRARY
LIBRFLAGS	/REPLACE
LINK	LINK
LINKFLAGS	/TRACE/NOMAP/EXEC = $(MMS$TARGET_NAME)
MACRO	MACRO
MFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
MSGFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
PASCAL	PASCAL
PENVFLAGS	/NOLIST
PFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
PLI	PLI
PLIFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
RPG	RPG
RPGFLAGS	/NOLIST/OBJECT = $(MMS$TARGET_NAME)
RUNOFF	RUNOFF
RFLAGS	/OUTPUT = $(MMS$TARGET_NAME)

Table A-3: MMS Built-in Rules

Source	Target	Action
.BAS	.OBJ	$(BASIC) $(BASFLAGS) $(MMS$SOURCE)
.BLI	.OBJ	$(BLISS) $(BFLAGS) $(MMS$SOURCE)
.B16	.OBJ	$(BLISS16) $(BFLAGS) $(MMS$SOURCE)
.B32	.OBJ	$(BLISS) $(BFLAGS) $(MMS$SOURCE)
.C	.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	.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	.OBJ	$(PASCAL) $(PFLAGS) $(MMS$SOURCE)
.PAS	.PEN	$(PASCAL) /ENVIRON = $(MMS$TARGET) $(PENVFLAGS) $(MMS$SOURCE)
.PLI	.OBJ	$(PLI) $(PLIFLAGS) $(MMS$SOURCE)
.RNH	.HLP	$(RUNOFF) $(RFLAGS) $(MMS$SOURCE)
.RNO	.MEM	$(RUNOFF) $(RFLAGS) $(MMS$SOURCE)
.REQ	.L32	$(BLISS) /LIBRARY = $(MMS$TARGET) $(BLIBFLAGS) $(MMS$SOURCE)
.RPG	.OBJ	$(RPG) $(RPGFLAGS) $(MMS$SOURCE)
.R16	.L16	$(BLISS16) /LIBRARY = $(MMS$TARGET) $(BFLAGS) $(MMS$SOURCE)
.R32	.L32	$(BLISS) /LIBRARY = $(MMS$TARGET) $(BFLAGS) $(MMS$SOURCE)

Table A-4: Built-in Rules for Library Files

To build a help library:

```
.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)
```

To build an object library:

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

To build a text library:

```
.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)")'" .EQS. "" -
        THEN $(FMS)/LIBRARY $(FMSFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
     IF "''F$SEARCH("$(MMS$TARGET)")'" .EQS. "" -
        THEN $(FMS)/LIBRARY/CREATE $(MMS$TARGET) $(MMS$SOURCE)
```

Table A-5: Built-in Rules for CMS Access

.ANL~.ANL	.HLP~.HLP
.BAS~.BAS	.MAC~.MAC
.BLI~.BLI	.MAR~.MAR
.B16~.B16	.MMS~.MMS
.B32~.B32	.MSG~.MSG
.C~.C	.OPT~.OPT
.CLD~.CLD	.PAS~.PAS
.COB~.COB	.PLI~.PLI
.COM~.COM	.REQ~.REQ
.COR~.COR	.RNH~.RNH
.DAT~.DAT	.RNO~.RNO
.DBL~.DBL	.RPG~.RPG
.DDL~.DDL	.R16~.R16
.FOR~.FOR	.R32~.R32
.FRM~.FRM	.TXT~.TXT
.H~.H	

Each rule in the table performs the following action to reestablish the default CMS library:

```
MMS$CMSLIB :== 'f$logical("CMS$LIB")'
IF " '' MMS$CMSLIB'" .nes. "$(MMS$TARGET)" -
     THEN $(CMS) SET LIBRARY $(MMS$TARGET)
$(CMS) FETCH $(MMS$SOURCE) $(CMSFLAGS) $(CMSCOMMENT)
IF MMS$CMSLIB .EQS. "" THEN $(CMS) SET LIBRARY 1234
IF MMS$CMSLIB .NES. "" .AND. MMS$CMSLIB .NES. "$(MMS$TARGET)" -
     THEN $(CMS) SET LIBRARY 'MMS$CMSLIB'
```

Because the action is the same for each built-in rule, it is not repeated after each target/source line. Therefore, this action applies to all of the dependencies listed in Table A-5.

