VMS Software

VSI DECset for OpenVMS Guide to the Module Management System

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

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

VSI DECset for OpenVMS Guide to the Module Management System

Copyright © 2025 VMS Software, Inc. (VSI), Boston, Massachusetts, USA

Legal Notice

Confidential computer software. Valid license from VSI required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for VSI products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. VSI shall not be liable for technical or editorial errors or omissions contained herein.

All other trademarks and registered trademarks mentioned in this document are the property of their respective holders.

Table of Contents

Preface

This guide explains how to use the Module Management System for OpenVMS (MMS). The guide provides both tutorial and reference material to show basic and advanced techniques.

MMS is patterned after the UNIX make utility. For details, see *Appendix B, "MMS and make Comparisons"*.

1. About VSI

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

2. Intended Audience

This guide is intended for experienced programmers and technical users who build systems using MMS.

3. Document Structure

This guide contains the following chapters and appendices:

- *Chapter 1, "Introduction to MMS"* describes the basic concepts of MMS and how MMS automates the software development cycle.
- *Chapter 2, "MMS Description Files"* describes how to create and use description files. It also discusses how to specify the input for the automatic description file generator.
- *Chapter 3, "Advanced Description File Techniques"* describes advanced techniques for using MMS as efficiently as possible.
- *Chapter 4, "Accessing Libraries and Other Objects"* explains how MMS processes files stored in OpenVMS, CMS, or FMS libraries. It also describes how MMS handles definitions stored in the Oracle CDD/Repository.
- *Chapter 5, "Command Dictionary"* describes the MMS command-line format and contains detailed descriptions of the MMS qualifiers.
- *Appendix A, "Error Messages"* lists and explains all the MMS messages.
- *Appendix B, "MMS and make Comparisons"* describes the differences between MMS and the UNIX make features.
- *Appendix C, "MMS Built-In Features"* contains tables of MMS defaults and includes explanatory information.

4. VSI Encourages Your Comments

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

5. OpenVMS Documentation

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

6. Typographical Conventions

The following conventions are used in this manual. In addition,please note that all IP addresses are fictitious.

Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 x	A sequence such as PF1 x indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
…	In examples, a horizontal ellipsis indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
()	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[]	In format descriptions, brackets indicate that whatever is enclosed within the brackets is optional; you can select none, one, or all of the choices. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
{}	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
red ink	Red ink indicates information that you must enter from the keyboard or a screen object that you must choose or click on. For online versions of the book, user input is shown in **bold**.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in online versions of the book.
italic text	Italic text represents information that can vary in system messages (for example, Internal error *number*).
UPPERCASE TEXT	Uppercase letters indicate that you must enter a command (for example, enter OPEN/READ), or they indicate the name of a

	routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.
-	Hyphens in coding examples indicate that additional arguments to the request are provided on the line that follows.
numbers	Unless otherwise noted, all numbers in the text are assumed to be decimal. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

Chapter 1. Introduction to MMS

This chapter describes the Module Management System for OpenVMS (MMS) and provides information on the following topics:

- An overview of MMS
- Invoking MMS
- Getting help
- MMS concepts
- Building the software system
- Rebuilding the software system

1.1. Overview

A software system can have many program files, object libraries, include files, compilers, and compilation and linking options. The more complex the system, the more difficult it is to reproduce the same program image for each build.

MMS automates and simplifies the building of software systems. It can build simple programs consisting of one or more source files, or complex programs consisting of many source files, message files, and documentation files.

With MMS, you can specify exactly how a software system is to be built and rebuilt. You do this by using a description file in which you describe the components of the system and the file dependencies used to build and rebuild the system. A file dependency occurs when one or more files are needed to build another file. For example, the existence of an executable file depends on an object file.

Each time you run MMS, it follows the description file you have created, reads the components and dependencies, and builds the same system.

MMS can also rebuild systems quickly when parts of the system change. MMS keeps track of source and include files. If any of the files in a software system change or are missing, MMS can determine which files are affected by the changed or missing files, then rebuild the affected portions of the system by using the sources for the changed or missing files. MMS does not rebuild portions of the system that have not changed, thus saving both processing time and storage space.

With MMS, you can also build and test modules locally before building or testing the modules in the source directory or library.

MMS is patterned after the UNIX make utility. For details, see *Appendix B, "MMS and make Comparisons"*.

1.2. Invoking MMS

You invoke MMS from the DCL command line as follows:

```
$ MMS
```

You can invoke the MMS DECwindows Motif interface from the MMS command-line, as follows:

```
$ MMS/INTERFACE=DECWINDOWS
```

MMS then attempts to process the default description file DESCRIP.MMS in your current directory. If DESCRIP.MMS does not exist, MMS attempts to process a description file called MAKEFILE. (with a period, but with no extension). If MAKEFILE. does not exist, MMS attempts to process a file called target-name.MMS. If all of these files exist, MMS uses only DESCRIP.MMS.

If none of these files exist, MMS uses built-in rules to build the target (see *Section 1.4.5, "Built-In Rules"*).

You can also invoke MMS with a specified description file, called SYSTEM1.MMS in this example:

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

MMS uses the .MMS file type when none is specified.

For information on creating a description file, see *Section 2.1, "Creating the Description File"*. For more information on invoking description files, see *Section 2.1.2, "Specifying the Target on the Command Line"* and *Section 2.1.3, "Using Mnemonic Names for Targets and Sources"*. For information on MMS syntax and qualifiers, see *Chapter 5, "Command Dictionary"*.

1.3. Getting Help

You can view online help from the DECwindows interface, as described in *Section 1.6.1, "Menu Bar"*. You can also get information about MMS at either the DCL ($) or MMS level. At the DCL level, the DCL command HELP MMS provides online help on MMS qualifiers and other topics. For example:

```
$ HELP MMS
```

To get help on a specific topic, such as the /MACRO qualifier, type the qualifier after the HELP MMS command. For example:

```
$ HELP MMS/MACRO
```

At the MMS level, the MMS qualifier /HELP gives general information about MMS and a list of topics, then returns you to the DCL level. For example:

```
$ MMS/HELP
```

To get help at the MMS level on a specific MMS topic, follow the /HELP qualifier with an equal sign and the topic name enclosed in quotation marks (" "). For example:

```
$ MMS/HELP="/MACRO"
```

1.4. MMS Concepts

This section explains basic MMS concepts.

1.4.1. Description Files

The **description file** contains definitions that describe to MMS all the components of a system build: the source files that make up the system, the compilers that will be used, the order in which to link the software modules, and the libraries to use when the modules are linked.

The description file also contains the components that are the definitions for the logical dependencies in the software system. For example, it can contain definitions for the include files used in source files, the source files that make up each object, the objects that make up each image, and the libraries used in a link. See *Section 1.4.6, "Dependencies"* for more information on how MMS uses dependencies.

Each time you run MMS, it follows the description file and builds the same system (you can use older description files to recreate previous versions of the system). After a system is built once, MMS uses the dependencies to rebuild the system.

For information on creating a description file, see *Chapter 2, "MMS Description Files"*.

1.4.2. Targets

A **target** is any file that must be built to complete the software system (a target can also be a mnemonic name; see *Section 2.1.3, "Using Mnemonic Names for Targets and Sources"* for more information).

You can think of a target as the goal of building the system. Targets are usually executable image files, but they can also be object files. For example, executable files are targets for object files, and object files are targets for source files.

1.4.3. Sources

Sources are used to create targets. For example, a file with programming code is a source for an object file, and an object file is a source for an executable file.

1.4.4. Action Lines

An action line is a DCL command that MMS uses to update the target. The commands in the action lines tell MMS how to build the target. Built-in rules (see *Section 1.4.5, "Built-In Rules"*) allow MMS to build only one target; action lines override this rule.

For example, the PASCAL command is the action line that uses X.PAS to update X.OBJ, and the LINK command is the action line that uses X.OBJ to update X.EXE.

Action lines follow target or source lines and specify how to use the source file to create the target. You must indent the action line and leave a blank line before the next dependency rule. For example:

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

In this example, the action line is LINK MAIN.OBJ.

1.4.5. Built-In Rules

MMS uses built-in rules to create targets with specific file types. Built-in rules are based on file extensions. A built-in rule is an action that builds a target based on the file extension from the target's source file. For example, executable targets are created by default with the .EXE file type; object file targets are created with the .OBJ file type.

MMS also uses built-in rules to determine which compiler to use. For example, MMS uses the Pascal compiler for files with the .PAS file type and the Fortran compiler for files with the .FOR file type.

A software system must follow built-in file-naming conventions for built-in rules to work.

1.4.6. Dependencies

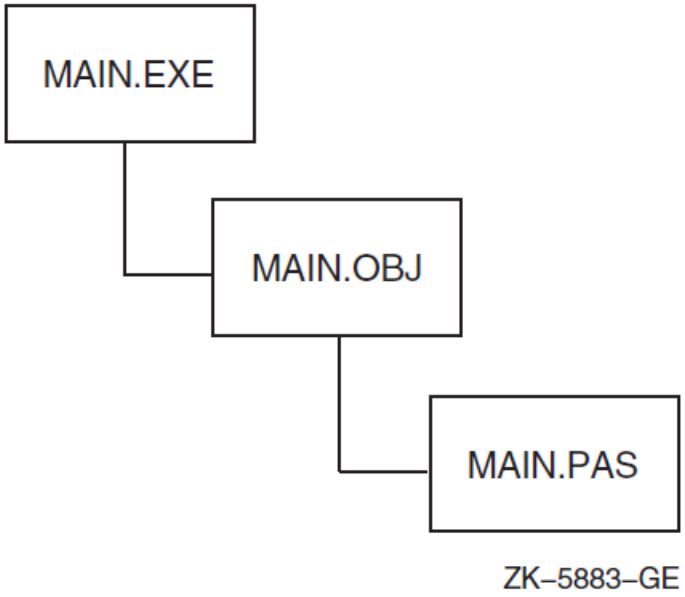
As you describe a system to MMS, you also state logical dependencies in that system. For example, by using the DEPENDS_ON keyword, a simple description file called SYSTEM1.MMS could contain the following dependencies:

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

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

Figure 1.1, "Dependencies in a Single-Source Syste" shows the relationship between the files.

Figure 1.1. Dependencies in a Single-Source Syste



MMS builds its own internal dependency tree. In general, MMS creates targets from the bottom up; that is, it searches for the source for the first target and then the source of the first target's source. It first builds the targets at the bottom of the dependency tree, then builds the targets that use those sources, then builds the targets that use those sources, and so on, until the primary target is built. MMS uses built-in rules (unless action lines are specified) to build the software system.

For example, MMS first creates an object file from a source code file, then creates the executable file from the object file.

1.5. MMS File Processing

As previously mentioned, MMS builds a target based on a description file and any related macro files and rules files. MMS follows three major steps to accomplish that goal:

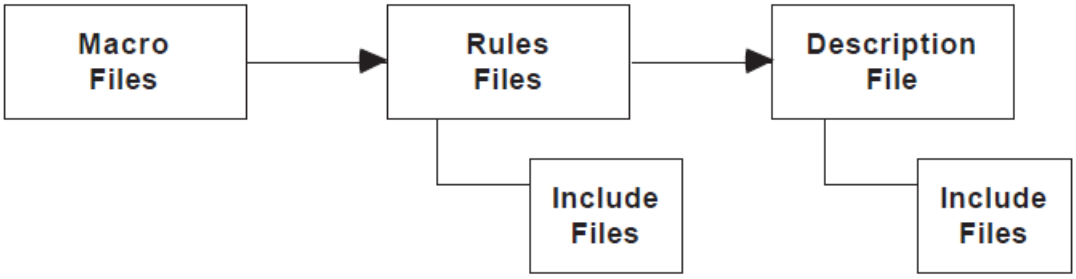
1. **Initialization**—MMS parses all the target's associated files and prepares its internal dependency tree in memory. This tree maps all sources and targets needed to build the main target.

2. **Verification**—MMS verifies the existence of the target's associated files, and compares the source revision dates to the target revision date. If any source is newer than the target, MMS notes the fact that the target should be built.

3. **File processing**—MMS processes all the dependencies from the bottom up, and builds the target.

As shown in *Figure 1.2, "Building a Target"*, MMS processes the macro and rules files before it processes the description file.

Figure 1.2. Building a Target



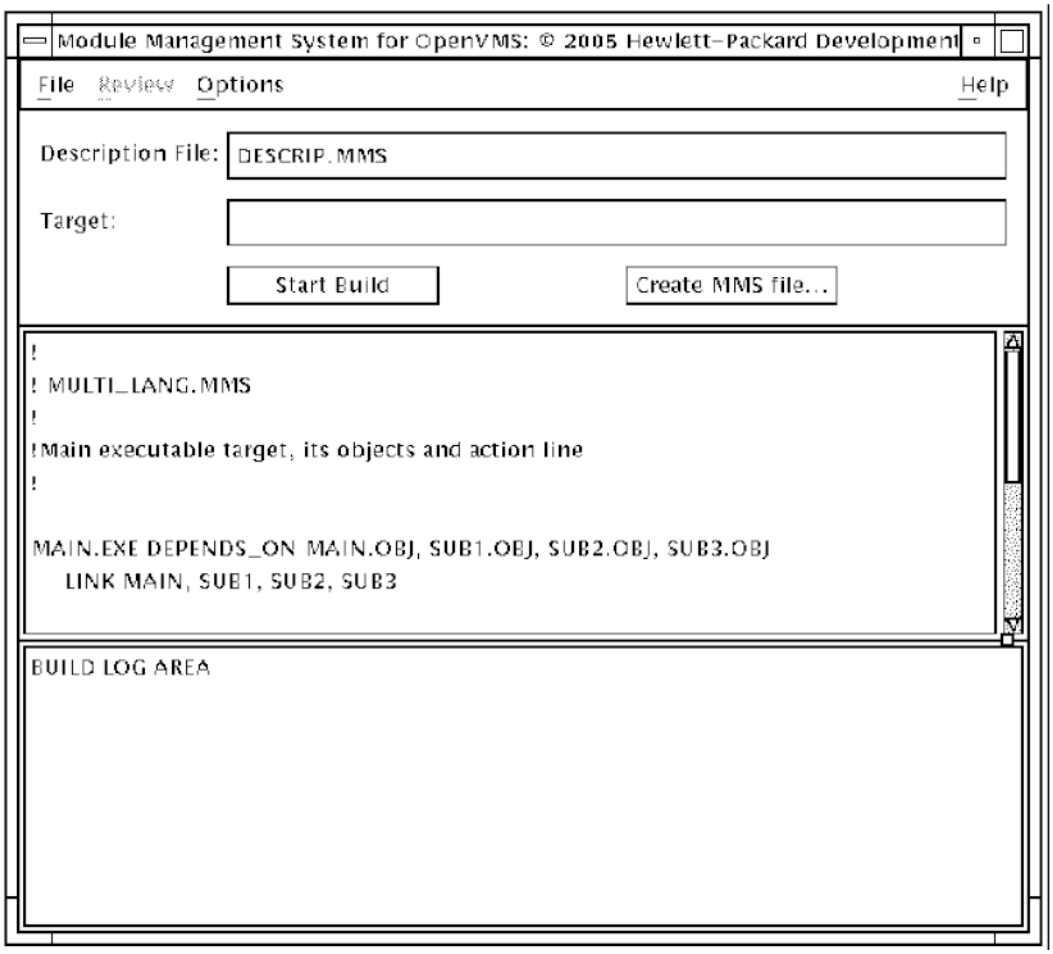
Note that a macro file can contain only macros, whereas a rules file or description file can contain any description-file element. However, VSI recommends that rules files should be limited to rules only. Note also that include files are processed at the location of the file where they are placed.

This processing order means that the proper placement of description-file elements is crucial, so that MMS behaves as you intended. For example, if you define a macro in the description file for use in a conditional in the rules file, MMS will not "see" the macro when processing the rules file. That is, the rules file is processed before the description file.

1.6. MMS User Interface

The MMS DECwindows interface consists of a main window, which is illustrated in *Figure 1.3, "MMS Main Window"*.

Figure 1.3. MMS Main Window



The MMS main window consists of the following working areas:

- Menu bar
- Build activation area
- Description file area and the Build log area

1.6.1. Menu Bar

The Menu bar contains some standard pull-down menus: File, Options, and Help, as well as the Review pull-down menu. These menus contain the following options:

- File menu—The items in this pull-down menu are used to control the contents of the MMS description file area. The menu items are as follows:
  - Open—Open an existing description file.
  - Fetch—Fetch the description file from the current CMS library.
  - Exit—Exit the MMS DECwindows interface.
- Review menu—The items in this pull-down menu are used to review the compilation diagnostics. The menu items are as follows:

- Next module—Bring up the diagnostics file in the LSE review buffer from the next module in which an error occurred.
- Previous module—Bring up the diagnostics file in the LSE review buffer from the previous module in which an error occurred.
- First module—Bring up the diagnostics file in the LSE review buffer from the first module in which an error occurred.
- Last module—Bring up the diagnostics file in the LSE review buffer from the last module in which an error occurred.

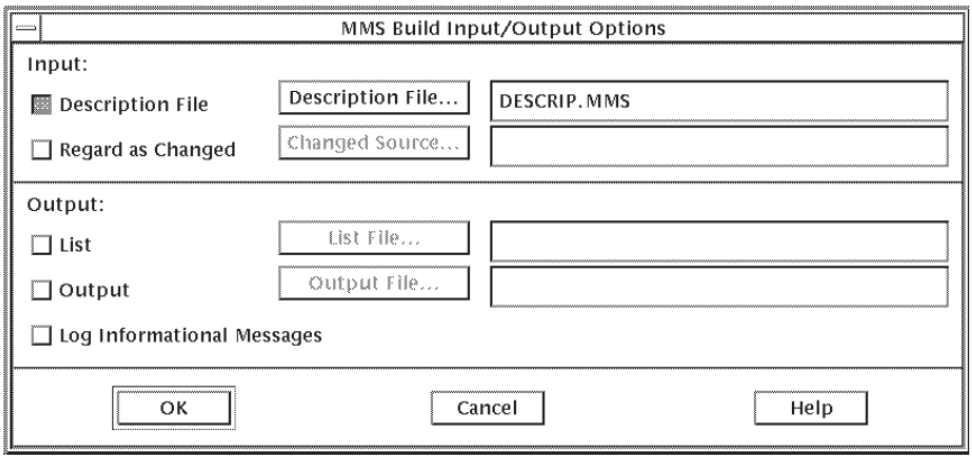
- Options menu—The items in this pull-down menu are used to modify the execution environment (see *Section 1.6.1.1, "MMS Options Menu"*). Also, a toggle switch is present in this pull-down to indicate whether the description file is displayed in the description file area.
- Help menu—This menu provides access to help screens. It contains the items that are usually present in DECset tool Help pull-down menus:
  - On Context—Bring up information about an object in a window or dialog box.
  - On Window—Display an overview.
  - On Help—Bring up information on how to use the help system.
  - On Version—Display copyright and version information.

1.6.1.1. MMS Options Menu

The Options pull-down menu items are used to modify the execution environment. *Figure 1.4, "MMS Input and Output Options Dialog Box"* illustrates the dialog box used to specify input and output. *Figure 1.5, "MMS Definitions and Directives Options Dialog Box"* illustrates the dialog box used to specify definitions and directives.

These dialog boxes give you a means to set options for the MMS command—they encompass the qualifiers that could go on the MMS command line. The qualifier defaults are represented as shown in *Figure 1.4, "MMS Input and Output Options Dialog Box"* and *Figure 1.5, "MMS Definitions and Directives Options Dialog Box"*.

Figure 1.4. MMS Input and Output Options Dialog Box



Input Section

Click on the buttons and use the text fields following the Input label to specify input files for any subsequent build. These buttons and fields are as follows:

- **Description File—**Click on this toggle button to specify an MMS description file for any subsequent build. Click on the Description File... push button and select a file from the dialog box, or enter a file specification in the text field. If the file-selection box is used, this field is filled in with the selected file.

- **Regard as Changed—**Click on this toggle button to direct MMS to treat only the specified sources as having been changed, regardless of their actual modification times. No date checking is performed at all; that is, MMS simply rebuilds any targets that depend on one or more of the specified sources.

Click on the Changed Source... push button and select a file from the dialog box, or enter a file specification in the text field. If more than one source is indicated, use a comma-separated list. If the file selection box is used, this field is filled in with the selected file.

Output Section

Click on the buttons and use the text fields following the Output: label to specify output files for any subsequent build. These buttons and fields are as follows:

- **List—**Click on this toggle button to specify an MMS list file for any subsequent build. Click on the List File... push button and select a file from the dialog box, or enter a file specification in the text field. If the file-selection box is used, this field is filled in with the selected file.

- **Output—**Click on this toggle button to specify an MMS output file for any subsequent build. Click on the Output File... push button and select a file from the dialog box, or enter a file specification in the text field. If the file selection box is used, this field is filled in with the selected file.

- **Log Informational Messages—**Click on this toggle button to instruct MMS to log informational messages, in addition to warning and error messages, to the List or Output file.

Navigation Buttons

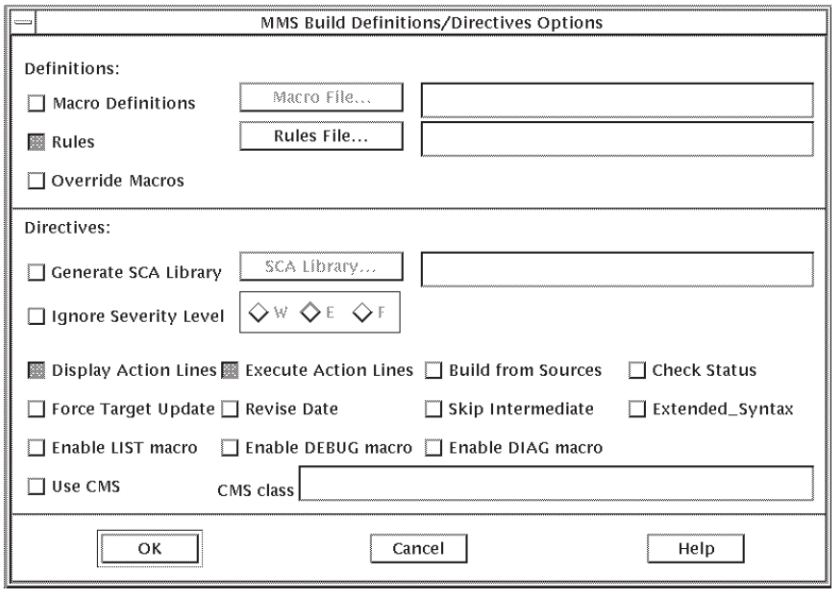
Click on the navigation buttons to save or cancel your selections, or get online Help. These buttons and fields are as follows:

- **OK—**Click on this button to indicate that the options for the MMS command are complete and you want to exit from the dialog box.

- **Cancel—**Click on this button to quit the dialog box without completing options and discarding changes for the MMS command.

- **Help—**Click on this button to bring up the textual explanation of the dialog box in a window.

Figure 1.5. MMS Definitions and Directives Options Dialog Box



Definitions Section

Click on the buttons and use the text fields following the Definitions label to specify definitions for any subsequent build. These buttons and fields are as follows:

- **Macro Definitions**—Click on this toggle button to direct MMS to add to or override the macro definitions in the description file. Click on the Macro File... push button and select a file from the dialog box, or enter a file specification in the text field. If the file selection box is used, this field is filled in with the selected file.

- **Rules**—Click on this toggle button to direct MMS to direct MMS to apply user-defined, built-in rules and a suffixes precedence list when it builds a system. Click on the Rules File... push button and select a file from the dialog box, or enter a file specification in the text field. If more than one source is indicated, use a comma-separated list. If the file selection box is used, this field is filled in with the selected file.

- **Override Macros**—Click on this toggle button to control the order in which MMS applies definitions when it processes macros.

Directives Section

The buttons and text fields following the Directives label are implemented as toggle buttons, although some require additional information. (For the toggle buttons not described here, see *Chapter 5, "Command Dictionary"* for the corresponding MMS command qualifiers.) If an additional file specification needs to be specified, a file-selection widget is used. The Directives section includes the following:

- **Generate SCA Library**—Click on this toggle button to direct MMS to generate an SCA library during the build process. Click on the SCA Library... push button and select a file from the dialog box, or enter a file specification into the text field. If the file selection box is used, this field is filled in with the selected file.

- **Ignore Severity Level**—Click on this toggle button to direct MMS to specify the severity levels of errors that MMS normally ignores when it executes action lines. Click on the buttons that correspond

to the DCL severity levels Warning, Error, and Fatal. These qualifiers affect the execution of action lines but not the behavior of MMS.

- **Display Action Lines**—Click on this toggle button to direct MMS to display action lines before executing them.
- **Execute Action Lines**—Click on this toggle button to direct MMS to execute the action lines in the description file.
- **Build from Sources**—Click on this toggle button to direct MMS to build a target from its sources, regardless of whether the target is already up-to-date.
- **Check Status**—Click on this toggle button to direct MMS to return 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.
- **Force Target Update**—Click on this toggle button to direct MMS to execute the action lines necessary to update one specific target.
- **Revise Date**—Click on this toggle button to direct MMS to change only the revision dates of all the targets that need updating, instead of performing the update.
- **Skip Intermediate**—Click on this toggle button to indicate that MMS should not build intermediate source/target files.
- **Extended Syntax**—Click on this toggle button to enable the extended syntax features of MMS. When this feature is selected, you can use the following in a description file:
  - Predefined functions
  - Macro redefinition
- **Enable LIST macro**—Click on this toggle button to direct MMS to use the macro definition "LIST=1." Automatically generated description files contain the following lines, unless the Use Built-in Rules option was specified:

  ```
  .IFDEF LIST
  LST = /LIST
  .ELSE
  LST = /NOLIST
  .ENDIF
  ```

  In addition, every compilation command includes the $(LST) macro.

  If you click on this toggle button, then every compilation in the build produces a listing file. If this toggle button is off, no listings are produced.
- Enable DEBUG macro—Click on this toggle button to direct MMS to use the macro definition "DEBUG=1." Automatically generated description files contain the following lines, unless the Use Built-in Rules option was specified:

  ```
  .IFDEF DEBUG
  DBG = /DEBUG
  DBGOPT = /NOOPTIMIZE/DEBUG
  .ELSE
  DBG = /NODEBUG
  DBGOPT = /OPTIMIZE/NODEBUG
  .ENDIF
  ```

In addition, every compilation command includes the $(DBG) or the $(DBGOPT) macro.

If you click on this toggle button, every compilation in the build is compiled with information to be included in the object module to use with the OpenVMS Debugger. In addition, the linker is directed to generate a debugger symbol table and give the debugger control when the image is run.

- **Enable DIAG macro—**Click on this toggle button to direct MMS to use the macro definition "DIAG=1." Automatically generated description files contain the following lines:

```
.IFDEF DIAG
DIA = /DIAGNOSTICS
.ELSE
DIA = /NODIAGNOSTICS
.ENDIF
```

In addition, every compilation command includes the $(DIA) macro.

If you click on this toggle button, every compilation in the build produces a diagnostics file. If diagnostics files are produced during a build and LSE is running, you can do a REVIEW of each compilation from MMS. If this toggle button is off, no diagnostics files are produced.

- **Use CMS—**Click on this toggle button to direct MMS to use the macro definition "CMSFLAGS=GENERATION=class-name." When you specify a CMS class, MMS still uses the CMS elements as the sources but it uses the designated class of generations, not necessarily the current generations.

Navigation Buttons

Click on the navigation buttons to save your selections, cancel your selections, or get online Help. The navigation buttons are as follows:

- **OK—**Click on this button to indicate that the options for the MMS command are complete and you want to exit from the dialog box.
- **Cancel—**Click on this button to quit the dialog box without completing options and discarding changes for the MMS command.
- **Help—**Click on this button to bring up the textual explanation of the dialog box in a window.

1.6.2. Build Activation Area

Figure 1.6, "MMS Build Activation Area" illustrates the Build activation area of the main window used to specify the target and start a build.

The Description File field displays the file specification of the description file displayed in the MMS description file area and used by MMS. You cannot set or modify the Description File field.

You can set or modify the Target field by using the keyboard or by cutting and pasting text from the description file area.

The Create MMS file... button, when selected, opens a dialog box for specifying the input for the automatic description file generator (see *Section 2.8, "Generating the MMS Description File Automatically"*).

If the Target field has a value, when you click on the Start Build button, MMS builds the specified target. If the Target field is blank, MMS builds the default target (as specified by the description file). When a

build has started, the Start Build button label changes to Stop Build. Click on the Stop Build button to abnormally terminate your application build.

Figure 1.6. MMS Build Activation Area

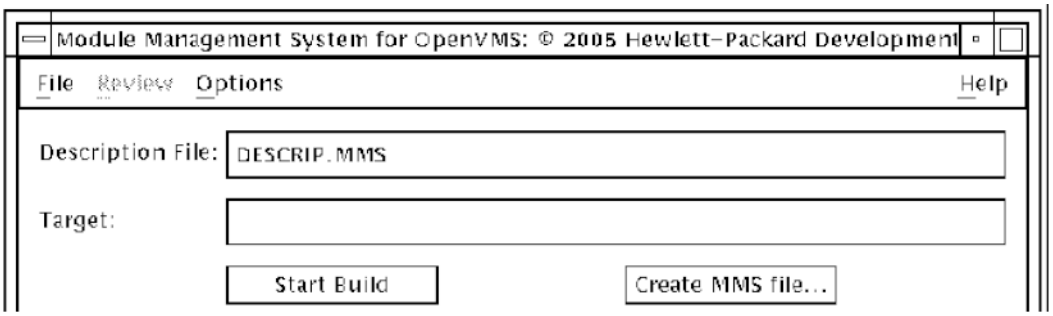


Figure 1.7. MMS Description File and Build Log Areas

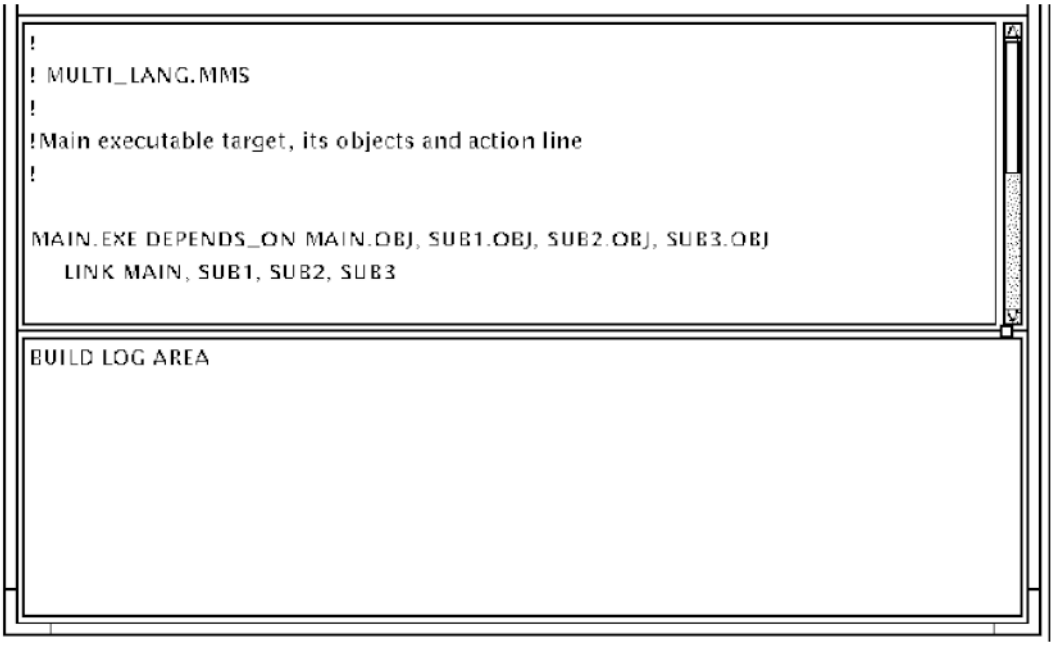


Figure 1.7, "MMS Description File and Build Log Areas" illustrates the MMS description file area and the MMS build log area.

The MMS description file area is displayed on the upper portion of the screen, while the MMS build log area is displayed on the lower portion of the screen.

The following sections describe these areas.

1.6.3. MMS Description File Area

The MMS description file area is used to display the current MMS description file.

If the DECwindows version of LSE is not presently invoked, the description file is displayed in a read-only scrollable window for browsing purposes. If the DECwindows version of LSE is currently invoked, the description file is displayed in an LSE modifiable buffer instead.

You can specify the contents of the MMS description file area using the following methods:

- Enter the description file specification on the MMS command line.
- Use the File pull-down menu options Open or Fetch.

  If a CMS library is not defined, the Fetch option is disabled.
- Click on the Create MMS File... button to automatically generate the description file.

The MMS description file area might be displayed, depending on the following settings:

- Command line qualifier (/GENERATE/[NO]SHOW_DESCRIPTION_FILE), which defaults to /GENERATE/SHOW_DESCRIPTION_FILE
- Description File toggle switch setting in MMS DECwindows Build Input/Output Options dialog box

1.6.4. Build Log Area

The Build log area is used to display the output from the most recent build. The build output is mapped to the bottom window of a split screen. The window is scrollable so you can browse the diagnostics (see *Figure 1.7, "MMS Description File and Build Log Areas"*).

If the DECwindows version of LSE is not presently invoked, the Review pull-down menu items are disabled.

If the DECwindows version of LSE is currently invoked, the diagnostics file from the first compilation is displayed in an LSE review buffer. You can review the diagnostics in the review buffer in exactly the same way as done for the COMPILE REVIEW command. As with the COMPILE REVIEW command, you can bring the source into a buffer by double clicking on a diagnostic, or by using the LSE GOTO SOURCE command or the menu option Source | Goto Source. In addition, you can navigate through all the compilations from the current build using the Review pull-down menu. For example, the Next module menu item brings up the diagnostics file from the next compilation in the LSE review buffer.

1.7. Building Software Systems

MMS can build simple or complex systems with many files of source code, multiple language compilers, and executable images. MMS builds the systems by using its built-in features and information obtained from the description file.

This section contains system-building information on the following topics:

- Single-object systems
- Multiple-object systems
- Multiple-language systems
- Multiple-include file systems
- Multiple-executable image systems
- Multiple-object library systems

1.7.1. Single-Source System

Example 1.1, "Description File Using a Single Object" shows a sample description file, called SYSTEM1.MMS, with a single object defined in it. In this example, the target MAIN.EXE has one

source file, MAIN.OBJ. The example uses comment lines, denoted by the exclamation point (!), and blank lines for readability.

Example 1.1. Description File Using a Single Object

```
!
! Description file SYSTEM1.MMS
!
MAIN.EXE DEPENDS_ON MAIN.OBJ
MAIN.OBJ DEPENDS_ON MAIN.PAS
```

Both the source file MAIN.PAS and the description file SYSTEM1.MMS must be located in your current directory. After you create the description file (see *Chapter 2, "MMS Description Files"*), you invoke MMS with the following command:

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

MMS builds the system by using the description file SYSTEM1.MMS, as follows:

1. Locates the first target (MAIN.EXE) in the description file.
2. Creates a dependency tree to determine which files need to be built.
3. Uses built-in rules to determine that MAIN.PAS is a Pascal program, then creates the target object file MAIN.OBJ from the source file MAIN.PAS and places it in your current directory. The built-in rule for building .OBJ files from .PAS files is as follows:

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

4. Uses built-in rules to determine that MAIN.OBJ is an object file, then creates the target executable file MAIN.EXE from the source file MAIN.OBJ and places it in your current directory. The built-in rule for building .EXE files from .OBJ files is as follows:

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

   Do not delete object files after using MMS, because MMS automatically recompiles all source code files at the next invocation. If no target or source files changed since the last system build, MMS displays the following message:

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

1.7.2. Multiple-Source System

Targets can have more than one source file. This section describes how to create and execute description files with multiple objects.

Example 1.2, "Description File Using Multiple Objects" shows a sample description file, called SYSTEM2.MMS, with more than one object defined in it. In this example, the target MAIN.EXE has two sources, MAIN.OBJ and SUB1.OBJ. The second line of the description file is the action line, which tells MMS how to build the target.

Example 1.2. Description File Using Multiple Objects

```
!
! SYSTEM2.MMS
!
MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ
```

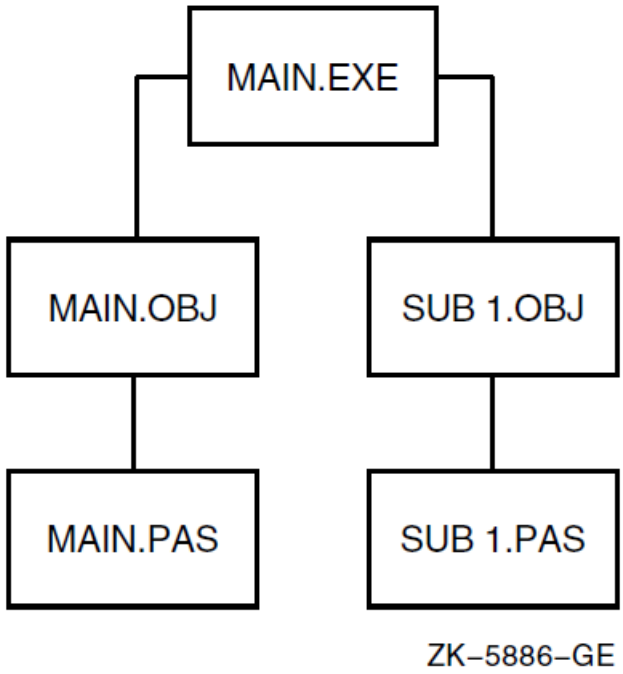
```
    LINK MAIN.OBJ, SUB1.OBJ
MAIN.OBJ DEPENDS_ON MAIN.PAS
SUB1.OBJ DEPENDS_ON SUB1.PAS
```

After you invoke MMS with the SYSTEM2.MMS description file, MMS builds the system as follows:

1. Locates the first target (MAIN.EXE) in the description file
2. Creates a dependency tree to determine which files need to be built
3. Uses built-in rules to create MAIN.OBJ from MAIN.PAS
4. Uses built-in rules to create SUB1.OBJ from SUB1.PAS
5. Uses the action line (which overrides the built-in rule that allows MMS to link only one object file) to create MAIN.EXE from the newly created MAIN.OBJ and SUB1.OBJ

Figure 1.8, "Multiple-Source System" shows the relationship between the files.

Figure 1.8. Multiple-Source System



1.7.3. Multiple-Language System

Using the built-in rules for file extensions, MMS can create object and executable files from program files created using different programming languages. MMS uses the different file types to choose the correct compiler during the system build.

Example 1.3, "Description File Using Multiple Language Compilers" shows the sample description file MULTI_LANG.MMS.

Example 1.3. Description File Using Multiple Language Compilers

```
!
! MULTI_LANG.MMS
!
!Main executable target, its objects and action line
```

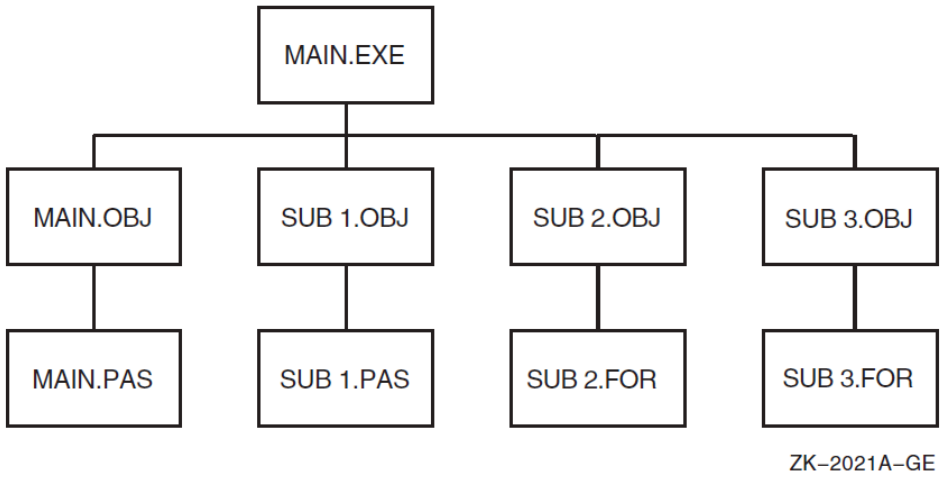
```
!
MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ, SUB2.OBJ, SUB3.OBJ
    LINK MAIN, SUB1, SUB2, SUB3
!
!Source code dependencies
!
MAIN.OBJ DEPENDS_ON MAIN.PAS
SUB1.OBJ DEPENDS_ON SUB1.PAS
SUB2.OBJ DEPENDS_ON SUB2.FOR
SUB3.OBJ DEPENDS_ON SUB3.FOR
```

After you invoke MMS with the MULTI_LANG.MMS description file, MMS builds the system as follows:

1. Locates the first target (MAIN.EXE) in the description file
2. Creates a dependency tree to determine which files need to be built
3. Uses built-in rules to create MAIN.OBJ from MAIN.PAS using the Pascal compiler
4. Uses built-in rules to create SUB1.OBJ from SUB1.PAS using the Pascal compiler
5. Uses built-in rules to create SUB2.OBJ from SUB2.FOR using the Fortran compiler
6. Uses built-in rules to create SUB3.OBJ from SUB3.FOR using the Fortran compiler
7. Uses the action line to create MAIN.EXE from the newly created MAIN.OBJ, SUB1.OBJ, SUB2.OBJ, and SUB3.OBJ

Figure 1.9, "Multiple-Language System" shows the relationship between the files.

Figure 1.9. Multiple-Language System



1.7.4. System with Include Files

Include files are frequently used in software development projects. Include files can contain code, such as a set of variables, or constant declarations. The include files are shared between developers. Individual programmers do not have to maintain their own separate copies of these include files; they just include the common file once.

If an include file changes, all developers using the include file automatically receive the new code the next time they use the common file. In this case, all source files that use the common file must be

recompiled. However, MMS detects this change and automatically recompiles the next time you perform a build.

The ability of MMS to handle include files ensures accurate system building. In a large system, you might not remember which compilation depends on which include file and you might forget to perform the compilations when an include file changes. When writing your MMS description file, inspect all your source code files for statements that include other files. You can use the DCL command SEARCH to search for these statements.

The description file must specify any include files on the same line as the program code that uses them, as in *Example 1.4, "Description File Using Include Files"*. You can list the include files in any order.

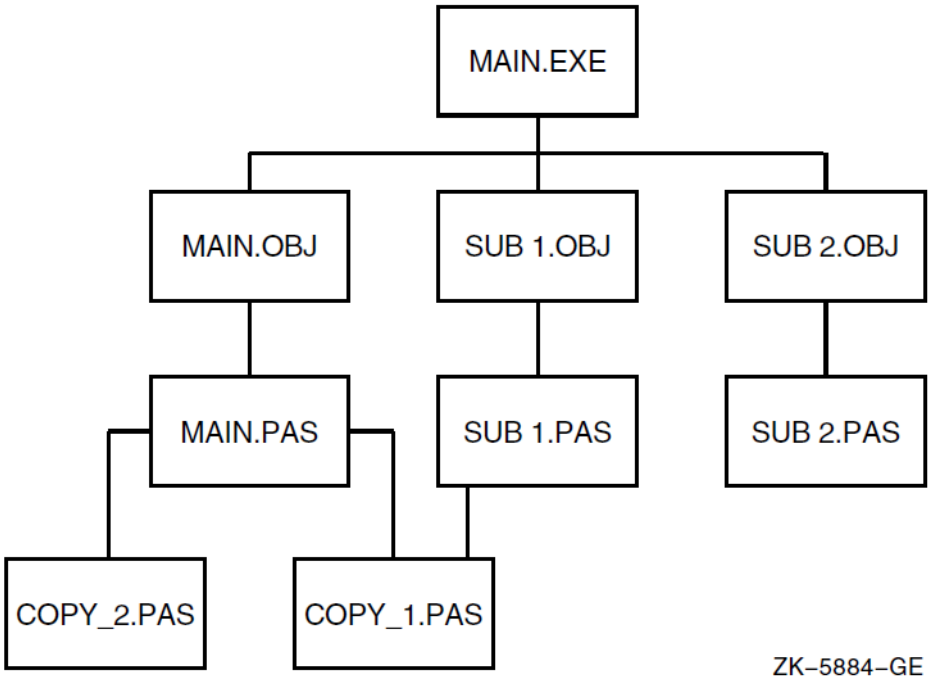
Example 1.4. Description File Using Include Files

```
!
! INCLUDE.MMS
!
! Main executable target, its objects, and action line
!
MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ, SUB2.OBJ
    LINK MAIN, SUB1, SUB2
!
! Source code files with include files COPY_1.PAS and COPY_2.PAS
!
MAIN.OBJ DEPENDS_ON MAIN.PAS, COPY_1.PAS, COPY_2.PAS
SUB1.OBJ DEPENDS_ON SUB1.PAS, COPY_1.PAS
SUB2.OBJ DEPENDS_ON SUB2.PAS
```

When MMS processes the description file INCLUDE.MMS in *Example 1.4, "Description File Using Include Files"*, it also processes the include files COPY_1.PAS and COPY_2.PAS when it processes MAIN.PAS. The include file COPY_1.PAS is processed again when MMS processes SUB1.PAS.

Figure 1.10, "Include Files in a System" shows the relationship between the files.

Figure 1.10. Include Files in a System



If you specify files to be included that do not exist, MMS displays an error message and stops.

1.7.5. System with Multiple Targets

If an executable image is especially complicated, place it in its own description file. Also place executable images that are not related in some significant way in separate description files. However, if a system has a number of executable images that use common object files, it is more efficient to build them using one description file.

This type of description file is layered with the overall target first, followed by the executable images, object files, and source code files. You can choose to build the entire system or selected executable files.

Example 1.5, "Description File Using Multiple Targets" shows the sample description file MULTI_EXES.MMS.

Example 1.5. Description File Using Multiple Targets

```
!
! MULTI_EXES.MMS
!
! Overall system target--this is a sample target name
!
SYSTEMX DEPENDS_ON MAIN.EXE, PROG1.EXE, PROG2.EXE
    ! no special action intended for overall system target
!
! What follows is the executable images and their object files
!
MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ
    LINK MAIN.OBJ, SUB1.OBJ
PROG1.EXE DEPENDS_ON PROG1.OBJ
    LINK PROG1.OBJ
PROG2.EXE DEPENDS_ON PROG2.OBJ
    LINK PROG2.OBJ
!
! The object files and their sources
!
MAIN.OBJ DEPENDS_ON MAIN.PAS
SUB1.OBJ DEPENDS_ON SUB1.PAS
PROG1.OBJ DEPENDS_ON PROG1.PAS
PROG2.OBJ DEPENDS_ON PROG2.PAS
```

In this example, the description file MULTI_EXES.MMS contains the target SYSTEMX, which is the overall target of building the system. There are no built-in rules for targets that do not have file types, and because no action line is specified, SYSTEMX has a null action and simply builds the executable files. The three executable images come from their object files. All the object files and source files have the same dependencies as in the previous examples.

After you invoke MMS with the MULTI_EXES.MMS description file, MMS builds the system as follows:

1. Locates the first target (SYSTEMX) in the description file

2. Creates a dependency tree to determine which files need to be built

3. Uses built-in rules to create MAIN.OBJ from MAIN.PAS using the Pascal compiler

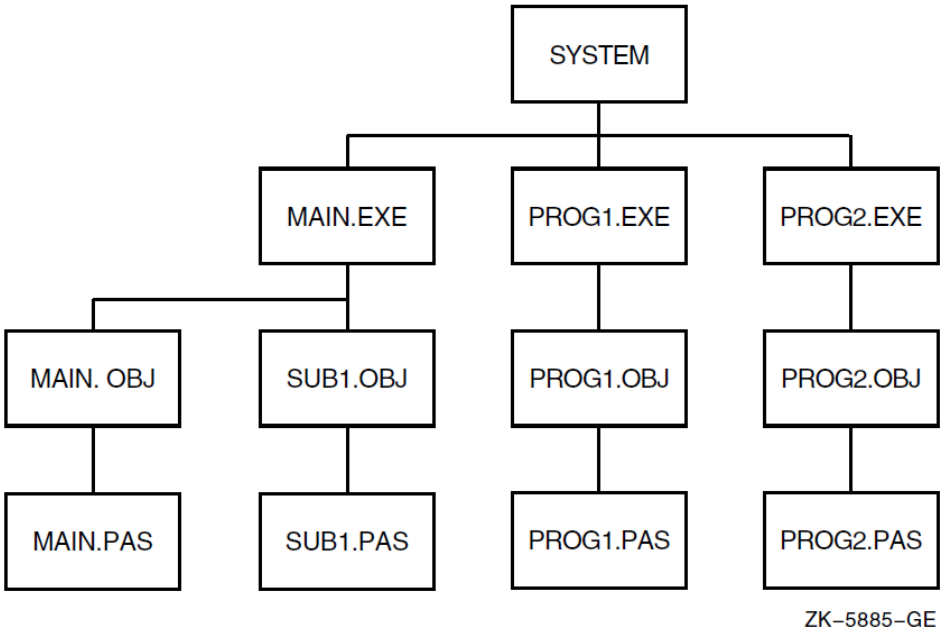
4. Uses built-in rules to create SUB1.OBJ from SUB1.PAS using the Pascal compiler

5. Uses the action line to create MAIN.EXE from MAIN.OBJ and SUB1.OBJ

6. Locates the second target (PROG1.EXE) in the description file, and so on

Figure 1.11, "System with More than One Executable Image" shows the relationship between the files.