A tilde (~) after a file type indicates that the file is in a CMS library.

Appendix B

DEC/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 VAX/VMS and UNIX are very different operating systems, certain system-imposed changes were necessary to provide the features of *make* on VAX/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/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(s) as part of a VAX/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/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.
- 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. However, 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 than *make*. See Table A-3 in Appendix A 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 VAX/VMS library module, you can use the ? wildcard character as a synonym for the % wildcard.

For compatibility with *make*, MMS provides alternative formats for dependency rules, user-defined rules, and directives, and recognizes two-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 3-3 in Chapter 3.

Appendix C

DEC/MMS Messages

This appendix lists the DEC/MMS messages. These messages are accompanied by explanations and, where possible, suggestions for actions needed to recover from errors.

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.

C.1 Message Format

The general format of messages displayed by the VAX/VMS operating system is the following:

```
%FACILITY-L-IDENT, text
```

FACILITY

The name of a VAX/VMS facility or component (in this case, MMS).

L

A severity level indicator. It has one of the following values:

Code	Meaning
S	Success
I	Information
W	Warning
E	Error
F	Fatal error

IDENT

An abbreviation of the message text. The message descriptions in this appendix are alphabetized by this abbreviation.

text

The actual text of the message.

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] qualifier 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 *Installing VAX DEC/MMS* for information on submitting SPRs.

C.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 *DEC/CMS Reference Manual* 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 *DEC/CMS Reference Manual* 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 support.

Explanation: You are trying to access a source in a CMS Version 2 library, but MMS was installed without CMS Version 2 support.

User Action: DEC/CMS Version 2 must already be installed on your system before you install MMS if you want access to CMS Version 2 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. See Section 1.5 for a discussion of 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. See Section 1.5 for a discussion of subprocess quotas and MMS. 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.

GMPRIVIO, MMS does not have read access to file 'filespec.'

Explanation: MMS is unable to read the specified file, possibly because the file protection does not allow read access.

User Action: This is an informational message. MMS assigns the file the oldest date known to VAX/VMS (17-NOV-1858 00:00:00.00) and continues processing the description file. You may want to check the file protection and change it so that the read access is allowed.

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 target(s) 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.

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.

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.

LEXINACCINCL, Included file 'filespec' is inaccessible.

Explanation: The file could not be found, could not be opened, or was already open.

User Action: Verify that the file exists. If it does, check its location, access mode, protection, and file-sharing characteristics.

LEXINACCRULE, Rules file 'filespec' is inaccessible.

Explanation: The specified rules file could not be found, could not be opened, or was already open.

User Action: Verify that the file exists. If it does, check its location, access mode, protection, and file-sharing characteristics.

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.

LFSUNEXP, Unexpected error from RMS while looking for 'filespec.'

Explanation: MMS encountered an error while doing a $SEARCH RMS call.

User Action: Collect as much information as possible and submit a Software Performance Report (SPR).

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 will apply.

User Action: If you want to redefine a macro, you must use the /MACRO qualifier on the MMS command line.

NODEPFIL, Cannot find default description file (either DESCRIP.MMS or MAKEFILE.).

Explanation: Because you did not specify a description file on the command line with the /DESCRIPTION qualifier and you did not specify /NODESCRIPTION, MMS looked first for DESCRIP.MMS and then for MAKEFILE. However, neither of these files exists in the working directory.

User Action: If the description file exists, reissue the command using the /DESCRIPTION qualifier. If a description file does not exist, create one.

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.

NOSPECDEP, Cannot find specified description file 'filespec.'

Explanation: You specified either an illegal file specification or a nonexistent file as the description file.

User Action: Invoke MMS again with the correct file specification.