Figure 1.11. System with More than One Executable Image



Building a Specific Target

In a multiple-target system, to build a specific target or executable image you must include a target name on the command line. For example:

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

When you specify the target name on the command line, MMS overrules its default of building the first target in the description file; only the targets you specify are built.

1.7.6. System with an Object Library

This section assumes that you have some knowledge of OpenVMS libraries. Object libraries are useful for quick compiling and linking during debugging sessions. MMS creates libraries, inserts modules, and updates libraries during software system builds.

An object library is a single file that contains multiple object files, otherwise known as object modules. The object library is usually named with a file type of .OLB. The object file name is an OpenVMS file name, usually named with a file type of .OBJ. The object module name is governed by the TITLE, MODULE, PROGRAM, or SUBROUTINE name in the source file. The object file name and object module name are frequently the same.

To include an object library in an MMS description file, you must include the name of the library, object module, and object file. The object library name follows the target's main source. The object module

name and the object file name follow the library name, enclosed in parentheses and separated with an equal sign. No spaces are allowed.

Example 1.6, "Description File Using Object Libraries" shows the sample description file OBJECT_LIB.MMS.

Example 1.6. Description File Using Object Libraries

```
!
! OBJECT_LIB.MMS
!
! Main executable target, its objects, and action line
!
MAIN.EXE DEPENDS_ON   MAIN.OBJ, -
                      MAIN_LIB.OLB(OPTIM=OPTIM.OBJ), -
                      MAIN_LIB.OLB(GET_RECORD=GETREC.OBJ), -
                      MAIN_LIB.OLB(PUT_RECORD=PUTREC.OBJ)
     LINK MAIN.OBJ, MAIN_LIB/LIB
!
! Program source code files
!
MAIN.OBJ DEPENDS_ON MAIN.FOR
OPTIM.OBJ DEPENDS_ON OPTIM.FOR
GETREC.OBJ DEPENDS_ON GETREC.FOR
PUTREC.OBJ DEPENDS_ON PUTREC.FOR
```

In this example, the executable image MAIN.EXE depends on one object library (MAIN_LIB.OLB). The object module OPTIM comes from the object file OPTIM.OBJ. The object modules GET_RECORD and PUT_RECORD come from the object files GETREC.OBJ and PUTREC.OBJ, respectively. Note that the first object module and object file name have the same name, and the following two object modules and file names have different names.

MMS builds the object library in *Example 1.6, "Description File Using Object Libraries"* as follows:

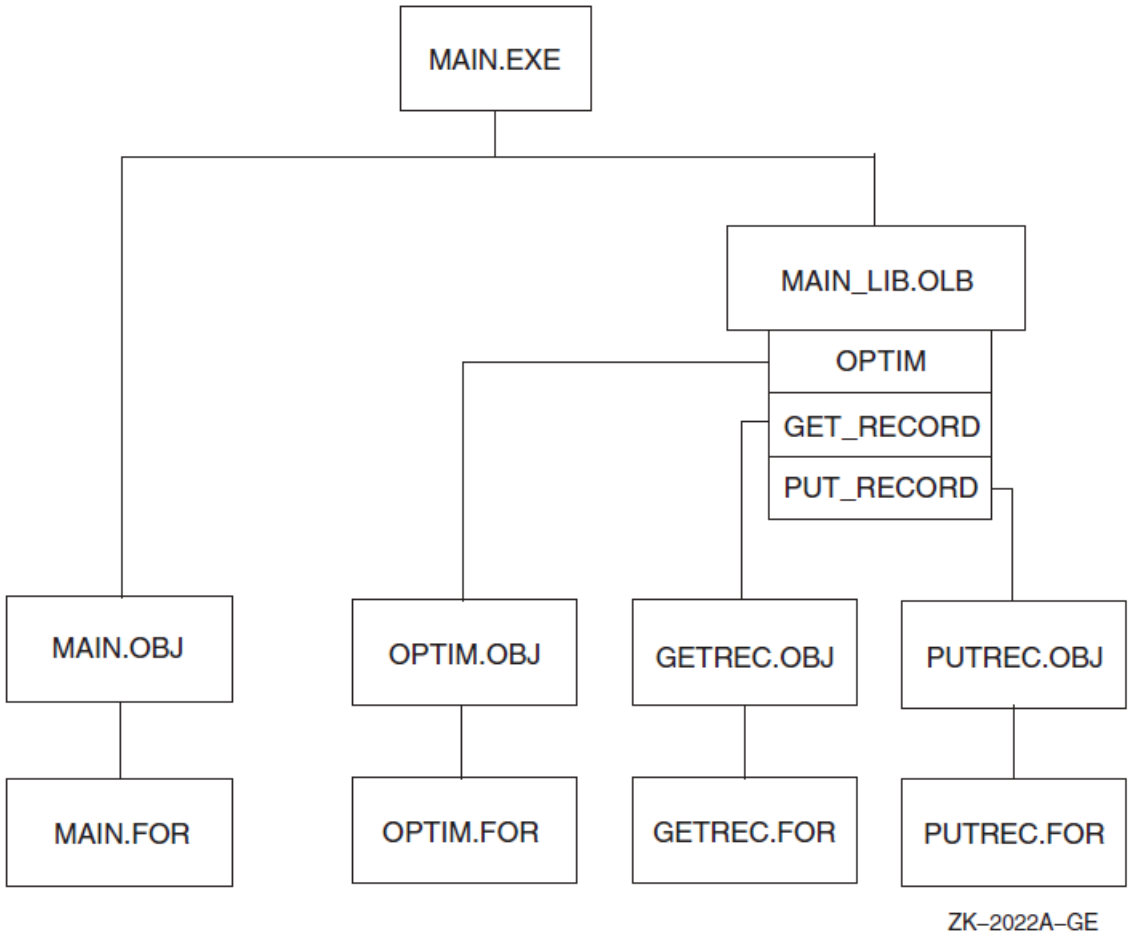
1. Compiles the source file for each object file

2. Checks for the existence of the library and creates it if it does not exist

3. Inserts each object file into the library (or replaces it, if it already exists)

Figure 1.12, "Object Library in a System" shows the relationship between the files.

Figure 1.12. Object Library in a System



1.8. Rebuilding Software Systems

During the development cycle, MMS can determine which components in a system are missing or 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 can then update them. The entire system is not rebuilt, only those components whose sources have been modified. MMS checks the modification dates of executable files against their source files. If the source files are newer than the executable files, MMS rebuilds the executable files; if the source files are older than the executable files, MMS determines that the executable files are up-to-date and does not rebuild them. MMS builds only targets whose sources are newer than their targets, saving you time and disk space.

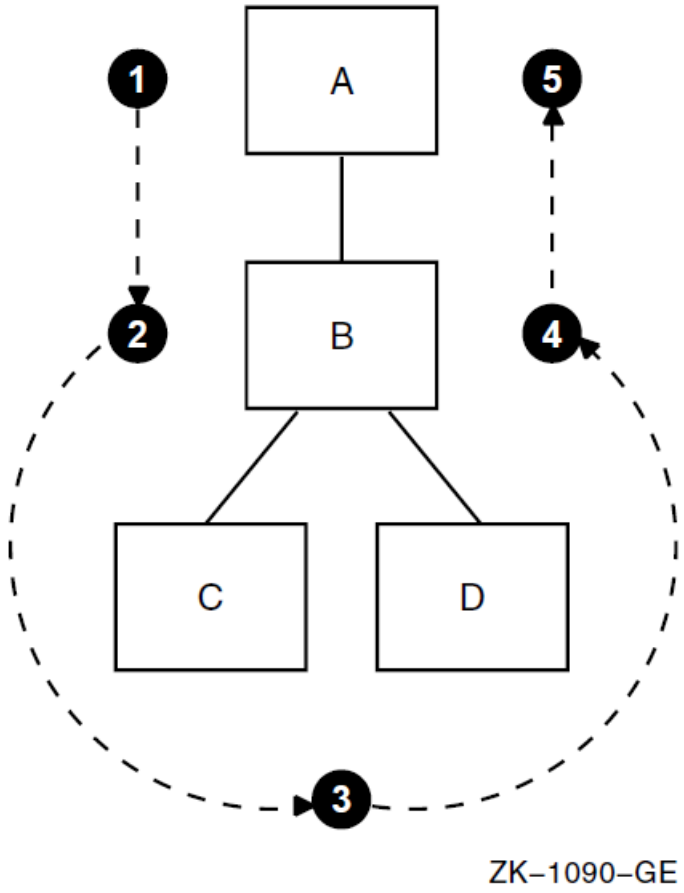
To rebuild a system, invoke MMS as you would for an initial build.

Figure 1.13, "How MMS Rebuilds a System" 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, B is a source file for A, and C and D are source files for B. The commands that update B (by using C and D) and A (by using the updated B) are the actions.

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

If any file listed in the description file is missing, MMS attempts to create it from the missing file's source. For example, if MAIN.OBJ is deleted from the directory, when MMS builds the system, it creates MAIN.OBJ from the MAIN.PAS program file.

Figure 1.13. How MMS Rebuilds a System



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

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

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

4. If the times of Components C or D are more recent than that of Component B, MMS updates B according to the action lines that you specify in the description file. If B is more recent than C and D, MMS does not do anything because B is already up-to-date.

5. Once Component B is updated, it is more recent than the target Component A; therefore, MMS updates Component A.

If you delete an object file but still have the executable file, MMS recompiles the source file and also relinks the executable file, even though your executable file is still compatible with your source file. MMS works from the bottom up and propagates any change up the dependency tree. However, if you delete your source file, MMS returns a fatal error because it cannot recreate source code from object or executable files.

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

1.8.1. Single-Source System

When rebuilding single-source systems, MMS checks the system from the bottom up to see if it is complete and up-to-date. If any part of the system is missing or any target is older than its sources, the system is recreated.

For example, if you edit the source file MAIN.PAS, it would be newer than its object file, MAIN.OBJ. When you invoke MMS, it does the following:

1. Finds the first target in the description file (MAIN.EXE)
2. Finds the source for MAIN.EXE (MAIN.OBJ) and its source (MAIN.PAS)
3. Compiles the source code file MAIN.PAS because MAIN.PAS is newer than its target MAIN.OBJ
4. Relinks the object file MAIN.OBJ because MAIN.OBJ is now newer than its target, MAIN.EXE

1.8.2. Multiple-Source System

If you edit one of the source files in a multiple-source system (for example, the source file SUB1.PAS), it will be newer than its object file, SUB1.OBJ. When you invoke MMS, it does the following:

1. Finds the first target in the description file (MAIN.EXE).
2. Finds the sources for MAIN.EXE (MAIN.OBJ and SUB1.OBJ) and their sources (MAIN.PAS and SUB1.PAS).
3. Compiles the source code file SUB1.PAS because SUB1.PAS is newer than its target SUB1.OBJ.
4. Uses the action line to create MAIN.EXE from MAIN.OBJ and SUB1.OBJ.
5. Because MAIN.OBJ is newer than MAIN.PAS, MMS does not compile MAIN.PAS. However, MAIN.EXE is now older than one of its source files (SUB1.OBJ), so MMS relinks the object file MAIN.OBJ, using the action line.

1.8.3. Multiple-Language System

If you edit two source files in a multiple-language system (for example, MAIN.PAS and SUB3.FOR), they will be newer than their object files, MAIN.OBJ and SUB3.OBJ. MMS compiles only the source code that has been updated, and uses the correct language compiler in each case. It then links all the objects to recreate the executable image.

1.8.4. System with Include Files

If you edit the include file COPY_1.PAS in a multiple-programming-language system, it is newer than any other source file used to build MAIN.EXE. When you invoke MMS, both MAIN.PAS and SUB1.PAS (which both include COPY_1.PAS) are recompiled, and all necessary objects are relinked.

If you also edit COPY_2.PAS and then rebuild the system, only MAIN.PAS is recompiled, and all necessary objects are relinked.

1.8.5. System with Multiple Targets

If you delete one of the executable files in your current directory (for example, PROG2.EXE) when you invoke MMS, it must link PROG2.OBJ to produce PROG2.EXE. MMS performs only the link necessary to complete the system.

1.8.6. System with an Object Library

If you edit the source file GETREC.FOR when you invoke MMS, it does the following:

1. Locates the first target (MAIN.EXE) in the description file
2. Creates a dependency tree to determine which files need to be rebuilt
3. Uses built-in rules to recompile GETREC.FOR, thereby producing GETREC.OBJ
4. Uses built-in rules to replace module GET_RECORD in the object library MAIN_LIB.OLB with the new GETREC.OBJ
5. Uses the action line to create a new version of MAIN.EXE

Chapter 2. MMS Description Files

The first step in using MMS is to write a description file for the system you want to build. The description file is a text file that describes how to build a software system, and explains the relationships among the various components of your system. The description file can contain some or all of the following elements:

- Targets (usually executable images)
- Intermediate files (usually object files)
- Source files (usually program code)
- Action lines
- Comment lines
- Built-in rules
- User-defined rules
- Directives

This chapter describes how to create a description file, and how the elements of the description file work together to build a system.

2.1. Creating the Description File

You create and modify the description file with any text editor. For example:

```
$ LSEDIT DESCRIP.MMS
```

When you invoke MMS, it first attempts to process the default description file DESCRIP.MMS in your current directory. If DESCRIP.MMS does not exist, MMS attempts to process a description file called MAKEFILE. If MAKEFILE. does not exist, MMS attempts to process a description file called targetname.MMS. If all of these files exist, MMS uses only DESCRIP.MMS. If none of these files exist, MMS uses built-in rules to build the target.

You can also invoke MMS with a specified description file, called SYSTEM1.MMS in this example:

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

MMS uses the .MMS file type when none is specified.

Once MMS locates the description file, it processes it by building the first target in the description file. However, if the description file is target-name.MMS, MMS attempts to build the actual target on the target-name (see *Section 2.1.2, "Specifying the Target on the Command Line"*).

To direct MMS to override a description file, use the /NODESCRIPTION qualifier and the target name on the MMS command line, as follows:

```
$ MMS/NODESCRIPTION filename
```

When you specify /NODESCRIPTION, MMS does not look for a description file, but instead relies entirely on its built-in rules to update the target. See *Chapter 5, "Command Dictionary"* for more information on the /[NO]DESCRIPTION qualifier.

2.1.1. Writing Dependency Rules

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

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

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

target

Specifies an OpenVMS file specification or a mnemonic name (*Section 2.1.3, "Using Mnemonic Names for Targets and Sources"* describes mnemonic names). The file specification can be complete, including node information. MMS locates the target file in your current default directory unless you specify another directory in your file specification.

source

Specifies an OpenVMS file specification or a mnemonic name (*Section 2.1.3, "Using Mnemonic Names for Targets and Sources"* describes mnemonic names). The file specification can be complete, including node information. MMS locates source files in your current default directory unless you specify other directories in your file specification.

comment

Specifies a string of text, introduced by an exclamation point (!). The comment provides detailed information in the description file. You can continue a comment line onto the next line with a hyphen or backslash. MMS considers any text on the next line following the continuation character as part of the comment line.

action line

Specifies a command-language command that MMS uses to update the target.

You can specify any number of action lines for a target. An action line is positioned below the corresponding target or source line and must be indented by at least one space or tab. MMS interprets all indented lines as action lines and associates them with the most recently specified target or source line.

If you omit the action line, MMS uses built-in rules to update the target if a built-in rule exists. (See *Section 2.2, "Using Built-In Rules"* for an explanation of built-in rules.)

You begin a target or source line in column 1 of the line and use the keyword DEPENDS_ON or a colon (:) to separate the target from the source. If you use a colon to separate the target from the source, insert at least one space or tab on either side of the colon, or MMS will interpret the colon as part of an OpenVMS file specification.

To improve the readability of description files, separate dependency rules from each other with one or more blank lines.

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 DEPENDS_ON - 1
   TEST1.BAS, - 2 ! Source modules for TESTS.OBJ- 3
   TEST2.BAS, -
```

```
   TEST3.BAS, -
   TEST4.BAS, -
   TEST5.BAS
      BASIC/OBJECT=TESTS TEST1+TEST2+TEST3+TEST4+TEST5
```

Key to the example:

1. The hyphen means that the next line is treated as part of the current line.
2. The second through fifth lines are continuations of the target or source line.
3. A comment can appear after a continuation character without affecting the processing of the description file.

Note

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

2.1.2. Specifying the Target on the Command Line

By default, MMS updates the first target specified in the description file.

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

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

If you specify MMS PRINT on the command line, MMS searches the description file for the dependency rule associated with PRINT, the specified target. MMS tries to update the target PRINT rather than TEST.OBJ, the default. If PRINT is up-to-date, no action takes place. See the description of the /NODESCRIPTION qualifier in the *Chapter 5, "Command Dictionary"* for more information.

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

2.1.3. Using Mnemonic Names for Targets and Sources

You can use a mnemonic name for a source or target, but you must supply the action lines that update the target. A mnemonic name is a name that identifies the purpose of a sequence of related actions. MMS relies on the source and target file types to apply built-in rules. *Section 2.2, "Using Built-In Rules"* describes how MMS uses built-in rules.

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

Mnemonic names are useful in several cases. For example:

- To update more than one file

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

By default, MMS builds only one target. To update several targets, make them sources in a dependency rule by using a mnemonic name for the target, as follows:

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

MMS considers the target NEW_SYSTEM as updated when it executes the action line or lines that follow it. Both A.EXE and B.EXE are updated, if necessary.

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

```
ALL DEPENDS_ON PROG.EXE, PRINT
   ! system completely built and the sources printed
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
   LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
PRINT DEPENDS_ON MOD1.C, MOD2.C, MOD3.C, DEFS1.H, DEFS2.H
   ! Print the source files
   PRINT MOD1.C, MOD2.C, MOD3.C, DEFS1.H, DEFS2.H
```

2.1.3.1. Specifying Target and Source Files

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

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

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

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

2.1.3.2. Specifying Multiple Targets and Sources

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

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

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

Because there is no action line, MMS uses built-in rules to determine what action is needed to update KERNEL.OBJ and DRIVER.OBJ and expands the previous rule, as follows:

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

MMS determines from the built-in rules that KERNEL.C and DRIVER.C are the sources for KERNEL.OBJ and DRIVER.OBJ, respectively. When both targets need updating, the original dependency rule expands to two dependency rules, resulting in two separate compilations.

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

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

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

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

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

Occasionally, MMS executes 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:

- If sources are being stored in a library to which more than one person has access, someone else might replace that source in the library after you have invoked MMS, but before MMS has checked the source's revision time. Therefore, when MMS checks the time, a source newer than the corresponding target might exist in the library. MMS would then execute the action to update the target.
- If the sources and targets in your description file do not reside on the same node of a network, the clocks on the nodes might not be synchronized and a source might have a revision time that is later than the target. In addition, in an OpenVMS Cluster environment, clocks on different nodes of the cluster might not be synchronized.

2.1.3.3. Hierarchy of Dependency-Rule Application

MMS has a hierarchy of rule application, as follows:

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

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

2.2. Using Built-In Rules

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

Built-in rules are made up of default macros, special macros, and string variables. A complete list of the MMS built-in rules is in *Table C.6, "MMS Built-In Rules"*. A file copy of the built-in rules resides in the following location:

```
SYS$COMMON:[SYSHLP.EXAMPLES.MMS]MMS$DEFAULT_RULES.MMS.
```

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

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

Built-in rules consist of the file extension of the source, the file extension of the target, and the action to update the target using the source. The actions in built-in rules use macros extensively, as shown in *Example 2.1, "Built-In Rule"*.

Example 2.1. Built-In Rule

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

Key to the example:

❶ PAS is the source file type.

❷ .OBJ is the target file type.

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

MMS attempts to use its built-in rules only when you omit the action line from a dependency rule. For example, MMS has a built-in rule that instructs it to use .FOR files when updating .OBJ files, and to produce the .OBJ files by invoking the Fortran compiler. However, these rules only apply if the target and source have the same file name. The following examples describe this relationship in detail:

- Same file name, with source and action line—In writing the description file, you can fully state dependencies and actions, as follows:

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

- Same file name, with source—You can rely on MMS built-in rules by omitting the action line, as follows:

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

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

- Same file name, no source—If you omit the source, MMS can still use built-in rules to locate it because MMS knows about implied dependencies among files with the same name but different file types. MMS uses its suffixes precedence list to determine which file type (source) would result in the target file type. In the previous example, because the target's file name is MOD3, MMS assumes that the source's file name is also MOD3.

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

  ```
  MOD3.OBJ :
  ```

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

- Different file name, no action line—Consider the following line, which shows a target with a different file name, but no action line:

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

  In this example, MMS cannot use its built-in rules to build the target, because of the different file names. In addition, a missing action line means that MMS has no information to proceed, so it issues an error message.

2.2.1. Suffixes Precedence List

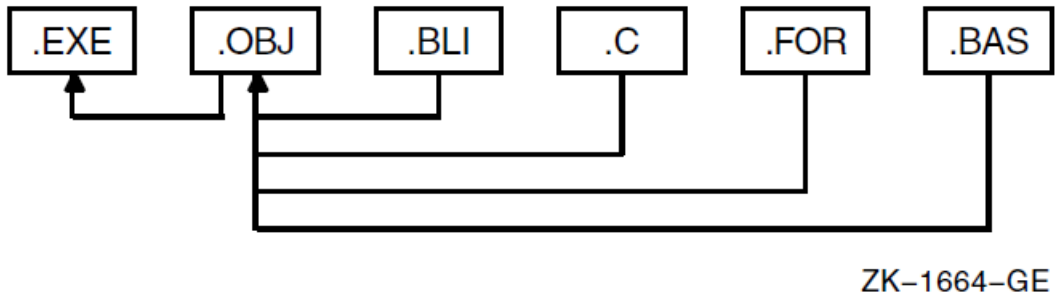
MMS checks its suffixes precedence list to determine the file type of the source and then uses the built-in rules to determine how the various types of files can be generated from the known rules. Consider the following suffixes precedence list:

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

According to this list, .EXE files have precedence over .OBJ files, which have precedence over .BLI files, which have precedence over .C files, and so on.

Figure 2.1, "Relationship Between Suffixes" shows the relationship between the suffixes list and the known rules.

Figure 2.1. Relationship Between Suffixes



The arrows in this figure indicate built-in rules. In this figure, a known rule specifies how an .EXE file is made from an .OBJ file. Similarly, the built-in rules direct MMS how to make an .OBJ file from a .BLI file, .C file, .FOR file, and .BAS file. Because .BLI precedes .C in the suffixes list, .BLI files have priority over .C files as a way to build .OBJ files. (The suffixes precedence list is contained in *Table C.4, "Suffixes-Precedence List"*. You can alter the order of the suffixes precedence list, as described in *Section 2.7.6, ".SUFFIXES, .SUFFIXES_AFTER, .SUFFIXES_BEFORE, and .SUFFIXES_DELETE Directives"*.)

The figure also shows that .OBJ files can be built from .BLI, .C, .FOR, and .BAS files. If MMS is trying to build MOD3.OBJ, it looks first for a source named MOD3.BLI. If such a source exists in the specified directory, MMS applies the known rule and creates MOD3.OBJ. If it finds no match for the file name and type, it continues looking in the specified directory for the same file name and the next file type from the suffixes list that can update the target. If MOD3.BLI does not exist, MMS next looks for MOD3.C. If MOD3.C does not exist, the next possible source is MOD3.FOR, and so on.

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

```
MOD3.OBJ :
```

This rule equates to the full dependency rule as follows:

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

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

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

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

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

```
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
   LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
MOD1.OBJ DEPENDS_ON MOD1.C
   CC MOD1.C
MOD2.OBJ DEPENDS_ON MOD2.C, DEFDIR:DEFS1.H, DEFDIR:DEFS2.H
   CC MOD2.C
MOD3.OBJ DEPENDS_ON MOD3.C, DEFDIR:DEFS2.H
   CC MOD3.C
```

Example 2.2, "Description File Using Built-In Rules" shows a description file of the same system that takes advantage of MMS built-in rules.

Example 2.2. Description File Using Built-In Rules

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

```
MOD2.OBJ, MOD3.OBJ DEPENDS_ON DEFDIR:DEFS2.H ❸
MOD2.OBJ DEPENDS_ON DEFDIR:DEFS1.H ❹
```

Key to the example:

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

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

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

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

2.2.2. Default Macros

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

2.3. Defining Your Own Macros

In addition to providing built-in rules, MMS allows you to define your own rules. Defining your own rules can involve deleting, adding to, or replacing the built-in rules. *Section 2.5, "Defining Your Own Rules"* describes when and how to define new rules.

MMS allows you to use three kinds of macros: default (OpenVMS utilities or qualifiers), special (target or source file names), and user-defined. These macros can use other macros in their definition. The full list of default macros is in *Table C.1, "MMS Default Macros"* and the list of special macros is in *Table C.3, "MMS Special Macros"*.

In MMS, macros contain the following information:

- The name of the hardware platform on which MMS is running (Alpha, VAX, or IA64)
- The names of compilers, the linker, and library utilities
- The default qualifiers for compiling, linking, and using the library utilities
- The file name of the target
- The list of sources for each target

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

Macro	Default Value	Description
PASCAL	PASCAL	Specifies the Pascal compiler.
PFLAGS	/NOLIST/OBJECT= $(MMS $TARGET_NAME)	The default qualifiers for the Pascal compiler.

MMS$TARGET_ NAME	Depends on target or source line	Lists the name of the target file.
MMS$SOURCE	Depends on target or source line	Lists the name of the first file in the source list.
MMS$SOURCE_ LIST	Depends on target or source line	Lists the names of all files in the source list.

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

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

2.3.1. Formatting Macro Definitions

A macro definition has the following format:

```
name = string
```

name

A macro name can be of unlimited length and can consist of any alphanumeric characters, including the dollar sign ($), period (.), or underscore (_). Note that macro names cannot contain any of the following characters or character sequences:

```
$( ) sequence
carriage return
control characters
equal sign ( = )
quotation marks ( " )
space
tab
```

string

The macro string is the text that replaces the macro name when the macro is expanded. A macro string can consist of any character sequence. You can use a hyphen (-) as a continuation character to continue a macro string onto the next line of the description file.

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

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

```
$(name)
```

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

Note that you must define a macro before you can use it; otherwise, the macro's expanded value is the null string. To determine whether a macro has been defined, keep in mind the order in which MMS processes macro definitions (see *Section 2.3.2, "Order of Processing Macros"*).

2.3.2. Order of Processing Macros

When processing macros, MMS examines definitions from various locations, normally in the following order:

1. Command line
2. Description file
3. Built-in
4. Command Line Interpreter (CLI) symbol

The order in which locations are examined can be altered by using the /OVERRIDE qualifier (see *Chapter 5, "Command Dictionary"* for a description). Once MMS finds a definition for a macro, it does not search those locations farther down the list for more definitions.

If MMS finds two or more definitions of the same macro in the description file, it issues an error message and uses the first definition found. However, you can alter this behavior by using the /EXTENDED_SYNTAX qualifier (see *Section 2.3.7, "Redefining Macros in a Description File"*).

2.3.3. Nested Macro Expansion

MMS allows you to nest functions and macro substitutions. Functions are always processed before macros, at any nesting level. However, macros that appear as function arguments are considered to be at a higher level than the function itself and are processed first. If a function value or macro substitution reveals a new function or macro, this new function or macro is processed in turn. For example, the following is supported:

```
x = (y)
y = z
z = test
a = $($$(x))    ! a receives the value "test"
```

If a syntax error is discovered in a function call as the description file is being parsed, an appropriate diagnostic message is issued. This message includes the name of the description file and the line number on which the error was detected.

Note

Action lines are processed just prior to execution, which occurs after the entire description file has been parsed. Therefore, description file and line number information is not available for function calls that appear in action lines.

2.3.4. Using Predefined Functions

MMS provides functions that allow the following to be performed in description files:

- Text processing operations

- File specification operations
- Operations that determine the origin of macros

Note that these functions can only be used if extended syntax has been enabled. To enable this feature, use the /EXTENDED_SYNTAX qualifier when invoking MMS from the command line, or click the Extended Syntax check box in the Build Definitions/Directives Options dialog box.

The syntax of a function call is as follows:

```
$(function arguments)
```

The name of the function is specified along with text arguments on which the function operates. MMS substitutes and inserts text returned by the function into the description file at the point of the call, similar to macro substitution.

function

The name of one of the predefined MMS functions. There is no provision for user-defined functions.

arguments

The arguments of the function, which are separated from the function name by one or more whitespace characters (spaces or tabs). Separate multiple arguments with commas (these commas are not part of an argument's value).

However, except for the first whitespace character between the function name and the first argument, any additional leading or trailing whitespace characters are considered to be part of the arguments in which they appear. Each argument must be specified; arguments are not allowed to be null.

Where an argument to a function is considered to be a list of words, a word is defined as any sequence of characters that does not contain whitespace characters (spaces or tabs) and that is terminated by a whitespace character or a right parenthesis. Functions that generate word lists produce lists that are white space compressed—that is, the lists do not contain leading or trailing spaces, and a single space character separates each word in the list.

The text for each argument is processed by substitution of function calls and macros (in that order) to produce the argument value, which is the text on which the function acts. The substitution is done in the order in which the arguments appear, from left to right.

Commas and unmatched parentheses cannot appear in the text of an argument as written. These characters can be put into the argument value by macro substitution.

2.3.4.1. Text Processing Functions

The following MMS functions perform text processing operations.

ADDPREFIX

```
$(ADDPREFIX prefix,text)
```

This function prepends the value of prefix to the start of each word specified by text.

ADDSUFFIX

```
$(ADDSUFFIX suffix,text)
```

This function appends the value of suffix to the end of each word specified by text.

FILTER

```
$(FILTER pattern...,text)
```

This function filters text. Words specified by text that do not match any words specified by pattern are removed. Pattern words can contain the wildcard characters asterisk (*) and percent sign (%).

FILTER-OUT

```
$(FILTER-OUT pattern...,text)
```

This function filters text. Words specified by text that match any words specified by pattern are removed. Pattern words can contain the wildcard characters asterisk (*) and percent sign (%).

FINDSTRING

```
$(FINDSTRING find,text)
```

This function performs a string search. If the value of the string specified by find occurs in the source text specified by text, the value of find is returned. Otherwise, the value is empty.

FIRSTWORD

```
$(FIRSTWORD text)
```

This function returns the first word in the source text specified by text.

FOREACH

```
$(FOREACH macro,list,text)
```

This function repeatedly expands text. For each word specified by list, the value of text is repeated with the value of macro defined as the word from list.

JOIN

```
$(JOIN list,text)
```

This function concatenates words. Each word specified by text is appended to the corresponding word in list forming a new word in the result. When the number of words in list and text are not the same, the remaining words from the longer list are simply appended to the result.

PATSUBST

```
$(PATSUBST pattern...,to,text)
```

This function performs pattern substitution. Each word specified by text that matches a word specified by pattern is replaced by the value of to. Pattern words can contain the wildcard characters asterisk (*) and percent sign (%).

If the value of to also contains wildcard characters, the wildcards are replaced by the text that matched the wildcard characters in pattern.

SORT

```
$(SORT text)
```

This function sorts text. The words specified by text are sorted and arranged in lexical order; duplicate words are removed.

STRIP

```
$(STRIP text)
```

This function performs whitespace compression. Leading and trailing whitespace is removed from source text specified by text. Each internal sequence of whitespace characters is replaced by a single space.

SUBST

```
$(SUBST from,to,text)
```

This function performs string substitution. Each occurrence of the value from in the source text specified by text is replaced by the value of to.

WORD

```
$(WORD n,text)
```

This function returns a word (based on its position) from the source text specified by text. The value of n should range from 1 to the total number of words in the list. If n is not in this range, the result is empty.

WORDS

```
$(WORDS text)
```

This function returns the number of words in text.

2.3.4.2. File Specification Functions

The following functions operate on file specifications. Each function has a single argument—a list of words where each word is considered to be an OpenVMS file specification.

BASENAME

```
$(BASENAME text)
```

For each file specification specified by text, this function returns the name portion of the specification (omitting the file type and version).

DIR

```
$(DIR text)
```

For each file specification specified by text, this function returns the directory portion of the specification (omitting the file name, type, and version).

FILETYPE

```
$(FILETYPE text)
```

For each file specification specified by text, this function returns the file type portion of the specification (omitting the file name and version).

FILEVERSION

```
$(FILEVERSION text)
```

For each file specification specified by text, this function returns the version portion of the specification (omitting the file name and type).

NOTDIR

```
$(NOTDIR text)
```

For each file specification in text, this function returns the file name and type portion of the specification (omitting the directory and version).

WILDCARD

```
$(WILDCARD text)
```

This function is used to search for files. It returns the name and type portion of files that match the file specifications specified by text. The file specifications may contain the wildcard characters asterisk (*) and percent sign (%).

2.3.4.3. Macro Functions

The following function locates the origin of one or more macros.

ORIGIN

```
$(ORIGIN macro)
```

This function returns the one of the following text strings, which indicates the source of the definition of macro:

String	Description
FILE	Defined in a description file
COMMAND LINE	Defined on the command line
SPECIAL	A special macro
DEFAULT	A default macro
CLI SYMBOL	A CLI symbol
TEMPORARY	Defined by function FOREACH
UNDEFINED	The specified macro is not defined

2.3.5. Invoking Macros

A macro string can also contain macro invocations that are expanded when the macro is defined. For example, suppose the following macro definitions appear in your description file:

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

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

MMS macros are not recursive if the default /NOEXTENDED_SYNTAX qualifier is in place. 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. See the /EXTENDED_SYNTAX qualifier description in the *Chapter 5, "Command Dictionary"*.

Example 2.3, "Macro Definitions in a Description File" shows a description file, CPROG.MMS, that defines two macros: FNAME, which expands to the string TESTS; and CCQUALS, which expands to the string /NOLIST.

Example 2.3. Macro Definitions in a Description File

```
FNAME = TESTS
CCQUALS = /NOLIST
$(FNAME).EXE : $(FNAME).OBJ, SYS$LIBRARY:STARLET.OLB
   LINK $(FNAME),-
      SYS$LIBRARY:STARLET.OLB/LIB
$(FNAME).OBJ : $(FNAME).C
   CC $(CCQUALS) $(FNAME).C
```

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

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

2.3.6. 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

This is an OpenVMS file specification or a logical name for a file that contains only macro definitions. The default file type is .MMS.

macro

This is a macro definition enclosed in quotation marks. Use the same format that you would use to define a macro in a description file: name = string.

If you specify more than one macro, separate the macros with commas and enclose the list in parentheses. The /MACRO qualifier is described in detail in the *Chapter 5, "Command Dictionary"*.

2.3.6.1. Redefining User-Defined Macros

To build a new program called TEST1.EXE, you can use the same description file with which you built TESTS.EXE (as shown in the *Example 2.3, "Macro Definitions in a Description File"*). You can redefine FNAME and override the macro definition in the description file, as follows:

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

MMS then interprets the description file:

```
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, 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
```

Next, invoke MMS with the /MACRO qualifier and the name of the macro definitions file:

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

MMS interprets the previous description file as follows:

```
TEST1.EXE : TEST1.OBJ, SYS$LIBRARY:STARLET.OLB
   LINK TEST1, SYS$LIBRARY:STARLET.OLB/LIB
TEST1.OBJ : TEST1.C
   CC/LIST/DEBUG TEST1.C
```

2.3.6.2. Redefining 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 object file and specified target have the same name. Because MMS has a built-in rule for generating .OBJ files from .C files, and because this rule invokes the default macro CFLAGS, you can get the same results with the following simple dependency rule:

```
PROG.OBJ :
```

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

```
CFLAGS = /LIST
```

```
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
   LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
MOD1.OBJ DEPENDS_ON
MOD2.OBJ DEPENDS_ON DEFDIR:DEFS1.H
MOD2.OBJ, MOD3.OBJ DEPENDS_ON DEFDIR:DEFS2.H
```

MMS interprets the description file as follows:

```
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
   LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
MOD1.OBJ DEPENDS_ON MOD1.C
   CC/LIST MOD1.C
MOD2.OBJ DEPENDS_ON MOD2.C, DEFDIR:DEFS1.H, DEFDIR:DEFS2.H
   CC/LIST MOD2.C
MOD3.OBJ DEPENDS_ON MOD3.C, DEFDIR:DEFS2.H
   CC/LIST MOD3.C
```

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

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

MMS interprets the description file as follows:

```
PROG.EXE DEPENDS_ON MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
   LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
MOD1.OBJ DEPENDS_ON MOD1.C
   CC/DEBUG/NOLIST MOD1.C
MOD2.OBJ DEPENDS_ON MOD2.C, DEFDIR:DEFS1.H, DEFDIR:DEFS2.H
   CC/DEBUG/NOLIST MOD2.C
MOD3.OBJ DEPENDS_ON MOD3.C, DEFDIR:DEFS2.H
   CC/DEBUG/NOLIST MOD3.C
```

2.3.7. Redefining Macros in a Description File

Macros can be redefined within the same description file for a series of conditions. To do this, use the /EXTENDED_SYNTAX qualifier at the command line, or select the Extended Syntax option in the DECwindows Build Definitions/Directives Options dialog box. When you exit the DECwindows session, MMS reverts back to the default /NOEXTENDED_SYNTAX state.

For example:

```
.IF PCA
LINKFLAGS=$(LINKFLAGS) /DEBUG
.ENDIF
```

This entry in the description file adds the /DEBUG qualifier to the default macro LINKFLAGS under the condition PCA. If you try to redefine the macro in the file without using the /EXTENDED_SYNTAX qualifier on the command line, MMS displays the message "illegal attempt to redefine macro."

2.4. Using Special Macros

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

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

- User-defined rules
- Macro definitions
- Action lines
- Comments

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

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

More information on the special macros that relate to the Code Management System (CMS) can be found in *Section 4.2, "Using MMS with CMS"*.

Note

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

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

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

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

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

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

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

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

```
PROG.EXE : PRINT.FLG, MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
   COPY NLA0: PRINT.FLG
   ! Make the revision date of PRINT.FLG more current
   PURGE PRINT.FLG
   LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
PRINT.FLG : MOD1.C, MOD2.C, MOD3.C
   ! Print the sources that have changed
   PRINT $(MMS$CHANGED_LIST)
COPY NLA0: PRINT.FLG
! Make the revision date of PRINT.FLG more current
PURGE PRINT.FLG
```

```
LINK/EXEC=PROG MOD1.OBJ, MOD2.OBJ, MOD3.OBJ
PRINT.FLG : MOD1.C, MOD2.C, MOD3.C
! Print the sources that have changed
PRINT $(MMS$CHANGED_LIST)
```

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

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

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

Your directory contains the following entries:

```
$ DIR/DATE=MODIFIED
Directory USER$:[MICHAELS]
MOD1.OBJ;2    2-DEC-2005 13:50
MOD2.OBJ;1    2-DEC-2005 09:22
MOD3.OBJ;2    2-DEC-2005 14:06
PROG.EXE;1    2-DEC-2005 11:47
Total of 4 files
$
```

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

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

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

2.5. Defining Your Own Rules

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

2.5.1. Creating a User-Defined Rule

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

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

.SRC.TAR [!comment]

action line... [!comment]

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

The following description file, NEWLINK.MMS shown in *Example 2.4, "Description File Using a User-Defined Rule"*, contains a user-defined rule that redefines the default MMS rule for linking.

Example 2.4. Description File Using a User-Defined Rule

```
$ TYPE NEWLINK.MMS
!
! User-defined rule
!
.OBJ.EXE
   $(LINK) $(LINKFLAGS) $(MMS$SOURCE_LIST) ❶
!
! Executable images and their sources
!
MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ ❷
!
! Object files and their sources
MAIN.OBJ DEPENDS_ON MAIN.PAS ❸
  SUB1.OBJ DEPENDS_ON SUB1.PAS
```

Key to the example:

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

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

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

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

2.5.2. Using User-Defined Rules

To use the description file NEWLINK (shown in *Example 2.4, "Description File Using a User-Defined Rule"*) to build your software system, you must have the following files in your current directory:

```
$ DIR/DATE=MODIFIED
Directory DISK1:[BUILD]
MAIN.PAS;1      3-JUL-2005 13:48
NEWLINK.MMS;2  14-JUL-2005 13:20
SUB1.PAS;10     3-JUL-2005 13:47
Total of 3 files.
```

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

Key to the example:

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

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

2.6. Using Action Lines

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

You can supply an action line for any source or target line. Action lines override all built-in and user-defined rules in a description file. Action lines are made up of any combination of default macros, special macros, and user-supplied strings.

To keep your description file simple, use built-in rules as much as possible. One user-defined rule can apply to several different target and source lines, so it is still preferable to an action line. However, when the action is so specific that it must be described for that individual case, you must use action lines.

Consider the description file, ACTION_LINES.MMS, shown in *Example 2.5, "Description File Using Action Lines"*.

Example 2.5. Description File Using Action Lines

```
$ TYPE ACTION_LINES.MMS
!
! Executable image and its sources
!
MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ
   $(LINK) $(LINKFLAGS) $(MMS$SOURCE_LIST) ❶
!
! Object files and their sources
!
MAIN.OBJ DEPENDS_ON MAIN.PAS
PASCAL /LIST MAIN.PAS ❷
SUB1.OBJ DEPENDS_ON SUB1.PAS
$(PASCAL) /LIST /MACHINE_CODE $(MMS$SOURCE) ❸
```

Key to the example:

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

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

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

It is better to write a description file with consistent action lines than to mix action lines, as shown in this example. The purpose of this example is to show the different ways to write an action line.

2.6.1. Multiple Action Lines

Sometimes a target requires a series of actions to update it. In this case, you can use more than one action line after a target or source line. Consider the description file, BALANCE.MMS, shown in *Example 2.6, "Description File Using Multiple Action Lines"*.

Example 2.6. Description File Using Multiple Action Lines

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

Key to the example:

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

❷ MMS then runs the program ACCOUNTS.EXE.

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

❹ MMS displays the results of the comparison.

If you use the previous description-file example, the output is as follows:

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

When you run MMS, all action lines and any comments specified on action lines are written to SYS $OUTPUT, or to a file you specify with the MMS /OUTPUT qualifier. The /OUTPUT qualifier is described in the *Chapter 5, "Command Dictionary"*.

2.6.2. $STATUS and $SEVERITY

As each action line completes execution, MMS executes a command in the subprocess to write the value of $STATUS to a mailbox. The parent process can then determine if the action line executed

successfully. The values of $STATUS and $SEVERITY are set when the execution of this internal MMS command succeeds. Consequently, $STATUS and $SEVERITY always indicate success. You cannot test the values of these variables in a description file, although they can be used in the action line itself. You can, however, control the behavior of MMS with the /[NO]IGNORE qualifier because it tells MMS what to do when it encounters warning, error, or fatal errors. See the *Chapter 5, "Command Dictionary"* for more information on severity errors.

2.6.3. MMS$STATUS and MMS$SEVEREST_STATUS

MMS uses the special symbols MMS$STATUS and MMS$SEVEREST_STATUS to record the return status of executed action lines. MMS$STATUS returns the status of the action line last executed. MMS $SEVEREST_STATUS returns the most severe status of all action lines executed. If more than one action line has the same level of severity, MMS$SEVEREST_STATUS returns the value of the action line most recently executed.

Both symbols are set in the parent process that runs MMS and reflect the value of $STATUS returned from the child process. Consequently, the value of these symbols changes constantly with the completion of each action line. Note that neither symbol can be used from within the child process, since the symbols are only passed from the parent to the child process when the child process is created. When MMS exits, MMS$STATUS reflects the status of the last command executed and MMS $SEVEREST_STATUS reflects the most severe status of all commands executed by the child process.

To check the value of these symbols, enter the DCL command SHOW SYMBOL after MMS has finished processing your description file. 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. 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.

2.6.4. Action-Line Prefixes

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

Table 2.1, "MMS Action-Line Prefixes" describes the MMS action-line prefixes.

Table 2.1. MMS Action-Line Prefixes

Prefix	Function
- (Ignore)	Causes MMS to ignore errors generated by the action line on which the prefix appears.
@ (Silent)	Suppresses the writing to the output file of the action line on which the prefix appears. (The output file can be SYS$OUTPUT, or the file specified by the /OUTPUT qualifier.)
? (Action Status)	Causes MMS to process the return status value of the action line as specified by the associated user-defined severity directive.

You cannot override the action-line prefixes from the MMS command line.

An action-line prefix must appear as the first non-blank character on an action line; however, a prefix cannot 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 multiple prefixes on the same action line by typing them next to each

other with no intervening spaces or tabs. The following example shows the use of the Silent and Ignore prefixes on the same action line:

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

MMS also provides directives (described in *Section 2.7, "Using Directives"*) that are similar in function to the prefixes. The difference between the action-line prefixes and the directives with the same functions is that a prefix affects the processing of only one line in the description file, whereas a directive affects the processing of the entire file.

2.6.5. Ignore Prefix (-)

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

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

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

2.6.6. Silent Prefix (@)

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

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

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

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

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

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

2.6.7. Action Status Prefix (?)

The Action Status prefix indicates that the OpenVMS severity associated with the action line must calculated according to the rules defined by the .ACTION_STATUS directive (as described in

Section 2.7.1, ".ACTION_STATUS Directive"). The (?) prefix must be followed by the name of an .ACTION_STATUS directive.

2.6.8. Action-Line Restrictions

Action lines are subject to the following restrictions:

- The maximum length of an uncontinued action line is 251 characters. The maximum length of a continued action line is 1019 characters. If you use Shell as your Command Language Interpreter (CLI), the limits are 131 and 507 characters, respectively, for uncontinued and continued action lines.
- The maximum length of a quoted string of a comment in action lines is 130 characters.
- Quotes embedded in other quotes on action lines might not behave as expected. However, if you assign the inner quote to be a DCL symbol, you can use the DCL symbol within the outer quoted string.
- An action line cannot receive data from SYS$INPUT. For example, an action line cannot contain the DCL command CREATE, and cannot read data from the terminal.
- An action line cannot contain the DCL commands LOGOUT, EXIT, or STOP.
- An action line can spawn a subprocess only by using the $(MMS) reserved macro (see *Section 3.3, "Invoking MMS from a Description File"*). The DCL command SPAWN is not allowed in an action line.
- An action line cannot contain the DCL commands SET VERIFY or SET ON. You can use these commands in a command procedure that you invoke from an action line. If you use SET VERIFY, however, be sure to issue SET NOVERIFY before the command procedure ends.
- An action line cannot contain the DCL command GOTO or labels. You can use GOTO or labels in a command procedure that you invoke from an action line because action lines are executed individually.
- You cannot direct output to TT: from an action line because MMS equates TT: as SYS$INPUT, and this can cause MMS to hang.
- $STATUS and $SEVERITY always indicate success. You cannot test the values of these variables in a description file, although they can be used in the action line itself. The value of these variables is set when the execution of an internal MMS command succeeds. The value of MMS$STATUS and MMS$SEVEREST_STATUS also changes with the completion of each action line executed.
- You cannot use the multiline form of IF - THEN - ELSE - ENDIF from an action line; each action line must be a complete DCL command.

2.7. Using Directives

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

A directive must start in column 1 of a line. You can type a directive in either uppercase or lowercase letters, or a combination of both. *Table 2.2, "MMS Directives"* lists the directives and their functions. Detailed descriptions of the directives are provided in the sections that follow.

Table 2.2. MMS Directives

Directive	Function
.ACTION_STATUS	Directs MMS to apply a user-defined severity value to action lines that do not return a standard OpenVMS status value.
.IGNORE	Causes MMS to ignore all errors generated by all action lines and continue processing the description file.
.IGNORE_ALL	Causes MMS to ignore warnings, errors, and fatal errors that occur during processing of the description file—for both action lines and with the description file itself.
.SILENT	Suppresses the writing of all action lines to the output file (whether to SYS$OUTPUT or to the file specified by the /OUTPUT qualifier).
.DEFAULT	Indicates actions to be performed if MMS built-in rules or user-defined rules do not specify how to update a target.
.SUFFIXES	Appends a list of file types to the suffixes precedence list.
.SUFFIXES_AFTER	Inserts a list of file types in the suffixes precedence list after the specified first file type.
SUFFIXES_BEFORE	Inserts a list of file types in the suffixes precedence list before the specified first file type.
.SUFFIXES_DELETE	Removes a list of file types from 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.
.IF	Causes subsequent lines of a description file to be processed only if the related boolean expression is true.
.ELSIF	Causes subsequent lines of a description file to be processed only if the related boolean expression is true.
.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 only if the macro for the preceding .IF, .IFDEF, or .ELSIF directives are undefined or false.
.ENDIF	Terminates the set of lines in the description file whose processing is controlled by .IF, .IFDEF, .ELSE, or .ELSIF.

2.7.1. .ACTION_STATUS Directive

The .ACTION_STATUS directive enables MMS to deal with action lines that do not return standard OpenVMS status values. This directive defines how the appropriate severity can be determined from a foreign status value and associates a name with this operation.

Prefixing an action line with a question mark (?) indicates that MMS must calculate the severity of the action line according to the rules defined by the matching .ACTION_STATUS directive.

The .ACTION_STATUS directive has the following form:

```
.ACTION_STATUS name [ .MASK m ]
[ .SUCCESS { s1,s2,... | OTHERS } ]
[ .INFORMATION { i1,i2,... | OTHERS } ]
[ .WARNING { w1,w2,... | OTHERS } ]
[ .ERROR { e1,e2,... | OTHERS } ]
[ .FATAL { f1,f2,... | OTHERS } ]
```

The value of name is an alphanumeric character string terminated by a comma (,), space, or tab. Spaces, tabs, or commas can also be used to separate lists of numbers. The name must be unique; you cannot use the same name for more than one .ACTION_STATUS directive.

The values of m, s, i, w, e, and f are decimal, octal (0...), or hexadecimal (%x... or 0x...) numbers representing status values to be associated with the corresponding OpenVMS severity level.

The keyword OTHERS associates all unspecified status values with the corresponding severity. This variable can only be specified for one severity. If it is not specified, all unspecified status values default to the least severe of any undefined severity (or ERROR, if all severity values are defined).

If the directive definition is longer than one line in length, use the line continuation character (-) to continue the definition.

Once defined, the .ACTION_STATUS directive is effective over the entire file. MMS then interprets the status returned from an action line prefixed by ?name, as follows:

1. If MASK is specified, MMS extracts the value from the action line status using the mask value; the extracted value is shifted right to match the first bit in the mask. Otherwise, MMS uses the literal action line status value.

2. MMS determines the appropriate severity to associate with the extracted status value.

3. MMS then continues, treating this severity as if it were a standard OpenVMS status value (identified by $SEVERITY).

The following example shows the use of the .ACTION_STATUS directive:

```
.ACTION_STATUS FOREIGN .MASK %xF0 .SUCCESS 0 .WARNING 1 -
.ERROR %xF .FATAL OTHERS
.FIRST
@ SET MESSAGE/NOFACILITY/NOSEVERITY/NOIDENTITY/NOTEXT
MY_OUTPUT.TXT : MY_INPUT.C
@ DEFINE/USER SYS$INPUT MY_INPUT.C
@ DEFINE/USER SYS$OUTPUT MY_OUTPUT.TXT
?FOREIGN @MY_TOOL
! my_tool returns a value in the least significant 8-bits of $STATUS.
! Success or failure of the operation is indicated by the value in the
! next 8-bits as follows:
```

```
! 0 -> success
! 1 -> warning
! %xF -> error
! all other values are treated as fatal
```

2.7.2. .IGNORE Directive

The .IGNORE directive tells MMS to ignore warnings, errors, and fatal errors that occur during the execution of an action line and 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 VSI Digital Standard Runoff (DSR) to update the target. For example:

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

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

```
$ MMS/DESCRIPTION=BOOK
RUNOFF CHAPTER1
VSI Digital Standard Runoff Version V2.0-014: No errors detected
5 pages written to "USER$:[MICHAELS]CHAPTER1.MEM;1"
RUNOFF CHAPTER2
VSI Digital Standard Runoff Version V2.0-014: No errors detected
16 pages written to "USER$:[MICHAELS]CHAPTER2.MEM;1"
RUNOFF CHAPTER3
%RUNOFF-W-CJL, Can't justify line
on output page 2; on input line 46 of page 1 of file "USER$:[MICHAELS]CH
APTER3.RNO;1"
%RUNOFF-W-CJL, Can't justify line
on output page 2; on input line 52 of page 1 of file "USER$:[MICHAELS]CH
APTER3.RNO;1"
%RUNOFF-W-TFE, Too few end commands
on output page 3; on input line 77 of page 1 of file "USER$:[MICHAELS]CH
APTER3.RNO;1"
%RUNOFF-W-BMS, Bad margin specification: ".lm70
on output page 4; on input line 102 of page 1 of file "USER$:[MICHAELS]C
HAPTER3.RNO;1"
%RUNOFF-W-COR, Can't open required file "TABLE1.RNO"
on output page 5; on input line 154 of page 1 of file "USER$:[MICHAELS]C
HAPTER3.RNO;1"
VSI Digital Standard Runoff Version 2.0-014: 5 diagnostic messages reported
10 pages written to "USER$:[MICHAELS]CHAPTER3.MEM;1"
```

```
RUNOFF CHAPTER4
VSI Digital Standard Runoff Version V2.0-014: No errors detected
13 pages written to "USER$:[MICHAELS]CHAPTER4.MEM;1"
COPY/LOG CHAPTER1.RNO BOOK.MEM
%COPY-S-COPIED, USER$:[MICHAELS]CHAPTER1.MEM;1 copied to USER$:
[MICHAELS]BOOK.ME
M;1 (35 blocks)
APPEND/LOG CHAPTER2.MEM BOOK.MEM
%APPEND-S-APPENDED, USER$:[MICHAELS]CHAPTER2.MEM;1 appended to USER$:
[MICHAELS]B
OOK.MEM;1 (1452 records)
APPEND/LOG CHAPTER3.MEM BOOK.MEM
%APPEND-S-APPENDED, USER$:[MICHAELS]CHAPTER3.MEM;1 appended to USER$:
[MICHAELS]B
OOK.MEM;1 (1508 records)
APPEND/LOG CHAPTER4.MEM BOOK.MEM
%APPEND-S-APPENDED, USER$:[MICHAELS]CHAPTER4.MEM;1 appended to USER$:
[MICHAELS]B
OOK.MEM;1 (621 records)
$
```

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

Note

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

To override the .IGNORE directive for a given MMS build, use the /NOIGNORE, /IGNORE, /IGNORE=WARNING, or /IGNORE=ERROR qualifier on the MMS command line when invoking MMS. (See *Chapter 5, "Command Dictionary"* for more information on the /IGNORE qualifier.)

2.7.3. .IGNORE_ALL Directive

The .IGNORE_ALL directive tells MMS to ignore warnings, errors, and fatal errors that occur during processing of the description file and continue processing the description file. Errors occurring in action lines and with the description file itself will be ignored.

The .IGNORE_ALL directive in the following description file tells MMS to continue processing even though the target "c" is not specified; for example:

```
.ignore_all
target : a,b,c
   show time
a :
   write sys$output "aaa"
b :
   write sys$output "bbb"
```

2.7.4. .SILENT Directive

The .SILENT directive tells MMS to suppress the display of action lines.

Normally, MMS writes action lines to SYS$OUTPUT, or to a file specified by the /OUTPUT qualifier. Action lines are always executed even if they are not displayed, unless you specify the /NOACTION qualifier on the command line. The /OUTPUT and /NOACTION qualifiers are described in the *Chapter 5, "Command Dictionary"*.

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

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

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

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

To override the .SILENT directive for a given MMS build, use the /VERIFY qualifier on the MMS command line when invoking MMS. (The /VERIFY qualifier is described in the *Chapter 5, "Command Dictionary"*.)

2.7.5. .DEFAULT Directive

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

The .DEFAULT directive has the following format:

.DEFAULT action line...

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

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

```
.DEFAULT
   ! Source $(MMS$TARGET) not yet added
TEST.A : TEST.B
TEST.B : TEST1.C TEST2.E TEST3.F
TEST1.C : TEST1.D
   COPY TEST1.D TEST1.C
```

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

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

```
! Source TEST.B not yet added
! Source TEST.A not yet added
$
```

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

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

```
.DEFAULT :
   COPY $(MMS$SOURCE) $(MMS$TARGET)
TEST.BLI : [PROJECT.FILES]TEST.BLI
PROG.BLI : [PROJECT.FILES]PROG.BLI
```

The sources in this description file exist in a common directory for the project. Because these dependency rules have no action lines and no built-in or userdefined rules apply, MMS executes the action line specified by .DEFAULT and copies the required files into your directory. (The MMS $SOURCE and MMS$TARGET special macros are described in *Section 2.4, "Using Special Macros"*.)

Note

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

2.7.6. .SUFFIXES, .SUFFIXES_AFTER, .SUFFIXES_BEFORE, and .SUFFIXES_DELETE Directives

The suffixes directives allow you to redefine the suffixes precedence list so you can reorder the list of file types, add new file types, or disable recognition of all file types.

MMS uses the suffixes precedence list to determine the order in which it looks for sources and targets when applying built-in rules. MMS also uses this list to determine which built-in rule will update the specified target. *Section 2.2, "Using Built-In Rules"* contains a detailed description of how the suffixes precedence list and MMS built-in rules work together. *Table C.4, "Suffixes-Precedence List"* lists the suffixes in order of their precedence.

The suffixes directives have the following format, where file-types-list represents a list of file extensions, in order of precedence:

```
.SUFFIXES [file-types-list]
.SUFFIXES_AFTER first-file-type [file-types-list]
.SUFFIXES_BEFORE first-file-type [file-types-list]
.SUFFIXES_DELETE [file-types-list]
```

- The .SUFFIXES directive appends the list of file types to the end of the suffixes precedence list. If a list is not specified, the entire suffixes precedence list is deleted.
- The .SUFFIXES_AFTER and .SUFFIXES_BEFORE directives insert the specified file types into the suffixes precedence list either after or before the suffix value specified by first-file-type. If the first-file-type is not in the suffixes precedence list, an informational message is displayed, and the list of file types is appended to the end of the suffixes precedence list.
- The .SUFFIXES_DELETE directive deletes the specified file types from the suffixes precedence list. If a list is not specified, the entire suffixes precedence list is deleted.

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

2.7.6.1. Adding a New File Extension to the Suffixes List

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

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

2.7.6.2. Using the .SUFFIXES Directive in a Description File

To add a new file type to your description file, use the directive .SUFFIXES. It clears the default suffixes list and allows you to write a new list in the description file. *Table 2.2, "MMS Directives"* lists the directives and their functions.

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

- Executable files
- Different types of library files
- Object files
- Source code files
- Include files

Consider the description file NEW_SUFFIX.MMS shown in *Example 2.7, "Description File Using the .SUFFIXES Directive"*.

Example 2.7. Description File Using the .SUFFIXES Directive

```
$ TYPE NEW_SUFFIX.MMS
!
! Set a new suffixes list
!
.SUFFIXES ❶
.SUFFIXES .EXE .OBJ .NEW .FOR ❷
!
! User-defined rules
!
.NEW.OBJ ❸
@NEW_COMPILER $(MMS$SOURCE) $(MMS$TARGET)
.OBJ.EXE ❹
$(LINK) $(LINKFLAGS) $(MMS$SOURCE_LIST)
!
! The executable and its sources
```

```
!
MAIN.EXE DEPENDS_ON MAIN.OBJ, SUB1.OBJ
!
! Object files and their sources
!
MAIN.OBJ DEPENDS_ON MAIN.FOR ❺
SUB1.OBJ DEPENDS_ON SUB1.NEW ❻
```

Key to the example:

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

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

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

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

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

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

2.7.6.3. Building a System with a New File Extension

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

```
$ DIR/DATE=MODIFIED
Directory DISK1:[BUILD]
MAIN.FOR;2          17-JUL-2005 14:27
NEW_COMPILER.COM;5  17-JUL-2005 14:27
NEW_SUFFIX.MMS;1    17-JUL-2005 14:27
SUB1.NEW;1          17-JUL-2005 14:27
Total of 4 files
$ MMS/DESCRIPTION=NEW_SUFFIX
   FORTRAN /NOLIST/OBJECT=MAIN MAIN.FOR
   @NEW_COMPILER SUB1.NEW SUB1.OBJ ❶
   LINK /TRACE/NOMAP/EXEC=MAIN MAIN.OBJ SUB1.OBJ
```

Key to the example:

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

Order of Suffixes

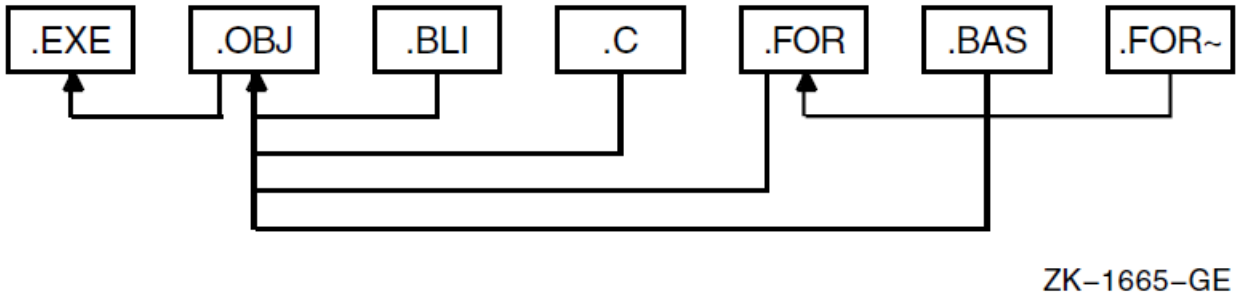
The order of suffixes changes according to their use:

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

2.7.6.4. Using the .SUFFIXES Directive with CMS Files

If you add a rule in your description file that directs MMS to build a .FOR file by fetching it from a CMS library, the relationship between the rules and the file types might be as shown in *Figure 2.2, "CMS Rules"*.

Figure 2.2. CMS Rules



Section 4.2, "Using MMS with CMS" explains how to specify CMS elements in description files. (The tilde (~) signifies a file in a CMS library.) When MMS considers .FOR as a possible target, it discovers that a rule exists for building .FOR files from .FOR~ files. Therefore, it looks for a file named MOD3.FOR in the CMS library. If one exists, it applies the known rule to update the .FOR target; if it cannot find such a file, it continues searching for a file to use.

If MMS does locate MOD3.FOR in the library, it uses this file to create all the necessary sources that finally result in an updated MOD3.EXE, the original target. The simple dependency "MOD3.EXE :" could result in the following sequence of actions:

```
MOD3.FOR DEPENDS_ON MOD3.FOR~
   CMS FETCH MOD3.FOR
MOD3.OBJ DEPENDS_ON MOD3.FOR
   FORTRAN/OBJ=MOD3 MOD3.FOR
MOD3.EXE DEPENDS_ON MOD3.OBJ
   LINK/EXEC=MOD3 MOD3.OBJ
```

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

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

2.7.7. .INCLUDE Directive

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

The .INCLUDE directive has the following format:

`.INCLUDE` filespec

A filespec is an OpenVMS file specification or logical name that identifies the include 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.

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

2.7.8. .FIRST Directive

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

The .FIRST directive has the following format:

.FIRST action line...

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

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

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

```
.FIRST
OPEN/WRITE MSGTEXT MSGTEXT.TXT
WRITE MSGTEXT "Build of $(MMS$TARGET) now beginning"
CLOSE MSGTEXT
MAIL MSGTEXT.TXT ANDERSON -
/SUBJECT="Report from MMS"
BOOK.MEM : CHAPTER1.MEM, CHAPTER2.MEM, CHAPTER3.MEM, CHAPTER4.MEM
COPY/LOG CHAPTER1.MEM BOOK.MEM
APPEND/LOG CHAPTER2.MEM BOOK.MEM
APPEND/LOG CHAPTER3.MEM BOOK.MEM
APPEND/LOG CHAPTER4.MEM BOOK.MEM
CHAPTER1.MEM : CHAPTER1.RNO
RUNOFF CHAPTER1
CHAPTER2.MEM : CHAPTER2.RNO
RUNOFF CHAPTER2
CHAPTER3.MEM : CHAPTER3.RNO
RUNOFF CHAPTER3
CHAPTER4.MEM : CHAPTER4.RNO
RUNOFF CHAPTER4
```

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

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

```
COPY CHAPTER1.MEM BOOK.MEM
APPEND CHAPTER2.MEM BOOK.MEM
APPEND CHAPTER3.MEM BOOK.MEM
APPEND CHAPTER4.MEM BOOK.MEM
```

2.7.9. .LAST Directive

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

The .LAST directive has the following format:

.LAST action line...

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

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 ADESC.MMS:

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

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

2.7.10. .IF, .IFDEF, .ELSE, .ELSIF, and .ENDIF Directives

These directives instruct MMS to process certain lines in your description file.

The .IF directive has the following format:

.IF boolean-expression [description-file-line]… {.ELSIF boolean-expression [description-file-line]… } [.ELSE[description-file-line]…] **.ENDIF**

The value of description-file-line is zero or more lines in a description file, which can include additional .IF directives. The .IF directive can be followed by any number of .ELSIF directives and up to one .ELSE directive. The .IF directive must always be accompanied by a matching .ENDIF directive.

MMS evaluates the boolean expression specified with the .IF directive. If the expression is true, MMS processes all lines in the description file between the corresponding .IF and .ELSIF, .ELSE, or .ENDIF directive. If MMS detects an .ELSIF or .ELSE directive, it ignores all lines up to to the corresponding .ENDIF directive. If the expression is false, MMS ignores all lines up to an .ELSE, .ENDIF, or .ELSIF directive whose associated boolean-expression is true.

For example, in the following statement:

```
.IF boolexp-1
   line-1
.ELSIF boolexp-2
   line-2
.ELSE
   line-3
.ENDIF
```

- Ifboolexp-1 is true, then line-1 is processed. line-2 and line-3 are ignored.
- Ifboolexp-1 is false and boolexp-2 is true, then line-2 is processed. line-1 and line-3 are ignored.
- Ifboolexp-1 and boolexp-2 are both false, then line-3 is processed. line-1 and line-2 are ignored.

The format for boolean expressions specified with .IF and .ELSIF directives is as follows:

```
boolean-expression :== [ .NOT ] boolean-operation |
                       [ .NOT ] boolean-operation boolean-operator
                                boolean-expression
boolean-operation :== ( boolean-expression ) |
                       word |
                       word comparison-operator word
word :== null | any sequence of characters, terminated by space,
                    and not starting with '.', '(' or ')'.
boolean-operator :== .AND | .OR
comparison-operator :== .EQ | .NE | .GE | .LE | .GT | .LT
```

As shown above, the operands in a boolean expression can take one of the following forms:

```
.IF word-0
.IF word-1 .EQ word-2
```

With the first form, MMS checks that word-0 is a macro defined with a nonnull value. If the macro is defined, the expression is true; if it is not defined, the expression is false. With this form, the directive .IFDEF can be used in place of .IF when compatibility with earlier versions of MMS is necessary.

With the second form, MMS performs the requested comparison between word-1 and word-2 to determine the expression value. Note that the comparison operation is case-sensitive. When using macros and functions in expressions of this form, you must reference the macro or function in the standard way.

For example, a statement to check that the macro FRUIT is defined to be BANANAS could be expressed as follows:

```
.IF $(FRUIT) .EQ BANANAS
```

If you need to compare words that start with a period (.) or parentheses (( )) or text containing layout characters, enclose the words or text in quotation marks ("") on both sides of the comparison operator. For example, a statement to check that the macro FILETYPE is defined as .MMS, and that the macro VERSION is not defined as 'Version 3.2', could be expressed as follows:

```
.IF "$(FILETYPE)" .EQ ".MMS" .AND "$(VERSION)" .NE "Version 3.2"
```

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

Consider the following description file, CPROG.MMS:

```
A.EXE : A.OBJ
   LINK A.OBJ
.IF VAX
A.OBJ : A.C
   CC/VAXC A
.ENDIF
.IF ALPHA
A.OBJ : A.C
   CC/DECC A
.ENDIF
```

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

```
$ MMS/DESCRIPTION=CPROG/MACRO="VAX=yes"
CC/VAXC A
LINK A.OBJ
$
```

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

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

```
A.EXE : A.OBJ
   LINK A.OBJ
.IF VAX
.IF $(CURRENT) .EQ VAXC
A.OBJ : A.C
   CC/VAXC A
.ENDIF
.ELSE
A.OBJ : A.C
```

```
   CC/DECC A
.ENDIF
```

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