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.

User Action: Either remove some of your symbols or have the system manager change the SYSGEN parameter CLISYMTBL.

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, collect as much information as possible and 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).

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(s) 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(s) 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.

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 VAX/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-defines 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 VAX/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

Action line prefixes, 3-19
  see also individual prefix
  Ignore (-), 3-20
  Silent (@), 3-20
Action lines, 2-3
  definition of, 2-2
  effect of /ACTION on, 5-3
  format of, 2-4
  restrictions on, 2-8
/ACTION qualifier, 5-3
/AUDIT qualifier, 4-9

B

Built-in rules, 2-10
  example of, 2-13
  for CMS libraries, A-5
  for libraries, A-4

C

CDD record
  syntax of, 4-9
CDD records
  access to, 4-8
CDDFLAGS default macro, 4-9
/CHECK_STATUS qualifier, 5-4
CLI symbols
  used as macros, 2-19
CMS commands
  in description files, 4-4
CMS elements
  automatic access of, 4-5
  explicit references to, 4-6
  including with .INCLUDE, 4-6
CMS libraries
  access to, 4-4
  built-in rules for, A-5
/CMS qualifier, 4-5, 4-7, 5-5
CMSCOMMENT default macro, 4-7
CMSFLAGS default macro, 4-5, 4-7
Command procedures
  advantages of MMS over, 1-6 to 1-7
  generating with /OUTPUT, 5-15

D

.DEFAULT directive, 3-11
Default macros, 2-18
  CDDFLAGS, 4-9
  CMSCOMMENT, 4-7
  CMSFLAGS, 4-5, 4-7
  FMSFLAGS, 4-8
  redefining, 2-18
  table of, A-2
Dependency rules, 2-2
  alternative format for, B-2
  comments in, 2-3
  continuing, 2-4
  double colon in, 3-7
  examples of, 2-9
  format of, 2-3
DESCRIP.MMS, 1-4
Description files, 2-1
  CMS commands in, 4-4
  default, 1-4
  examples of, 2-9, 2-20, 6-1
  in CMS libraries, 4-7
/DESCRIPTION qualifier, 5-6
Differences, MMS and *make*, B-1
Directives, 3-7
  see also individual directives
  .DEFAULT, 3-11
  .ELSE, 3-18
  .ENDIF, 3-18
  .FIRST, 3-16
  .IFDEF, 3-18
  .IGNORE, 3-8
  .INCLUDE, 3-14
  .LAST, 3-17
  .SILENT, 3-10
  .SUFFIXES, 3-12
Double colon
  in dependency rules, 3-7

E

.ELSE directive, 3-18
.ENDIF directive, 3-18
Error messages, C-1

F

.FIRST directive, 3-16
FMS forms
  access to, 4-8
  syntax of, 4-8
FMSFLAGS default macro, 4-8
/FROM_SOURCES qualifier, 5-8

H

/HELP qualifier, 5-9

I

/IDENTIFICATION qualifier, 5-10
.IFDEF directive, 3-18
.IGNORE directive, 3-8
  overriding, 3-10
Ignore prefix (-), 3-20
/IGNORE qualifier, 5-11
.INCLUDE directive, 3-14
Including files, 3-14

L

.LAST directive, 3-17
Libraries
  built-in rules for, A-4
Libraries (Cont.)
  CMS, 4-4
   access to elements in, 4-5 to 4-6
  FMS, 4-8
   syntax of forms in, 4-8
  VMS, 4-1
   syntax of modules in, 4-1
/LOG qualifier, 5-13

M

/MACRO qualifier, 2-17, 5-14
Macros, 2-14
  CLI symbols as, 2-19
  default, 2-18
   table of, A-2
  defining, 2-14
  defining in a file, 2-17
  defining on the command line, 2-17, 5-14
  example of, 2-16
  format of, 2-15
  invoking, 2-15
  ($MMS), 3-21
  $(MMSQUALIFIERS), 3-22
  $(MMSTARGETS), 3-22
  redefining, 2-17
  redefining default, 2-18
  special, 3-2
    table of, 3-3
    used with libraries, 4-3
  user-defined, 2-14
make and MMS differences, B-1
MAKEFILE., 1-4
Messages, MMS, C-1
MM$TARGET_NAME macro
  example of, 3-5
MMS command
  abbreviating, 5-2
  format of, 5-1
  qualifiers, 5-2
   see also Qualifiers
MMS messages, C-1
MMS$CHANGED_LIST special macro
  example of, 3-4
MMS$LIB_ELEMENT special macro, 4-3
MMS$RULES, 5-18
MMS$SOURCE special macro
  example of, 3-4
  used with libraries, 4-3
MMS$STATUS, 5-4, 5-17
MMS$STATUS special symbol, 2-8
MMS$TARGET special macro
  example of, 3-4
  used with libraries, 4-3
MMS$TARGET_NAME special macro
  used with libraries, 4-3
$(MMS) reserved macro, 3-21
$(MMSQUALIFIERS) reserved macro, 3-22
$(MMSTARGETS) reserved macro, 3-22
Mnemonic names, 2-6

N

/NOACTION qualifier, 5-3
/NOCHECK_STATUS qualifier, 5-4
/NOCMS qualifier, 5-5
/NODESCRIPTION qualifier, 1-4, 5-6
/NOIGNORE qualifier, 5-11
/NOLOG qualifier, 5-13
/NOOVERRIDE qualifier, 5-16
/NOREVISE_DATE qualifier, 5-17
/NORULES qualifier, 5-18
/NOSKIP_INTERMEDIATE qualifier, 5-19
/NOVERIFY qualifier, 5-21

O

/OUTPUT qualifier, 5-15
/OVERRIDE qualifier, 5-16

P

Precedence list, 3-12
  table of, A-1

Q

Qualifiers
  abbreviating, 5-2
  /ACTION, 5-3
  /AUDIT, 4-9
  /CHECK_STATUS, 5-4
  /CMS, 4-5, 4-7, 5-5
  default, 5-1
  /DESCRIPTION, 5-6
  /FROM_SOURCES, 5-8
  /HELP, 5-9
  /IDENTIFICATION, 5-10
  /IGNORE, 5-11
  /LOG, 5-13
  /MACRO, 2-17, 5-14
  /NODESCRIPTION, 1-4
  /OUTPUT, 5-15
  /OVERRIDE, 5-16
  /REVISE_DATE, 5-17
  /RULES, 5-18
  /SKIP_INTERMEDIATE, 5-19
    example of, 6-2
  /VERIFY, 5-21
Quotas
  subprocess, 1-5

R

/REVISE_DATE qualifier, 5-17
Rules
  built-in
    see Built-in rules
  dependency
    see Dependency rules
  user-defined
    see User-defined rules
/RULES qualifier, 5-18

S

.SILENT directive, 3-10
  overriding, 3-10
Silent prefix (@), 3-20
/SKIP_INTERMEDIATE qualifier, 5-19
  example of, 6-2
Software development cycle, 1-1
Sources, 1-2, 2-3
  mnemonic names for, 2-6
  multiple, 2-5
Special macros, 3-2
  table of, 3-3
  used with libraries, 4-3
Subprocesses
  invoking MMS as, 3-21
  MMS's use of, 1-5
  quotas for, 1-5
.SUFFIXES directive, 3-12
Suffixes precedence list, 3-12
  table of, A-1

T

Targets, 1-2, 2-3
  mnemonic names for, 2-6
  multiple, 2-5

U

User-defined rules, 3-1
  alternative format for, B-2
  example of, 3-2
  format of, 3-1

V

/VERIFY qualifier, 5-21
VMS library access, 4-1

VAX DEC/MMS
User's Guide
AA-P119B-TE

READER'S COMMENTS

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

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

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

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

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

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

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

Do Not Tear - Fold Here

Cut Along Dotted Line