```
$ MMS/DESCRIPTION=CPROG/MACRO=("VAX=yes",CURRENT=VAXC)
CC/VAXC A
LINK A.OBJ
$
```

2.8. Generating the MMS Description File Automatically

The description file contains definitions that describe to MMS all the components of a system build: the source files, the compiler commands used, the order in which to link, and the libraries to use. The description file's primary function is to describe dependencies.

The ability to generate an MMS description file from input sources requires that the input sources be scanned for dependencies. The appropriate target for the source, the source, the dependencies, and an action line is written to the description file for each input source.

The following languages are supported by the MMS Description File Generator:

VSI BASIC for OpenVMS
VSI BLISS-32 for OpenVMS
VSI C for OpenVMS
VSI C++ for OpenVMS
VSI COBOL for OpenVMS
VSI DECforms for OpenVMS
VSI Fortran for OpenVMS
VSI Pascal for OpenVMS
Oracle CDD/Repository
VAX Rdb/ELN
VAX SQL

You can generate an MMS description file from input sources in DECwindows MMS, or with an MMS command-line qualifier.

2.8.1. Using the DECwindows User Interface

The DECwindows UI consists of the following:

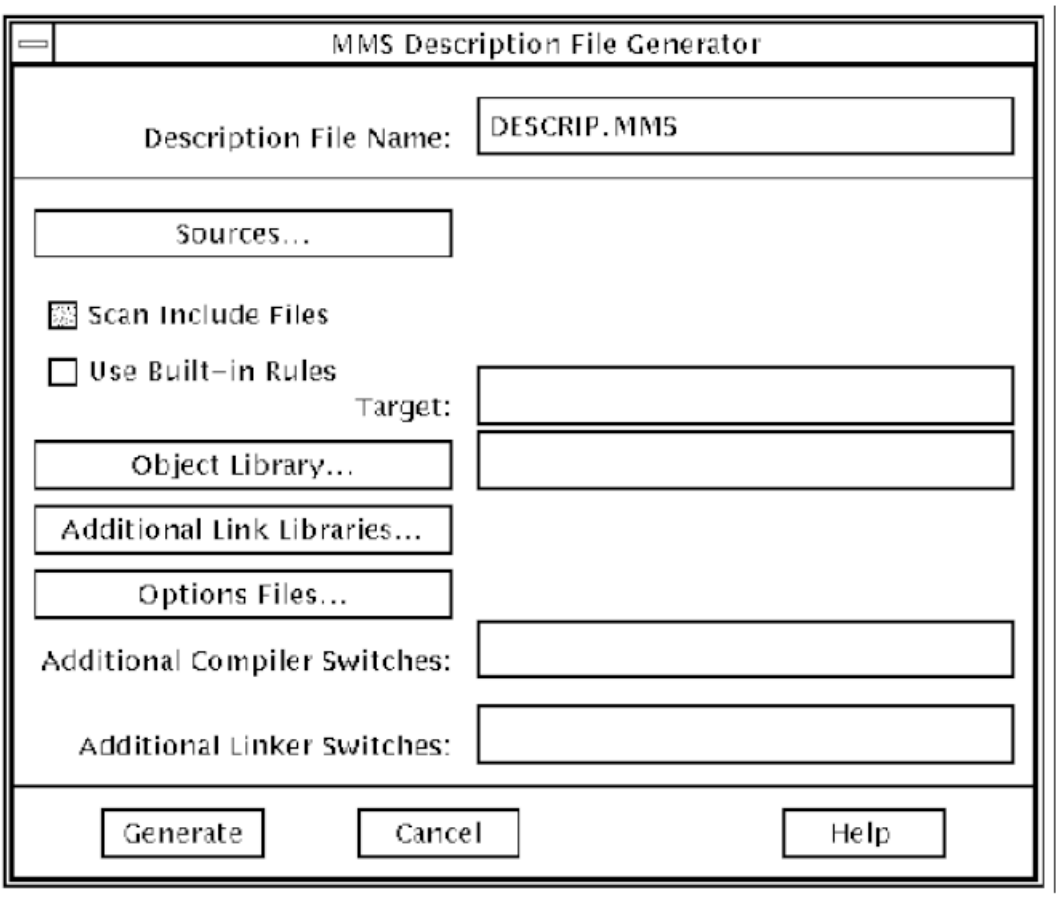
- MMS Description File Generator dialog box
- MMS Sources dialog box

When you click on the Create MMS file... button in the main window of the MMS GUI (shown in *Figure 1.3, "MMS Main Window"*, MMS opens a dialog box for specifying the input to the automatic description file generator.

2.8.1.1. MMS Description File Generator Dialog Box

Figure 2.3, "MMS Description File Generator Dialog Box" illustrates the MMS Description File Generator dialog box.

Figure 2.3. MMS Description File Generator Dialog Box



File Name and Options Section

Click on the following buttons and use the following text fields to specify input files and libraries for any subsequent build:

- **Description File Name**—Use this field to specify a name for the generated (output) description file. The file type of (MMS) is assumed if none is specified.

- **Sources...**—Click on this button to display a dialog box for specifying the input sources. Input sources are the sources to scan for dependencies for the generated description file (see *Figure 2.4, "MMS Sources Dialog Box"*). If no sources are indicated when you click on the Generate button, MMS generates an error message box and no description file is generated.

- **Scan Include Files**—Click on this toggle button to indicate whether C include files are scanned for dependencies.

- **Use Built-in Rules**—Click on this toggle button to indicate whether the MMS built-in compilation actions are used. This causes MMS only to include dependency lines in the generated description file.

- **Target:—**Use this field to specify the name for the build target.

- **Object Library...—**Click on this button to display a file selection dialog box for specifying the object library to be included in the LINK command in the description file. MMS inserts object files from the compilations into this object library. If you invoke the DECwindows version of MMS, the value for the object library defined in the current context is set as the default.

  If you indicate no object library when you click on the Generate button, MMS uses the default library MMS$OLB.OLB.

- **Additional Link Libraries...—**Click on this button to display a selection dialog box similar to the Input Source List dialog box, for specifying additional object libraries included in the LINK command in the description file. If you invoke the DECwindows version of MMS, the value for additional link libraries defined in the current context is a default selection.

- **Options Files...—**Click on this button to display a selection dialog box similar to the Input Source List dialog box, for specifying user-written options files included in the LINK command in the description file. If you invoke the DECwindows version of MMS, the value for options files defined in the current context is a default selection.

- **Additional Compiler Switches:—**Use this field to indicate any additional compilation switches that should be added to all the compile commands in the description file. You must enter information into these fields exactly as would be concatenated to the end of the command line; that is, they should include the (/). The description file generator does not check to ensure that a proper command is created.

- **Additional Linker Switches:—**Use this field to indicate any additional switches that should be added to the LINK command in the description file. You must enter information into these fields exactly as would be concatenated to the end of the command line; that is, they should include the (/). The description file generator does not check to ensure that a proper command is created.

Navigation Buttons

Click on the navigation buttons to generate the description file, cancel your selections, or get online Help. The navigation buttons are as follows:

- **Generate—**Click on this button to indicate that the description file should be generated; and subsequently, you want to exit from the dialog box.

  When the description file is generated, MMS writes the file to the specified output file. If none is specified, the default output file is DESCRIP.MMS.

  If you invoke the description file generator from the MMS DECwindows interface, MMS displays the file in the MMS description file area of the MMS main window. MMS displays any diagnostics in the Build log area of the MMS main window.

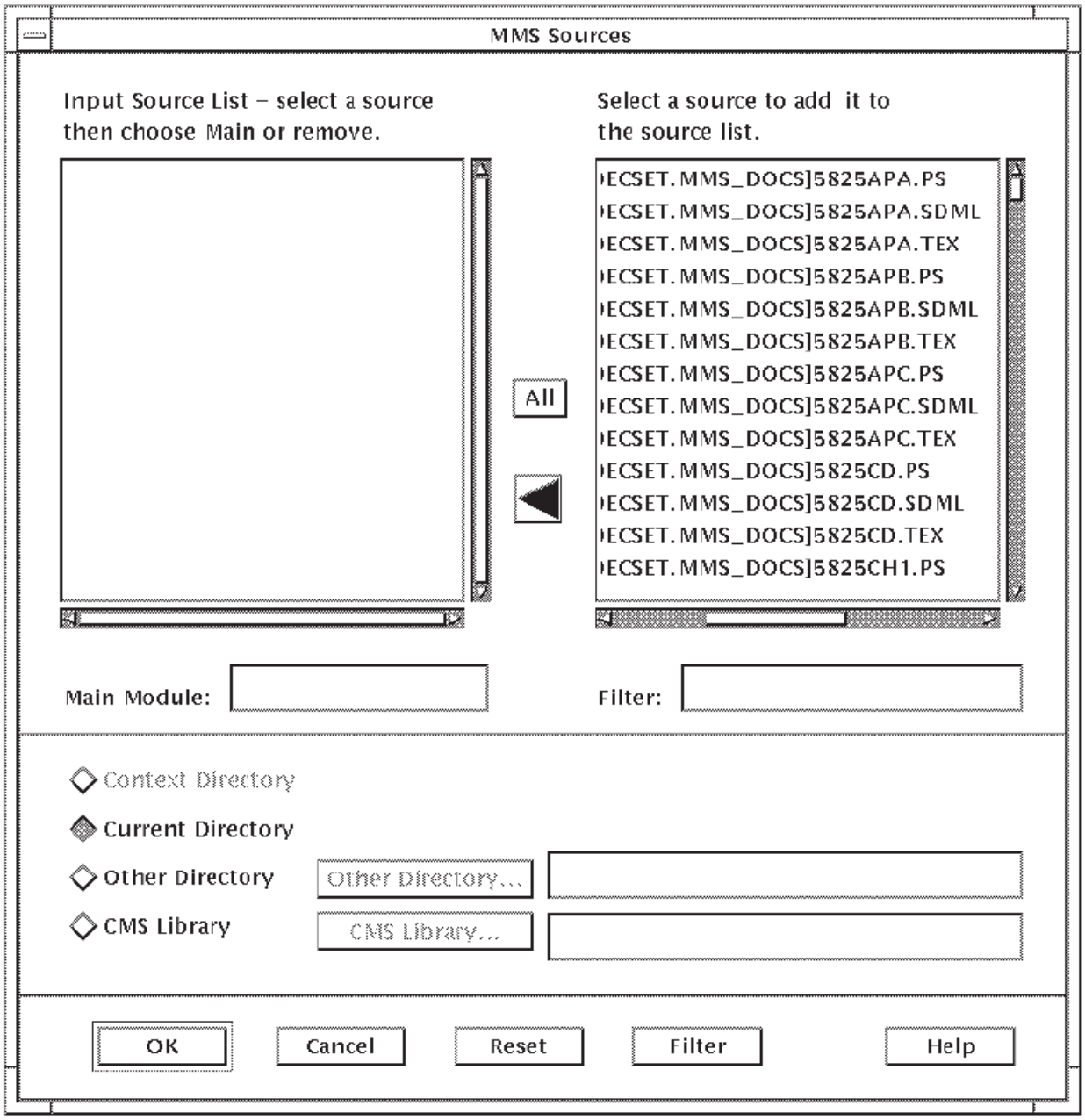
- **Cancel—**Click on this button to quit the dialog box without proceeding with the description file generation.

- **Help—**Click on this button to bring up the textual explanation of the dialog box in a window.

2.8.1.2. MMS Sources Dialog Box

Figure 2.4, "MMS Sources Dialog Box" illustrates the MMS Sources dialog box.

Figure 2.4. MMS Sources Dialog Box



The MMS Sources dialog box includes the following working areas: the Input Source List box, the Selection List box, and the Filter field.

Input Source List Box

Use this scrollable list box to indicate the input sources that are scanned when generating the description file. Initially blank, the top-most source specifies the source for the target of the complete build. You can remove files from the list or move them up in the list by using the right arrow button or the MAIN button. These buttons are displayed when you click on a file name.

Selection List Box

Use this scrollable list box to indicate the possible source files to choose from to insert into the Input Source List box. The Selection List box lists all the files in the current default directory. You have the

option of picking a different directory to list the files from, or indicating that files should be listed from a CMS library (by clicking the corresponding radio button).

As you scroll through the Selection List box, any file that is double-clicked is inserted into the Input Source List box. You can also click the left arrow button or the All button to have all the sources listed moved to the Input Source List box.

All Button, Main Button, and Arrow Buttons

When you invoke the dialog box, the All button and a left arrow are visible. If you click on a file name in the Selection List box, you can click the left arrow to add the file name to the Input Source List box. You can move a file name without the left arrow if you double click on the file name. All the file names in the Selection List box can be moved to the Input Source List box by clicking on the All button.

If you select a file name in the Input Source List box, the All button and the left arrow become invisible, and two new buttons become visible – a Main button and a right arrow. The Main button moves the selected file name into the Main Module field and to the top of the Input Source List box. The right arrow removes the selected file name from the list.

Filter Field

Use the Filter field to specify a mask that filters which file names are listed.

Navigation Buttons

Click on the navigation buttons to generate the save, reset, filter, or cancel your selections, or get online Help. The navigation buttons are as follows:

- **OK—**Click on this button to indicate that the input source file list is complete and you want to exit out of the dialog box.
- **Cancel—**Click on this button to exit out of the dialog box without a completed input source file list.
- **Reset—**Click on this button to set the dialog box back to its initial state.
- **Filter—**Click on this button to apply the filter mask defined in the Filter field to the specified directory (or library). The appropriate source files are listed in the Selection list box.
- **Help—**Click on this button to bring up the textual explanation of the dialog box in a window.

2.8.2. Using the Automatic Description File Generator

The Generator allows you to generate an MMS description file from source files. You use the resulting MMS description file to build a complete application, or any of the several building blocks that comprise the complete application, such as the ones listed in *Table 2.3, "Building Blocks"*:

Table 2.3. Building Blocks

Building Block	Result
DECforms Form	.FORM
DECforms Escape Image	.EXE
TDMS Request Library	.RLB
Application Image	.EXE
Message Image	.EXE

For example, an MMS description file for an application image will contain MMS commands to build the .EXE.

MMS takes as a parameter a list of any of the above building block source files. MMS searches through the supplied source files for key statements indicating dependencies on CDD elements or other source files. For example, the COBOL COPY xxx FROM DICTIONARY statement indicates that the object module is dependent on the listed record definition. The utility then searches through the source file associated with that definition for any additional dependencies.

This search through related files for dependencies continues until the chain is complete and an MMS description file can be generated to reflect all the dependencies of the building block supplied.

2.8.2.1. Application Dependencies

Although several files refer to a number of other objects, only some of them directly affect the module currently being read. For example, although a module might refer to a DECforms form, a modification to that form does not require the module to be rebuilt, and vice-versa.

In contrast, a change to a record definition, used in a module in the application, does require a rebuild of the application. Rather than making a dependency between the image file and all those objects in the image, the utility makes the dependency more straightforward: the application only depends on its own object. This object, in turn, depends on all the called objects and CDD record definitions. Each CDD record definition, in turn, depends on its own source definition file.

2.8.2.2. Trigger Summary Dependencies

Table 2.4, "Dependencies in Supported Language File Types" lists the file types and languages supported by the automatic Description File Generator.

Table 2.4. Dependencies in Supported Language File Types

File Type	Language
.BAS	BASIC
.SBA	BASIC with embedded SQL
.RBA	BASIC with embedded RDML
.B32	Bliss
.BLI	Bliss
.C	C
.SC	C with embedded SQL
.RC	C with embedded RDML
.CXX	C++
.COB	COBOL
.SCO	COBOL with embedded SQL
.RCO	COBOL with embedded RDML
.IFDL	DECforms
.CDO	DECforms
.DDL	DECforms
.LDF	DECforms
.RDF	DECforms

File Type	Language
.F	Fortran
.F77	Fortran
.F90	Fortran
.FOR	Fortran
.SFO	Fortran with embedded SQL
.RFO	Fortran with embedded RDML
.MAR	Macro
.PAS	Pascal
.SPA	Pascal with embedded SQL
.RPA	Pascal with embedded RDML
.RGA	VAX Rally
.RGC	VAX Rally
.RGE	VAX Rally
.RGH	VAX Rally
.RGM	VAX Rally
.SQLMOD	SQL Module Language

Table 2.5, "Dependencies in Supported Language Source Files" describes the definition file phrases that trigger dependency creation and the resulting dependencies in the supported language source files. In general, MMS will trigger properly for both singular and plural forms of allowable syntax, although only the singular form is listed in the table.

Table 2.5, "Dependencies in Supported Language Source Files" lists the trigger phrases for each supported language. Although some trigger phrases do not relate to the previous list, they are allowed in the syntax.

Table 2.5. Dependencies in Supported Language Source Files

Language	Trigger Phrase
ALL (SQL)	EXEC SQL filespec EXEC SQL INCLUDE SQLCA EXEC SQL INCLUDE SQLDA EXEC SQL INCLUDE filespec EXEC SQL INCLUDE ’filespec’ EXEC SQL INCLUDE "filespec" EXEC SQL INCLUDE FROM DICTIONARY CDO^ EXEC SQL INCLUDE FROM DICTIONARY ’CDO^’ EXEC SQL INCLUDE FROM DICTIONARY "CDO^"
BLISS	LIBRARY ’filespec’

Language	Trigger Phrase
	REQUIRE ’filespec’ ROUTINE
COBOL	COPY CDO^ FROM COPY "txt" IN "library" COPY "filespec" CALL ".OBJ" PROGRAM-ID "procedure"
Fortran	DICTIONARY INCLUDE ’library txt’ INCLUDE ’filespec’ CALL EXTERNAL SUBROUTINE, FUNCTION, or PROGRAM procedure
BASIC	%INCLUDE %FROM %CDD "CDO^" %INCLUDE "txt" %FROM %LIBRARY "library" %INCLUDE "filespec" CALL EXTERNAL SUB EXTERNAL datatype FUNCTION SUB, FUNCTION, or PROGRAM procedure
Pascal	%DICTIONARY CDO^ %INCLUDE "library(txt)" %INCLUDE "filespec" {FUNCTION |PROCEDURE} {EXTERN|EXTERNAL|FORTRAN} ENVIRONMENT (’filespec’) INHERIT (’filespec’)
C, C++	#dictionary "CDO^" #include filespec

Language	**Trigger Phrase**
	#include "filespec" #include <filespec> procedure (...); procedure(...) {
SQLMOD	RECORD FROM "CDO^" END RECORD PROCEDURE procedure
IFDL	COPY ifdlspec COPY "ifdlspec" COPY text FROM "library" COPY record FROM DICTIONARY CALL
LDF	%INCLUDE REQUEST IS
RDF	%INCLUDE RECORDS ARE FORMS ARE
CDO	DEFINE RECORD field reference
DDL	COPY FROM

Chapter 3. Advanced Description File Techniques

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

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

3.1. Using Double-Colon Dependencies

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

```
MOD2.OBJ, MOD3.OBJ : DEFS1.DEF
MOD2.OBJ : DEFS2.DEF
   PASCAL MOD2
```

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

In contrast, the following construction not valid:

```
MOD2.OBJ, MOD3.OBJ : DEFS1.DEF
   PRINT DEFS1.DEF
MOD2.OBJ : DEFS2.DEF
   PASCAL MOD2
```

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

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

```
MOD2.OBJ, MOD3.OBJ :: DEFS1.DEF ! If at least one source is
   PRINT DEFS1.DEF      ! newer than targets, print DEFS1.DEF
MOD2.OBJ :: DEFS2.DEF   ! If MOD2.PAS or DEFS2.DEF is newer than
   PASCAL MOD2          ! MOD2.OBJ, compile MOD2.PAS
```

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

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

3.2. Maintaining a Library of Object Fi

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

```
UTIL.OLB :: MOD1.OBJ
   LIBR UTIL.OLB MOD1.OBJ
UTIL.OLB :: MOD2.OBJ
   LIBR UTIL.OLB MOD2.OBJ
UTIL.OLB :: MOD3.OBJ
   LIBR UTIL.OLB MOD3.OBJ
```

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

```
UTIL(MOD1) : MOD1.OBJ
   LIBR UTIL.OLB MOD1.OBJ
UTIL(MOD2) : MOD2.OBJ
   LIBR UTIL.OLB MOD2.OBJ
UTIL(MOD3) : MOD3.OBJ
   LIBR UTIL.OLB MOD3.OBJ
```

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

3.3. Invoking MMS from a Description File

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

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

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

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

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

3.3.1. Using the $(MMS) Reserved Macro

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

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

As MMS processes the description file, it executes any action line that contains the reserved macro $(MMS), even if you specified the /NOACTION qualifier on the command line. (/NOACTION suppresses the execution of action lines and is described in the *Chapter 5, "Command Dictionary"*.) Thus, the MMS subprocess is created, but no other actions are performed.

3.3.2. Process Quotas for MMS Subprocesses

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

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

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

For example, the following description file includes a DCL WAIT statement:

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

3.3.3. Process Quotas for Using MMS

To invoke MMS as a top-level process, you need the minimum process quotas listed in the *Module Management System for OpenVMS Release Notes*.

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

3.3.4. MMS Reserved Macros

MMS includes other reserved macros, which you can use when you invoke MMS as a subprocess. These qualifiers pass the following information to the subprocess:

- $(MMSQUALIFIERS) passes the command-line qualifiers.
- $(MMSTARGETS) passes the targets from the command line.
- $(MMSDESCRIPTION_FILE) passes the complete file specification of the original description file being processed.
- $(MMSARCH_NAME) passes the hardware architecture name (Alpha, IA64, or VAX).
- $(MMSALPHA) if the architecture name is ALPHA, this macro passes the value of ALPHA. Otherwise, it is undefined.
- $(MMSIA64) if the architecture name is IA64, this macro passes the value of IA64. Otherwise, it is undefined.
- $(MMSVAX) if the architecture name is VAX, this macro passes the value of VAX. Otherwise, it is undefined.

These macros, along with $(MMS), are reserved; you cannot redefine them.

Note

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 follows:

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

If you do not use the $(MMSQUALIFIERS) macro, MMS uses the default qualifiers. A list of the default qualifiers and complete descriptions of all MMS qualifiers are contained in the *Chapter 5, "Command Dictionary"*.

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

```
ALL.EXE : A.OBJ, B.OBJ
   LINK/EXEC=ALL A,B
A.OBJ :
   $(MMS) $(MMSQUALIFIERS) /DESCRIPTION=A A.OBJ
B.OBJ :
   $(MMS) $(MMSQUALIFIERS) /DESCRIPTION=B B.OBJ
```

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

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

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

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

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

3.4. Invoking MMS from a Command Procedure

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

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

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

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

Command Procedures and User-Defined Macros

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

You can create a user-defined macro for a CLI symbol. With the /OVERRIDE qualifier, you can instruct MMS to use the value of the user-defined macro in your command procedure over the default value of the CLI symbol. Consider the command procedure DEBUG_VERSION.COM, and the MMS description file SYSTEM1.MMS, shown in *Example 3.1, "Invoking MMS from a Command Procedure"*.

Example 3.1. Invoking MMS from a Command Procedure

```
$ TYPE DEBUG_VERSION.COM
$ ! ------------------------------------------------------
$ !
$ ! Command procedure using the /OVERRIDE qualifier with MMS
$ !
$ ! Create the user defined macros we want
$ !
$ PFLAGS = "/LIST/NOOPTIM/DEBUG" ❶
$ LINKFLAGS = "/MAP/DEBUG" ❶
$ !
$ ! Invoke MMS and direct it to use our macros
$ !
$ MMS /DESCRIPTION=SYSTEM1 /OVERRIDE ❷
$ !
$ ! ------------------------------------------------------
$ !
$ TYPE SYSTEM1.MMS ❸
MAIN.EXE DEPENDS_ON MAIN.OBJ
MAIN.OBJ DEPENDS_ON MAIN.PAS
```

Key to the example:

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

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

❸ The description file describes the dependencies.

This example produces a software system very different from the one MMS would normally have built. The user-defined macro definitions in the command procedure appear in the MMS output exactly where the default macro actions would have appeared.

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

```
$ DIR/DATE=MODIFIED
Directory DISK1:[BUILD]
MAIN.PAS;3            18-JUL-2005 16:35
DEBUG_VERSION.COM;2   18-JUL-2005 16:35
SYSTEM1.MMS;1         18-JUL-2005 16:35
Total of 3 files.

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

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

3.5. Invoking a Command Procedure from a Description File

You can invoke DCL command procedures from your description file. *Example 3.2, "Invoking a Command Procedure from a Description File"* describes a command procedure that loops until a

given file becomes available. As the example shows, you can invoke that command procedure, called GETFILE.COM, from your description file.

Example 3.2. Invoking a Command Procedure from a Description File

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

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

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

Note

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

3.6. Changing System Build Options

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

The command procedure CHANGE_OPTIONS.COM, shown in *Example 3.3, "Command Procedure to Change Build Options"*, uses the DCL INQUIRE command to change compiling and linking options for a system build.

Example 3.3. Command Procedure to Change Build Options

```
$ TYPE CHANGE_OPTIONS.COM
$ !
$ ! Command procedure to vary build options for MMS
$ !
$ ! Ask for compilation and linking options
$ !
$ INQUIRE PFLAGS "Enter PASCAL compilations options"
$ INQUIRE LINKFLAGS "Enter link options"
```

```
$ !
$ ! Invoke MMS and direct it to use new default macros
$ !
$ MMS /DESCRIPTION=SYSTEM1 /OVERRIDE
$ !
$ TYPE SYSTEM1.MMS
MAIN.EXE DEPENDS_ON MAIN.OBJ
MAIN.OBJ DEPENDS_ON MAIN.PAS
Enter PASCAL compilations options: /LIST/NOOPTIM/DEBUG ❶
Enter link options: /MAP/DEBUG ❶
PASCAL /LIST/NOOPTIM/DEBUG MAIN.PAS ❷
LINK /MAP/DEBUG MAIN.OBJ ❷
$ EDIT MAIN.PAS ❸
$ @CHANGE_OPTIONS ❹
Enter PASCAL compilations options: /NODEBUG/NOLIST ❶
Enter link options: /NOMAP/NODEBUG ❶
PASCAL /NODEBUG/NOLIST MAIN.PAS ❷
LINK /NOMAP/NODEBUG MAIN.OBJ ❷
```

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

❷ The system is built using the options you supplied.

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

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

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

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

```
$ DIR/DATE=MODIFIED
Directory DISK1:[BUILD]
CHANGE_OPTIONS.COM;1
                21-JUL-2005 15:51
MAIN.PAS;3      18-JUL-2005 16:35
SYSTEM1.MMS;1   18-JUL-2005 16:35
Total of 3 files.
$ @CHANGE_OPTIONS
```

3.7. Gathering Statistics

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

3.7.1. Finding Missing Sources

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

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

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

3.7.2. Creating a Checkpoint File

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

```
! Suffixes list with .PNT in the first position.
.SUFFIXES
.SUFFIXES .PNT .EXE .OBJ .C .C~
! User-defined rule to build .EXE files from .PNT files.
.EXE.PNT :
   IF ""F$SEARCH("CHECK.PNT")'" .EQS. "" THEN -
      COPY NL: CHECK.PNT
   OPEN/APPEND CHECK CHECK.PNT
   WRITE CHECK "Completed build of $(MMS$SOURCE) at "f$time()'"
   CLOSE CHECK
MAIN_TARGET : TEST1.PNT, TEST2.PNT, TEST3.PNT
   MAIL CHECK.PNT MICHAELS -
   /SUBJECT="Build summary of $(MMS$TARGET_NAME) ending at "f$time()'"
   DELETE CHECK.PNT;
```

Note

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

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

```
CC /NOLIST TEST1.C
LINK /TRACE TEST1.OBJ
IF ""F$SEARCH("CHECK.PNT")'" .EQS. "" THEN COPY NL: CHECK.PNT
OPEN/APPEND CHECK CHECK.PNT
WRITE CHECK "Completed build of TEST1.EXE at "f$time()'"
CLOSE CHECK
CC /NOLIST TEST2.C
LINK /TRACE TEST2.OBJ
IF ""F$SEARCH("CHECK.PNT")'" .EQS. "" THEN COPY NL: CHECK.PNT
OPEN/APPEND CHECK CHECK.PNT
WRITE CHECK "Completed build of TEST2.EXE at "f$time()'"
CLOSE CHECK
CC /NOLIST TEST3.C
LINK /TRACE TEST3.OBJ
IF ""F$SEARCH("CHECK.PNT")'" .EQS. "" THEN COPY NL: CHECK.PNT
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-2005 14:48
To:    MICHAELS
Subj:  Build summary of MAIN_TARGET ending at 21-FEB-2005 14:48:06.85
Completed build of TEST1.EXE at 21-FEB-2005 14:47:32.99
Completed build of TEST2.EXE at 21-FEB-2005 14:47:49.65
Completed build of TEST3.EXE at 21-FEB-2005 14:48:06.33
```

3.8. Creating and Using Time Stamps

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

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

3.8.1. Using DCL Symbols

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

```
PROJECT_SOURCES = PARSE.Y, TOUCH.C, GM.C, DRIVE.C, CLP.C, -
                  LEX.C, GRAFBUILD.C, GRAFWALK.C, LFS.C, -
                  MACROBANK.C, MB.C, MMSPRINT.C, UTILS.C, -
                  EXECCMD.C, RULES.C, LBR.C, CMSACCESS.C, -
                  MMSMSG.MSG, FILTER.C, GRAPH.H, GLOBALS.H, -
                  LBRDEF.H, PDEFS.H, TOKEN.H, CLP.H, TC.H
! Special CMS file types 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 COPY NL: CMSMODS.RPT
```

```
   OPEN/APPEND CHECK CMSMODS.RPT
   WRITE CHECK ""MODS' MODIFICATIONS DETECTED AT "F$TIME()'"
   CLOSE CHECK
INIT :
   MODS = 0
```

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

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

3.8.2. Using Include Files

Consider the following directories and files:

```
[DIR1] contains FILE1.X
[DIR2] contains FILE2.Y
[DIR3] contains FILE3.Z
```

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

```
.SILENT
RECORD_CHANGE = .INCLUDE CHANGE.REC
REPORT_CHANGE : INIT FILE1.TIM FILE2.TIM FILE3.TIM
   IF ""F$SEARCH("CHANGES.DOC")'" .NES. "" -
      THEN TYPE CHANGES.DOC
   IF ""F$SEARCH("CHANGES.DOC")'" .EQS. "" -
      THEN WRITE SYS$OUTPUT "No changes detected"
INIT :
   IF ""F$SEARCH("CHANGES.DOC")'" .NES. "" -
      THEN DELETE CHANGES.DOC;*/NOLOG
! Testing the time stamps
FILE1.TIM : [DIR1]FILE1.X
$(RECORD_CHANGE)
FILE2.TIM : [DIR2]FILE2.Y
$(RECORD_CHANGE)
FILE3.TIM : [DIR3]FILE3.Z
$(RECORD_CHANGE)
```

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

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

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

```
IF ""F$SEARCH("CHANGES.DOC")'" .EQS. "" -
   THEN COPY NL: CHANGES.DOC
   OPEN/APPEND 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 appears as follows:

```
.SILENT
REPORT_CHANGE : INIT FILE1.TIM FILE2.TIM FILE3.TIM
   IF ""F$SEARCH("CHANGES.DOC")'" .NES. "" -
      THEN TYPE CHANGES.DOC
   IF ""F$SEARCH("CHANGES.DOC")'" .EQS. "" -
      THEN WRITE SYS$OUTPUT "No changes detected"
INIT :
   IF ""F$SEARCH("CHANGES.DOC")'" .NES. "
      THEN DELETE CHANGES.DOC;*/NOLOG
! Testing the time stamps
FILE1.TIM : [DIR1]FILE1.X
.INCLUDE $(REC_PROC)
FILE2.TIM : [DIR2]FILE2.Y
.INCLUDE $(REC_PROC)
FILE3.TIM : [DIR3]FILE3.Z
.INCLUDE $(REC_PROC)
```

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

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

3.9. Deleting Files Selectively

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

- Create a command procedure.
- Use a macro definition.
- Use the .LAST directive.

The first two methods are described in the following sections. The use of the .LAST directive is described in *Section 2.7.9, ".LAST Directive"*.

3.9.1. Using 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; take care that you are deleting only those files you want deleted.

The modified .OBJ.OLB rule appears as follows:

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

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

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

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

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

```
$ MMS/SKIP_INTERMEDIATE DELETE
```

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

3.9.2. Using a Macro Definition

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

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

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

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

Thus, the .OBJ.OLB rule appears as follows:

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

The command line is the following:

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

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

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

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

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

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

3.10. Using Parallel Processing

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

```
PARALLEL_PROC : TARG1 TARG2 TARG3 ! Names for parts of your system
   ! Files submitted
TARG1 :
   MMS/CMS/OUT=TARG1.COM/NOACTION PROG.EXE
   SUBMIT $(MMS$TARGET_NAME)
TARG2 :
   MMS/CMS/OUT=TARG2.COM/NOACTION MOD.EXE
   SUBMIT $(MMS$TARGET_NAME)
.
.
.
! The rules building the parts of your system
PROG.EXE : PROG.OBJ
   action
MOD.EXE : MOD.OBJ
   action
.
.
.
```

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

3.11. Using MMS in Complex Examples

This section demonstrates the following advanced uses of MMS:

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

3.11.1. MMS and Object Libraries

Example 3.4, "Description File Using Object Libraries" contains a sample MMS description file using object libraries.

Example 3.4. Description File Using Object Libraries

```
!
! DESCRIP.MMS
!
! This description file builds the INTERCOM facility.
! ❶
!
!
! Define macros for the following commands and built-in rules
!
DEBUG       = /noDEBUG
TRACE       = /noTRACE
LIST        = /LIST=LIS$:
BFLAGS      = $(LIST) $(DEBUG) /TERM=STAT
MFLAGS      = $(LIST) $(DEBUG)
CLDFLAGS    = $(LIST)
!LIBRFLAGS  = /LOG
LIBRFLAGS   =
LINKFLAGS   = /FULL $(DEBUG) $(TRACE) /MAP=LIS$:
!
! TNAME gives just the name portion of the target
!
TNAME       = 'F$PARSE("MMS$TARGET_NAME",,,"NAME","SYNTAX_ONLY")
!
! Define "built-in" rules
! ❷
!
.SUFFIXES : ;
.SUFFIXES : .EXE .OLB .OBJ -
.B32 .BLI   .MAR .CLD .L32 .R32 .REQ .SDL .MSG
.B32.OBJ  : ; $(BLISS) $(BFLAGS) /OBJ=$(MMS$TARGET) $(MMS$SOURCE)
.MSG.OBJ  : ; MESSAGE $(LIST) /OBJ=$(MMS$TARGET) $(MMS$SOURCE)
.MAR.OBJ  : ; $(MACRO) $(MFLAGS) /OBJ=$(MMS$TARGET) $(MMS$SOURCE)
.CLD.OBJ  : ; SET COMMAND /OBJECT=$(MMS$TARGET) $(CLDFLAGS) $(MMS$SOURCE)
.REQ.L32  : ; $(BLISS) /LIBRARY $(BFLAGS) /NOOBJ $(MMS$SOURCE)
.OBJ.OLB  : ❸
      @ IF F$SEARCH(F$PARSE("$(MMS$TARGET)")) .NES. F$SEARCH("$(MMS
$TARGET)")-
        THEN COPY/LOG $(MMS$TARGET) $(MMS$TARGET)
      @ IF F$SEARCH("$(MMS$TARGET)") .EQS. "" -
        THEN $(LIBR)/CREATE/LOG $(MMS$TARGET)
      $(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
!
! Define groups of OBJs, how modules in the OLB relate to OBJs
!
OLB_ELEMENTS = -
NOTEFILE=OBJ$:NOTEFILE.OBJ ,-
CLASS=OBJ$:CLASS.OBJ ,-
ENTRY=OBJ$:ENTRY.OBJ ,-
KEYWORD=OBJ$:KEYWORD.OBJ ,-
NOTE=OBJ$:NOTE.OBJ ,-
PROFILE=OBJ$:PROFILE.OBJ ,-
USER=OBJ$:USER.OBJ ,-
CALLABLE_INTERCOM=OBJ$:CALLABLE_INTERCOM.OBJ ,-
CALLUSER=OBJ$:CALLUSER.OBJ ,-
PARSEACT=OBJ$:PARSEACT.OBJ ,-
INTERCOMTPU=OBJ$:INTERCOMTPU.OBJ ,-
```

```
-
FILEIO=OBJ$:FILEIO.OBJ ,-
HANDLER=OBJ$:HANDLER.OBJ ,-
ITEMSIZE=OBJ$:ITEMSIZE.OBJ ,-
ITEMLIST=OBJ$:ITEMLIST.OBJ ,-
IDPARSE=OBJ$:IDPARSE.OBJ ,-
INTERCOMMSG=OBJ$:INTERCOMMSG.OBJ ,-
INTERCOMUTIL=OBJ$:INTERCOMUTIL.OBJ ,-
INTERCOM$COMMAND_TABLE=OBJ$:COMMANDS.OBJ ,-
INTERCOM$MAIN=OBJ$:INTERCOM$MAIN.OBJ ,-
INTERCOM$SERVER=OBJ$:INTERCOM$SERVER.OBJ ,-
TFRVEC=OBJ$:TFRVEC.OBJ
!
! Define the main targets. These are put in the kit.
!
MAIN_TARGETS = -
OBJ$:INTERCOM$SHARE.EXE, -
OBJ$:INTERCOM$MAIN.EXE, -
OBJ$:INTERCOM$SERVER.EXE, -
OBJ$:INTERCOM$SECTION.GBL, -
OBJ$:INTERCOM$HELP.HLB, -
SRC$:INTERCOM_INTERFACE.TPU, -
SRC$:INTERCOM$STARTUP.COM, -
SRC$:INTERCOMDCL.CLD
!
! Define dependency rules.
!
! Specify the main target(s) -- everything.
! This is the first dependency rule in this file;
! by calling this target "*", we can say "MMS *" at DCL level.
!
* : $(MAIN_TARGETS)
CONTINUE
!
! Define the kit (this is not built by default)
! Use "MMS OBJ$:INTERCOM_KIT" to build the installation kit.
!
OBJ$:INTERCOM_KIT : OBJ$:INTERCOM000.A
CONTINUE
OBJ$:INTERCOM000.A : $(MAIN_TARGETS), SRC$:KITINSTAL.COM, SRC$:SPKITBLD.COM
! ❹
- DELETE NNP$:[INTERCOM.TEMP]*.*.*
COPY $(MAIN_TARGETS),SRC$:KITINSTAL.COM NNP$:[INTERCOM.TEMP]
@SRC$:SPKITBLD.COM $(TNAME) OBJ$: NNP$:[INTERCOM.TEMP]*.*
- DELETE NNP$:[INTERCOM.TEMP]*.*.*
!
! Build the documents (not built by default)
!
SPECS : OBJ$:INTERCOMPLAN.MEM, OBJ$:INTERCOMSPEC.MEM !To say "MMS SPECS" at
 DCL
CONTINUE
OBJ$:INTERCOMPLAN.MEM : SRC$:INTERCOMPLAN.RNO
COPY NL: SYS$SCRATCH:INTERCOMPLAN.RNT !Create dummy file
RUNOFF /INTERMEDIATE=SYS$SCRATCH:INTERCOMPLAN /NOOUTPUT SRC$:INTERCOMPLAN
RUNOFF /CONTENTS /OUTPUT=SYS$SCRATCH:INTERCOMPLAN SYS$SCRATCH:INTERCOMPLAN
RUNOFF SRC$:INTERCOMPLAN /OUT=OBJ$:INTERCOMPLAN
- DELETE SYS$SCRATCH:INTERCOMPLAN.*;*
OBJ$:INTERCOMSPEC.MEM : SRC$:INTERCOMSPEC.RNO
```

```
COPY NL: SYS$SCRATCH:INTERCOMSPEC.RNT !Create dummy file
RUNOFF /INTERMEDIATE=SYS$SCRATCH:INTERCOMSPEC /NOOUTPUT SRC$:INTERCOMSPEC
RUNOFF /CONTENTS /OUTPUT=SYS$SCRATCH:INTERCOMSPEC SYS$SCRATCH:INTERCOMSPEC
RUNOFF SRC$:INTERCOMSPEC /OUT=OBJ$:INTERCOMSPEC
- DELETE SYS$SCRATCH:INTERCOMSPEC.*;*
!
! Build the executables and libraries
!
LINK_SHR = $(LINK) $(LINKFLAGS) /SHAR=$(MMS$TARGET) $(MMS$SOURCE)/OPT /
NODEBU ❺
LINK_EXE = $(LINK) $(LINKFLAGS) /EXEC=$(MMS$TARGET) $(MMS$SOURCE)/OPT
OBJ$:INTERCOM$SHARE.EXE : SRC$:INTERCOM$SHARE.OPT OBJ$:INTERCOM.OLB ;
 $(LINK_SHR)
OBJ$:INTERCOM$MAIN.EXE : SRC$:INTERCOM$MAIN.OPT OBJ$:INTERCOM.OLB ;
 $(LINK_EXE)
OBJ$:INTERCOM$SERVER.EXE : SRC$:INTERCOM$SERVER.OPT OBJ$:INTERCOM.OLB ;
 $(LINK_EXE)
OBJ$:INTERCOM$SECTION.TPU$SECTION : SRC$:INTERCOM_INTERFACE.TPU
DEFINE /NOLOG /USER INTERCOM$SECTION OBJ$:INTERCOM$SECTION
- EDIT /TPU /SECTION=EVESECINI /NOJOURNAL /NODISPLAY /COMMAND=$(MMS$SOURCE)
OBJ$:INTERCOM$HELP.HLB : SRC$:INTERCOMHELP.HLP ❸
@ IF F$SEARCH(F$PARSE("$(MMS$TARGET)")) .NES. F$SEARCH("$(MMS$TARGET)")-
THEN COPY/LOG $(MMS$TARGET) $(MMS$TARGET)
@ IF F$SEARCH("$(MMS$TARGET)") .EQS. "" -
THEN $(LIBR)/CREATE/LOG/HELP $(MMS$TARGET)
$(LIBR) $(LIBRFLAGS) /HELP $(MMS$TARGET) $(MMS$SOURCE)
! Anything depending on INTERCOM.OLB is presumed to depend on all its
 modules.
!
OBJ$:INTERCOM.OLB : OBJ$:INTERCOM.OLB($(OLB_ELEMENTS))
CONTINUE
! Any BLISS modules that REQUIRE 'INTERCOMREQ' also depend on
 INTERCOMLIB.L32,
! since it is referenced by INTERCOMREQ.REQ. We combine these two in a
 macro.
!
COM_REQ = SRC$:INTERCOMREQ.REQ OBJ$:INTERCOMLIB.L32
! .B32 sources
!
OBJ$:NOTEFILE.OBJ        : SRC$:NOTEFILE.B32        $(COM_REQ) OBJ$:CXF.L32
OBJ$:CLASS.OBJ           : SRC$:CLASS.B32           $(COM_REQ) OBJ$:CXF.L32
OBJ$:ENTRY.OBJ           : SRC$:ENTRY.B32           $(COM_REQ) OBJ$:CXF.L32
OBJ$:NOTE.OBJ            : SRC$:NOTE.B32            $(COM_REQ) OBJ$:CXF.L32
OBJ$:KEYWORD.OBJ         : SRC$:KEYWORD.B32         $(COM_REQ) OBJ$:CXF.L32
OBJ$:PROFILE.OBJ         : SRC$:PROFILE.B32         $(COM_REQ) OBJ$:CXF.L32
OBJ$:USER.OBJ            : SRC$:USER.B32            $(COM_REQ) OBJ$:CXF.L32
OBJ$:FILEIO.OBJ          : SRC$:FILEIO.B32          $(COM_REQ)
OBJ$:HANDLER.OBJ         : SRC$:HANDLER.B32         $(COM_REQ)
OBJ$:ITEMLIST.OBJ        : SRC$:ITEMLIST.B32        $(COM_REQ)
OBJ$:ITEMSIZE.OBJ        : SRC$:ITEMSIZE.B32        $(COM_REQ)
OBJ$:INTERCOMUTIL.OBJ    : SRC$:INTERCOMUTIL.B32    $(COM_REQ)
OBJ$:IDPARSE.OBJ         : SRC$:IDPARSE.B32         $(COM_REQ)
OBJ$:CALLUSER.OBJ        : SRC$:CALLUSER.B32        $(COM_REQ) OBJ
$:USERDEF.L32
OBJ$:PARSEACT.OBJ        : SRC$:PARSEACT.B32        $(COM_REQ) OBJ
$:USERDEF.L32
OBJ$:INTERCOMTPU.OBJ     : SRC$:INTERCOMTPU.B32     $(COM_REQ) OBJ
$:USERDEF.L32
```

```
OBJ$:INTERCOM$MAIN.OBJ   : SRC$:INTERCOM$MAIN.B32   $(COM_REQ)
OBJ$:INTERCOM$SERVER.OBJ : SRC$:INTERCOM$SERVER.B32 $(COM_REQ)
OBJ$:CALLABLE_INTERCOM.OBJ : SRC$:CALLABLE_INTERCOM.B32 $(COM_REQ) OBJ
$:USERDEF.L32
! .MAR sources
!
OBJ$:TFRVEC.OBJ : SRC$:TFRVEC.MAR
! .REQ sources
!
OBJ$:CXF.L32      : SRC$:CXF.REQ      OBJ$:INTERCOMLIB.L32
OBJ$:USERDEF.L32  : SRC$:USERDEF.REQ  OBJ$:INTERCOMLIB.L32
!
! Several require files are combined to form a single BLISS library
!
OBJ$:INTERCOMLIB.L32 : SRC$:INTERCOMTRUC.REQ SRC$:INTERCOMMAC.REQ SRC
$:RTN.REQ -
OBJ$:INTERCOMMSG.R32 OBJ$:NOTEITEMS.R32
$(BLISS) /LIBRARY $(BFLAGS) /NOOBJ /NOLIST /LIBR=$(MMS$TARGET)-
SRC$:INTERCOMTRUC.REQ+SRC$:INTERCOMMAC.REQ+SRC$:RTN.REQ+-
OBJ$:INTERCOMMSG.R32+OBJ$:NOTEITEMS.R32
! .MSG source
!
OBJ$:INTERCOMMSG.OBJ : SRC$:INTERCOMMSG.MSG
! .CLD sources
!
OBJ$:COMMANDS.OBJ : SRC$:COMMANDS.CLD
```

Key to the example:

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

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

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

The logical names were defined in either of the following formats:

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

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

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

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

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

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

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

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

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

3.11.2. Producing Multiple Outputs with MMS

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

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

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

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

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

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

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

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

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

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

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

3.11.2.1. Independent Outputs

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

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

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

3.11.2.2. Dependent Outputs

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

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

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

```
AB.EXE DEPENDS_ON A.OBJ B.OBJ
   LINK/EXECUTABLE=AB.EXE A.OBJ, B.OBJ
A.OBJ DEPENDS_ON A.PAS B.PEN
   PASCAL/OBJECT=A A.PAS
B.OBJ DEPENDS_ON B.PAS
   PASCAL/ENVIRONMENT=B/OBJECT=B B.PAS
```

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

```
B.OBJ B.PEN DEPENDS_ON B.PAS
```

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

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

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

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

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

3.11.3. Multiple Outputs Workaround

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

You can introduce a context variable into the action block to monitor whether the compilation has been performed. Consider the following description file:

```
AB.EXE DEPENDS_ON A.OBJ B.OBJ
   LINK/EXECUTABLE=AB.EXE A.OBJ, B.OBJ
A.OBJ DEPENDS_ON A.PAS B.PEN
   PASCAL/OBJECT=A A.PAS
B.OBJ B.PEN DEPENDS_ON B.PAS
   IF F$TYPE(B_COMPILED) .EQS. "" THEN B_COMPILED=0
   IF .NOT. B_COMPILED THEN
     PASCAL/ENVIRONMENT=B/OBJECT=B B.PAS
   IF .NOT. B_COMPILED THEN
     B_COMPILED=1
```

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

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

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

Chapter 4. Accessing Libraries and Other Objects

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

- OpenVMS libraries created with the LIBRARY utility
- Code Management System (CMS) libraries
- Forms Management System (FMS) libraries
- Oracle CDD/Repository
- Source Code Analyzer (SCA) libraries

4.1. Creating and Accessing Files in OpenVMS Libraries

You can use MMS to access files that are contained in OpenVMS libraries and to create library files using certain built-in rules. The built-in rules that MMS uses to create library files are listed in *Section C.7, "Built-In Rules for Library Files"*. These rules tell MMS to create the specified library if one does not already exist; they then cause MMS to replace the source module in the target library.

Note

You cannot use MMS to access modules in an RSX library because only a module's revision date is recorded, not its revision time.

4.1.1. Formatting Library Module Specifications

To specify that a source or target in a dependency rule is a module in a OpenVMS library, use the following format. Avoid adding any spaces or tabs because they might cause problems in processing.

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

In this format, the *library* is an OpenVMS file specification that denotes a library. The default file type is .OLB if you are referring to a module within the library. If you are referring to the entire library, there is no default file type.

The *module* is the name of the module in the library. The *filespec* is the OpenVMS file specification that corresponds to the module in the library. The default file type depends on the file type of the library.

The format shown in this section describes how to refer to modules within a library. You can also use MMS to process the library file itself by providing the library file specification (for example, CRTLIB.OLB).

There is a restriction in using library module specifications as targets in a double-colon dependency rule. See *Section 3.1, "Using Double-Colon Dependencies"* for more information.

4.1.2. Using Logical Names in a Library Module Specification

You can use a logical name for the name of the module if you supply the action lines that update the target. MMS cannot apply its built-in rules to a logical name because it relies on file types to determine which built-in rule is appropriate.

For example, CRTLIB(C$STRLEN=STRLEN.OBJ) designates that the module C$STRLEN is in library CRTLIB.OLB; C$STRLEN is found in the file named STRLEN.OBJ.

4.1.3. Specifying Multiple Modules

You can specify multiple modules in the same library in two ways:

- You can enclose the module names in one set of parentheses and separate them with commas. For example, to refer to three modules in the library CRTLIB.OLB, you can specify CRTLIB(C $STRLEN=STRLEN.OBJ, C$STRPAD=STRPAD.OBJ, C$STRIND=STRIND.OBJ).
- You can use the OpenVMS * and % wildcard characters. For example, the specification CRTLIB (C$*) directs MMS to look for all modules in the library CRTLIB.OLB whose names begin with the characters C$. In the same way, the specification CRTLIB (C$STR%) directs MMS to look for all modules in CRTLIB.OLB whose names begin with the characters C$STR followed by only one character.

4.1.4. Accessing Library Modules with Non-OpenVMS File Specifications

You can use a complete library specification when you need to access library modules whose names do not correspond to OpenVMS file specifications (for example, C$STRLEN=STRLEN.OBJ). However, MMS can interpret shorter specifications as follows:

- If the module's name in the library is the same as its file name, you can provide just the module name in parentheses after the library name. For example, if the module in the previous example were named STRLEN, you could refer to the module in the library as CRTLIB (STRLEN).
- If the module's file type is associated by default with the type of the library, you can omit the file type. In the specification CRTLIB (STRLEN), MMS assumes that the file name is STRLEN.OBJ, because .OLB libraries are assumed to contain .OBJ modules. If the module's file type is something other than the default for that kind of library, you must supply the file type. For example, if the module were STRLEN.C, you would have to specify CRTLIB (STRLEN.C). MMS would then expand this specification to the following two dependencies:

```
CRTLIB(STRLEN=STRLEN.OBJ) : STRLEN.OBJ
STRLEN.OBJ : STRLEN.C
```

4.1.5. Using Special Macros with Library Specifications

When used with library specifications, certain MMS special macros have slightly different meanings:

- If a library is the source in a dependency rule, MMS$SOURCE expands to the complete specification of the module in the library. For example, in the specification CRTLIB (C $STRLEN=STRLEN), MMS$SOURCE expands to CRTLIB.OLB (C$STRLEN=STRLEN.OBJ).

- If a library is the target in a dependency rule, MMS$TARGET expands to the name of the library. For example, in the specification CRTLIB (C$STRLEN), MMS$TARGET becomes CRTLIB.
- If a library is the target in a dependency rule, MMS$TARGET_NAME expands to the module name (without its file type). For example, if the library module is STRLEN, MMS$TARGET_NAME expands to STRLEN.

In addition to these MMS special macros, there is another special macro, MMS$LIB_ELEMENT, which you can use only in library specifications. MMS$LIB_ELEMENT expands to the string between parentheses, that is, to the module name and its corresponding file specification. For example, in the specification CRTLIB (STRLEN), MMS$LIB_ELEMENT becomes STRLEN=STRLEN.OBJ. The expansion of MMS$LIB_ELEMENT does not depend on whether the library is the source or the target in a dependency rule. If the library is specified as both the source and the target, the target is expanded.

4.1.6. Using Libraries as a Source

The use of libraries as a source is shown in the following example. Suppose your description file contains the following dependency rules:

```
TOOLTITLE.EXE : SYS$LIBRARY:CRTLIB.OLB, USER$:[WATKINS]FLIB.OLB
   LINK TOOLTITLE.OBJ, SYS$LIBRARY:VAXCRTL/LIB, USER$:[WATKINS]FLIB/LIB
TOOLTITLE.OBJ : SMGTEXLIB.TLB(SMGDEF=SMGDEF.H)
   CC TOOLTITLE + SMGTEXLIB/LIB
```

TOOLTITLE.C is a C program that includes the file SMGDEF.H, which is stored in the text library SMGTEXLIB.TLB under the name SMGDEF. The action line in the second dependency rule invokes the C compiler to compile TOOLTITLE.C and the text library. You must state this action line explicitly. In this case, specifying only the target and source is not sufficient. The first dependency rule invokes the OpenVMS Linker to link TOOLTITLE.OBJ with one system library and one user library.

4.2. Using MMS with CMS

If the Code Management System (CMS) is installed on your system, you can use MMS to access elements in CMS libraries. CMS elements are denoted by the tilde (~). You should be familiar with CMS before you read this section.

MMS provides default macros and special macros tailored for use with CMS. *Appendix C, "MMS Built-In Features"* lists these default and special macros.

MMS can build your system software from source code files stored in CMS libraries. MMS fetches the source code file from its library, compiles the code into object files, then links the object files into executable images. MMS treats the CMS library as the source for your source code files.

MMS treats the source code in your directory and an element in a CMS library, just as it does any other source or target pair. If the source code file in the default directory is missing, MMS fetches that element from its CMS library. If the source code file in the default directory is older than the library element, MMS fetches the element.

MMS supports only one CMS qualifier: /GENERATION. The default macro CMSFLAGS expands to the /GENERATION qualifier. You can also specify /GENERATION and a generation number after the tilde (~) in the element name, as shown in this example:

```
PROG.OBJ : [OTHER.CMS]PROG.C~/GEN=4A1
```

The tilde format for an explicit reference to a CMS element is useful when you are certain that the most up-to-date source is stored in the CMS library. A more convenient use of MMS with CMS is to let MMS

determine where the newest source is located and to fetch the CMS element automatically, if necessary. To take advantage of this feature, use the /CMS qualifier on the MMS command line and do not use the tilde format for specifying the source.

4.2.1. Using CMS Commands in a Description File

You can use any CMS command in an MMS description file.

As MMS examines target and source lines in your main process, it uses the CMS$LIB logical name to establish the CMS directory from which sources will be fetched if the target must be updated. If you use the CMS SET LIBRARY command in an action line, that command is executed in the subprocess where MMS normally executes action lines. Such an action establishes a new library as the current default for any subsequent action lines that execute CMS commands. However, the value of CMS$LIB is not changed in the main process because the CMS SET LIBRARY command is executed in the subprocess. MMS still looks for sources in the library represented by CMS$LIB.

The MMS qualifier /REVISE_DATE has no effect when MMS is accessing elements in CMS libraries.

4.2.2. Automatic Access of CMS Elements from Dependency Rules

The /CMS qualifier directs MMS to look for sources in the current default CMS library, as well as in the directories specified in the description file. If the CMS element has been replaced in the library since the file in the specified directory was last revised, MMS directs CMS to fetch the source from the library so the target can be rebuilt. MMS has built-in rules that instruct CMS how to find the correct source (see *Section C.9, "Built-In Rules for CMS Access"* for the built-in CMS rules). MMS uses the sources fetched from CMS to update the target by executing the action lines in the description file. The CMSFLAGS default macro determines which generation of an element is fetched as the source, or from which class the source element is fetched.

If the file in the specified directory is newer than the CMS element, MMS uses that file. Therefore, you could edit a source in your directory and build a new system with the edited source, rather than with the corresponding CMS element.

For example, consider a description file that contains the following dependencies:

```
A.EXE : A.OBJ
A.OBJ : A.PAS
```

If you invoke MMS with the /CMS qualifier, MMS processes the description file by looking in the current default CMS library for A.PAS. If it locates that source, it compares the revision time with the revision time of A.PAS in the current directory (if A.PAS exists there). If the CMS element is newer, MMS uses it to update A.OBJ.

The CMSFLAGS default macro always fetches the most recent generation of an element on the main line of descent. You can redefine CMSFLAGS to indicate a specific element generation or the element generation that belongs to a particular class. However, if you do so, you must be aware that if newer generations exist in the library, they will not be fetched; MMS checks 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 uses it even though more recent generations of the element might 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 the *Chapter 5, "Command Dictionary"*.

4.2.3. Explicit References to CMS Elements in Dependency Rules

The /CMS qualifier causes MMS to compare the times of a CMS element and a file in the specified directory, if both exist. You can also direct MMS to check only the CMS element by putting a tilde immediately after the source-file name in a dependency rule. For example, the tilde in the following target or source line directs MMS to look for the source PROG.C in the current default CMS library:

```
PROG.OBJ : PROG.C~
```

If you use the tilde format to indicate CMS elements, you can specify only one element in a given dependency rule. You cannot specify a list of CMS elements if their file specifications are followed by tildes.

If the element is in a CMS library other than the current default library, you must type the library specification before the element name:

```
PROG.OBJ : [OTHER.CMS]PROG.C~
```

You might not be able to access elements in a CMS library that reside on a DECnet node other than your own.

4.2.4. Using CMS Elements to Build the System

The following example demonstrates how to build your software system with CMS elements. Consider the description file CMS_MMS.MMS, shown in *Example 4.1, "Description File Using CMS Libraries"*.

Example 4.1. Description File Using CMS Libraries

```
!
! Executable target
!
MAIN.EXE : MAIN.OBJ, SUB1.OBJ ❶
LINK $(MMS$SOURCE_LIST)
!
! Object files and their sources
!
MAIN.OBJ : MAIN.PAS ❷
SUB1.OBJ : SUB1.PAS
!
! Where the sources are stored
!
MAIN.PAS : DISK1:[SYSTEM2_LIB]MAIN.PAS~ ❸
SUB1.PAS : DISK1:[SYSTEM2_LIB]SUB1.PAS~)
```

Key to the example:

❶ The executable target is stated.

❷ The objects or targets are stated.

❸ Each source code file has a target or source line. The element in the CMS library is the source.

You can build your system with the description file CMS_MMS.MMS, as shown in *Example 4.2, "Building a System from CMS Library Elements"*.

Example 4.2. Building a System from CMS Library Elements

```
$ DIR/DATE=MODIFIED ❶
 Directory DISK1:[TEST]
 CMS_MMS.MMS;1   30-JUL-2005 13:10
 Total of 1 file.
$ MMS/DESCRIPTION=CMS_MMS ❷
 mms$cmslib :== 'f$logical("CMS$LIB")
 IF mms$cmslib .nes. "DISK1:[SYSTEM2_LIB]"
    THEN CMS SET LIBRARY DISK1:[SYSTEM2_LIB]
CMS FETCH MAIN.PAS /GEN=1+ "" ❸
 %CMS-S-FETCHED, generation 1 of element MAIN.PAS fetched
 IF mms$cmslib .EQS. "" THEN CMS SET LIBRARY 1234
 IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM2_LIB]"
    THEN CMS SET LIBRARY 'mms$cmslib'
PASCAL /NOLIST/OBJECT=MAIN MAIN.PAS ❹
 mms$cmslib :== 'f$logical("CMS$LIB")
 IF mms$cmslib .nes. "DISK1:[SYSTEM2_LIB]" THEN
 CMS SET LIBRARY DISK1:[SYSTEM2_LIB]
CMS FETCH SUB1.PAS /GEN=1+ "" ❺
 %CMS-S-FETCHED, generation 1 of element SUB1.PAS fetched
 IF mms$cmslib .EQS. "" THEN CMS SET LIBRARY 1234
 IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM2_LIB]"
    THEN CMS SET LIBRARY 'mms$cmslib'
PASCAL /NOLIST/OBJECT=SUB1 SUB1.PAS ❻
LINK MAIN.OBJ, SUB1.OBJ ❼
$ DIR/DATE=MODIFIED ❽
Directory DISK1:[TEST]
MAIN.EXE;1     30-JUL-2005 13:19
MAIN.OBJ;1     30-JUL-2005 13:19
MAIN.PAS;1     30-JUL-2005 13:10
SUB1.OBJ;1     30-JUL-2005 13:19
SUB1.PAS;1     30-JUL-2005 13:10
CMS_MMS.MMS;1  30-JUL-2005 13:10
Total of 6 files.
```

Key to the example:

❶ The default directory contains only the software description.

❷ MMS is invoked using the CMS_MMS.MMS description file.

❸ MMS fetches the missing source code file MAIN.PAS from CMS. Note that MMS fetches the latest generation of each element by using the 1+ notation.

❹ MMS compiles the source code file MAIN.PAS.

❺ MMS fetches missing source code file SUB1.PAS from CMS.

❻ MMS compiles the source code file SUB1.PAS.

❼ MMS links the object files using the action line.

❽ The default directory has a complete software system.

MMS fetches sources from CMS if the source code file in your directory is missing or older. In this example, only the description file is in the default directory before you invoke MMS. There is no source

code to compile in your default directory, but because the description file states the source of each source code file, MMS fetches the source code files from CMS.

Because of the MMS built-in rule for accessing CMS libraries, MMS generates more output when it fetches source files from CMS than when it builds a system with files from your directory. The extra actions reflected in the output are concerned with checking that the CMS library is set to the correct place before and after the fetch, in case the source code is stored in more than one library.

Appendix C, "MMS Built-In Features" contains a full list of extensions that MMS must know to fetch files from a CMS library and to access other libraries.

4.2.5. Using CMS Libraries to Rebuild the System

Rebuilding a system from a CMS library is similar to all other system rebuilding in that MMS first checks that all parts of the system are present and up-to-date, then recreates executable files and object files that are old or missing. MMS also uses the CMS library as the source to update old and missing source code.

Example 4.3, "Rebuilding Using CMS Libraries" is an example of what can happen during the software development cycle. You reserve an element, fix a bug in the element, and test the fix before replacing the code element in the library. No one else has updated the library element since the last system build.

Example 4.3. Rebuilding Using CMS Libraries

```
$ DIR/DATE=MODIFIED ❶
Directory DISK1:[TEST]
MAIN.EXE;1     30-JUL-2005 13:15
MAIN.OBJ;1     30-JUL-2005 13:13
MAIN.PAS;1     30-JUL-2005 13:10
SUB1.OBJ;1     30-JUL-2005 13:13
SUB1.PAS;1     30-JUL-2005 13:10
CMS_MMS.MMS;1  30-JUL-2005 13:10
Total of 6 files.
$ CMS SET LIBRARY DISK1:[SYSTEM2_LIB]
%CMS-I-LIBIS, CMS library is DISK1:[SYSTEM2_LIB]
$ CMS RESERVE SUB1.PAS "Fixing a bug in stack handler" ❷
%CMS-S-RESERVED, generation 1 of element SUB1.PAS reserved
$ EDIT SUB1.PAS ❸
$ PURGE
$ DIR/DATE=MODIFIED ❹
Directory DISK1:[TEST]
MAIN.EXE;1     30-JUL-2005 13:15
MAIN.OBJ;1     30-JUL-2005 13:13
MAIN.PAS;1     30-JUL-2005 13:10
SUB1.OBJ;1     30-JUL-2005 13:13
SUB1.PAS;3     30-JUL-2005 13:17
CMS_MMS.MMS;1  30-JUL-2005 13:10
Total of 6 files.
$ MMS/DESCRIPTION=CMS_MMS ❺
PASCAL /NOLIST/OBJECT=SUB1 SUB1.PAS ❻
  LINK MAIN.OBJ, SUB1.OBJ
```

Key to the example:

❶ You have a complete, up-to-date system from a previous system build.

❷ You reserve a CMS element.

❸ You modify SUB1.PAS to fix a bug.

❹ One source code file SUB1.PAS is newer than its object SUB1.OBJ.

❺ You invoke MMS to rebuild the system.

❻ MMS rebuilds the system from files in your default directory, fetching nothing from the library. MMS compiles SUB1.PAS and links MAIN.OBJ and SUB1.OBJ.

When you invoke MMS in *Example 4.3, "Rebuilding Using CMS Libraries"*, MMS does not fetch any files from the CMS library. Nothing in your directory is missing and nothing is older than its library element.

In the next example, you have a complete, up-to-date copy of the entire system in your directory. However, another person has updated one of the code files in the CMS library after you built your copy of the system. When you invoke MMS to see if your system is up-to-date, MMS detects the newer file in the CMS library and fetches the updated file from the library. It rebuilds the system using the newly fetched file. This demonstrates how MMS uses the CMS library as the source of your targets. Just as an object file is rebuilt if the code has changed, your file is updated if the library has changed. For example:

```
$ DIR/DATE=MODIFIED ❶
Directory DISK1:[TEST]
MAIN.EXE;2      30-JUL-2005 13:15
MAIN.OBJ;1      30-JUL-2005 13:13
MAIN.PAS;1      30-JUL-2005 13:10
SUB1.OBJ;2      30-JUL-2005 13:13
SUB1.PAS;3      30-JUL-2005 13:10
CMS_MMS.MMS;1   30-JUL-2005 13:10
Total of 6 files.
$ ! (Another user reserves, changes and replaces MAIN.PAS) ❷
$ MMS/DESCRIPTION=CMS_MMS ❸
 mms$cmslib :== 'f$logical("CMS$LIB")
 IF mms$cmslib .nes. "DISK1:[SYSTEM2_LIB]" THEN
  CMS SET LIBRARY DISK1:[SYSTEM2_LIB]
CMS FETCH MAIN.PAS /GEN=1+ "" ❹
 %CMS-I-FILEXISTS, file already exists, DISK1:[TEST]MAIN.PAS;2 created
 %CMS-S-FETCHED, generation 2 of element MAIN.PAS fetched
 IF mms$cmslib .EQS. "" THEN CMS SET LIBRARY 1234
 IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM2_LIB]"
   THEN CMS SET LIBRARY 'mms$cmslib'
PASCAL /NOLIST/OBJECT=MAIN MAIN.PAS ❺
 LINK MAIN.OBJ, SUB1.OBJ
```

Key to the example:

❶ Your directory has a complete, up-to-date system.

❷ Another person updates a file in the project's CMS library.

❸ You invoke MMS to update your system.

❹ MMS fetches the newer source code in the library.

❺ MMS compiles and links the newer source code.

When creating the "official" release of a software product, you want to be sure that the release is built from code in the library, not from test code that you might have in your default directory. It is better to

create a [RELEASE] directory that is used only for building the system from its CMS libraries. No other operations are performed in that directory.

4.2.6. Building a System from a Specified CMS Class

You can build your system from a specified CMS class. Building with a class specifier in CMS is identical to building from current generations in CMS.

In building a system from a specified CMS class, MMS still uses the CMS elements as the sources, but it uses the designated class of generations, not necessarily the current generations. MMS allows you to find and modify old source code and recreate previous versions of your system by building from a specific CMS class.

MMS looks for the description file on the main line of descent, unless you override the default macro CMSFLAGS. If you specify /MACRO="CMSFLAGS=/GENERATION=class-name", MMS instead uses the specified class. If MMS cannot find a description file in either your default directory or the CMS library, it stops execution.

You can use a user-defined macro to control the class that MMS fetches. The user-defined macro is used as a qualifier to one of the MMS actions. For example, you define the macro CMSFLAGS, which contains the qualifiers MMS uses when it fetches an element from a CMS library. A /GENERATION qualifier on this macro causes MMS to fetch files from a certain class. Consider the command procedure BUILD_CLASS.COM and the description file SYSTEM3.MMS in *Example 4.4, "Description File for Building from a CMS Class"*.

Example 4.4. Description File for Building from a CMS Class

```
$ TYPE BUILD_CLASS.COM ❶
$ !
$ ! Command procedure to build any CMS class of SYSTEM2.
$ !
$ ! Create the user defined macros we want
$ !
$ INQUIRE CLASS "Enter name of class to build" ❷
$ CMSFLAGS = "/GENERATION=" + CLASS ❸
$ !
$ ! Invoke MMS and direct it to use our macros
$ !
$ MMS /DESCRIPTION=SYSTEM3 /OVERRIDE ❹
$ !
$ TYPE SYSTEM3.MMS ❺
!
! Executable target
!
MAIN.EXE : MAIN.OBJ, SUB1.OBJ, SUB2.OBJ
LINK $(LINKFLAGS) $(MMS$SOURCE_LIST)
!
! Object files and their sources
!
MAIN.OBJ : MAIN.PAS
SUB1.OBJ : SUB1.PAS
SUB2.OBJ : SUB2.PAS
!
! Where the sources are stored
!
MAIN.PAS : DISK1:[SYSTEM3_LIB]MAIN.PAS~
```

```
SUB1.PAS : DISK1:[SYSTEM3_LIB]SUB1.PAS~
SUB2.PAS : DISK1:[SYSTEM3_LIB]SUB2.PAS~
```

Key to the example:

❶ The command procedure invokes MMS.

❷ The command procedure prompts you for the name of the CMS class to build.

❸ The CMSFLAGS macro (which is used by the built-in rule for fetching your source files) is set and the /GENERATION qualifier is added to the class name.

❹ The procedure invokes MMS with the /OVERRIDE qualifier so your macro is used instead of the default.

❺ The description file lists where the source files are stored in the CMS library.

You can insert the MMS description file and its calling command procedure into the appropriate CMS class. In this way, the description file in a given class builds that class. As a system changes, you can continue to put a copy of the description file in each new class you create.

4.2.7. Building a System from a Previous Class

In this example, you build your system from a previous class in a directory that is empty except for the command procedure and the description file. MMS fetches files from CMS only if the library copy is newer than your copy. The files in the previous class are certain to be older than your code, so MMS does not fetch them unless you build the previous releases in a clean directory. For example, consider the system build shown in *Example 4.5, "Building a System from a Previous CMS Class"*.

Example 4.5. Building a System from a Previous CMS Class

```
$ DIR/DATE=MODIFIED ❶
Directory DISK1:[VERSION13]
BUILD_CLASS.COM;4  5-AUG-2005 13:17
SYSTEM3.MMS;3      5-AUG-2005 13:09
Total of 2 files.
$ @BUILD_CLASS ❷
Enter name of class to build: VERSION_1_3 ❸
   mms$cmslib :== 'f$logical("CMS$LIB")
   IF mms$cmslib .nes. "DISK1:[SYSTEM3_LIB]" THEN
    CMS SET LIBRARY DISK1:[SYSTEM3_LIB]
   %CMS-I-LIBIS, CMS library is DISK1:[SYSTEM3_LIB]
CMS FETCH MAIN.PAS /GEN=VERSION_1_3 "" ❹
   %CMS-S-FETCHED, generation 2 of element MAIN.PAS fetched
   IF mms$cmslib .EQS. "" THEN -
    CMS SET LIBRARY 1234
   %CMS-E-NOREF, error referencing 1234
   -CMS-E-MUSTBEDIR, 1234 must be a directory specification
   %CMS-W-UNDEFLIB, CMS library is now undefined
   IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM3_LIB]" -
    THEN CMS SET LIBRARY 'mms$cmslib'
   PASCAL /NOLIST/OBJECT=MAIN MAIN.PAS
   mms$cmslib :== 'f$logical("CMS$LIB")
   IF mms$cmslib .nes. "DISK1:[SYSTEM3_LIB]" THEN
    CMS SET LIBRARY DISK1:[SYSTEM3_LIB]
   %CMS-I-LIBIS, CMS library is DISK1:[SYSTEM3_LIB]
CMS FETCH SUB1.PAS /GEN=VERSION_1_3 "" ❹
```

```
   %CMS-S-FETCHED, generation 1 of element SUB1.PAS fetched
   IF mms$cmslib .EQS. "" THEN -
    CMS SET LIBRARY 1234
   %CMS-E-NOREF, error referencing 1234
   -CMS-E-MUSTBEDIR, 1234 must be a directory specification
   %CMS-W-UNDEFLIB, CMS library is now undefined
   IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM3_LIB]"
    THEN CMS SET LIBRARY 'mms$cmslib'
   PASCAL /NOLIST/OBJECT=SUB1 SUB1.PAS
   mms$cmslib :== 'f$logical("CMS$LIB")
   IF mms$cmslib .nes. "DISK1:[SYSTEM3_LIB]" THEN -
    CMS SET LIBRARY DISK1:[SYSTEM3_LIB]
   %CMS-I-LIBIS, CMS library is DISK1:[SYSTEM3_LIB]
CMS FETCH SUB2.PAS /GEN=VERSION_1_3 "" ❹
   %CMS-S-FETCHED, generation 3 of element SUB2.PAS fetched
   IF mms$cmslib .EQS. "" THEN -
    CMS SET LIBRARY 1234
   %CMS-E-NOREF, error referencing 1234
   -CMS-E-MUSTBEDIR, 1234 must be a directory specification
   %CMS-W-UNDEFLIB, CMS library is now undefined
   IF mms$cmslib .NES. "" .AND. mms$cmslib .NES. "DISK1:[SYSTEM3_LIB]" -
    THEN CMS SET LIBRARY 'mms$cmslib'
PASCAL /NOLIST/OBJECT=SUB2 SUB2.PAS ❺
   LINK/TRACE/NOMAP/EXEC=MAIN MAIN.OBJ, SUB1.OBJ, SUB2.OBJ
$ DIR/DATE=MODIFIED ❻
Directory DISK1:[VERSION13]
BUILD_CLASS.COM;4  5-AUG-2005 13:17
MAIN.EXE;1         5-AUG-2005 13:22
MAIN.OBJ;1         5-AUG-2005 13:21
MAIN.PAS;1         30-JUL-2005 13:22
SUB1.OBJ;1         5-AUG-2005 13:22
SUB1.PAS;1         30-JUL-2005 13:10
SUB2.OBJ;1         5-AUG-2005 13:22
SUB2.PAS;1         5-AUG-2005 13:12
SYSTEM3.MMS;3      5-AUG-2005 13:09
Total of 9 files.
```

Key to the example:

❶ Your directory contains the command procedure and the description file.

❷ You invoke the command procedure.

❸ You enter the class name.

❹ MMS fetches all program code from the chosen class using the macro CMSFLAGS to override the default.

❺ The source code is compiled and linked normally.

❻ Your directory contains the complete system.

After you have built the previous version of your system, you can fix the bug you might have found. The following steps ensure a complete and accurate system:

1. Reserve the CMS element generation with the bug.

2. Edit the element to fix the bug.

3. Replace the CMS element as an alternate line of descent.

4. Replace the generation that was in the CMS class with the fixed version (CMS INSERT GENERATION /SUPERSEDE).

4.2.7.1. Using Logical Names for CMS Library Specifications

When writing CMS library specifiers in a description file, you can use logical names instead of a hard-coded device and directory name. This allows the exact location of the library to change, and can make the description file easier to read. The command procedure that invokes MMS can set the logical name (or perhaps it is set in a group or system logical name table).

4.2.8. Using the .INCLUDE Directive to Include CMS Files

You can also use the tilde format with the .INCLUDE directive to include files 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 described in *Section 2.7.7, ".INCLUDE Directive"*.)

When a tilde occurs in your description file, MMS looks for the file in the current CMS library, even if you specify /NOCMS on the command line. However, if the CMS element is newer than the target in the dependency, the element is not fetched from its CMS library unless an action line directs CMS to fetch the source.

4.2.9. Using a User-Defined Rule to Access a Single CMS Element

The following example shows a user-defined rule for accessing a single-file CMS element:

```
.C~.OBJ :
   CMS FETCH $(MMS$CMS_ELEMENT) $(CMSFLAGS) $(CMSCOMMENT)
   $(CC) $(CFLAGS) $(MMS$CMS_ELEMENT)
```

This dependency rule tells MMS to do the following:

1. Fetch the .C source file from the current default CMS library, applying the qualifiers specified by the CMSFLAGS macro and writing to the CMS history file the remark specified by the CMSCOMMENT macro.

2. Run the C compiler on the file fetched from the CMS library, applying the qualifiers specified by the CFLAGS macro.

CMSFLAGS and CMSCOMMENT are default MMS macros. You can redefine them so the same qualifiers or the same remarks are used for all accesses to CMS elements.

4.2.10. Accessing a CMS Element Not in the Default CMS Library

The next example shows how to access a CMS element that is not in the current default CMS library.

```
TEST.C : [OTHER.CMS]TEST.C~
   CMS SET LIBRARY [OTHER.CMS]
   CMS FETCH TEST.C "Auto fetch from MMS"
```

This dependency rule causes MMS to set the current default CMS library to [OTHER.CMS], fetch the element TEST.C, and write the specified remark to the CMS history file. (MMS does not reset the CMS library back to the default in this example. This action differs from that of the built-in rules for CMS element access. See *Section C.9, "Built-In Rules for CMS Access"*.)

4.2.11. Accessing Description Files in CMS Libraries

If a description file does not exist in your default directory, and if you have defined a CMS library, you can request that MMS retrieve the description file from the CMS library by using the /CMS qualifier on the MMS command line. If the description file exists in your directory and is newer than the element in the CMS library, MMS uses the file in your directory.

If you know that the description file you want to use is stored in a CMS library, you can explicitly request MMS to use that file. When you use the /DESCRIPTION qualifier on the MMS command line, you can follow the name of the description file with a tilde character so MMS automatically fetches the file from the current CMS library. For example:

```
$ MMS/DESCRIPTION=ALL~
```

This command directs MMS to fetch the description file ALL.MMS from the current CMS library.

If the file you specify with /DESCRIPTION does not exist in the current CMS library, MMS issues an error message.

4.3. Checking for Replacement of CMS Elements

If more than one programmer is working on a project, you might 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 next 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.TIM, as in the following example.

```
THERE.TIM : NEEDED.FOR~ ! The name of the element
   IF ""F$SEARCH("THERE.TIM")'" .NES "" -
      THEN MAIL NL: 'F$GETJPI(" ","USERNAME")'-
   /SUBJECT="$(MMS$SOURCE) is back in the CMS library."
   SET DEFAULT 1234567890 ! Causes MMS to stop with $STATUS = failure
```

The command procedure CHECKCMS.COM that loops until the specified element is available in the CMS library is as follows:

```
$ CMS SET LIBRARY [LOUISE] ! The CMS library
$ SET DEFAULT [LOUISE.WORK]       ! Your working directory
$ IF ""F$SEARCH("THERE.TIM")'"   .EQS. "" THEN COPY NL: THERE.TIM
$ LOOP:
$ MMS/DESCRIPTION=THERE
$ IF .NOT. $STATUS THEN EXIT
$ WAIT 0:5 ! or some interval
```

```
$ GOTO LOOP
```

When submitted to the batch queue, this command procedure runs MMS, which checks to see whether the element in the CMS library is newer than THERE.TIM. If it is not (that is, if the element has not been replaced in the CMS library), $STATUS is 1 and MMS waits the specified interval before trying again. If the element has been replaced, the first bit in $STATUS is 0, and MMS mails you the message "NEEDED.FOR is back in the CMS library."

You can run this procedure in a subprocess (instead of submitting it to the batch queue) by typing the following command:

```
$ SPAWN/NOWAIT @CHECKCMS
```

4.4. Accessing Forms in an FMS Library

If FMS is installed on your system, you can use MMS to access forms stored in FMS libraries. You should be familiar with FMS before reading this section.

To specify an FMS form in a dependency rule, use the same syntax as for files in OpenVMS libraries. This syntax is explained in detail in *Section 4.1, "Creating and Accessing Files in OpenVMS Libraries"*. The file type .FLB after the library name informs MMS that the library contains FMS forms. The default file type for FMS forms is .FRM.

For example, consider the following dependency rule:

```
A.FLB(B) : B.FRM
$(FMS) $(FMSFLAGS) A.FLB B.FRM
```

B.FRM is the source that updates the target B in the FMS library A.FLB. FMS and FMSFLAGS are default macros that invoke FMS with the /REPLACE qualifier.

MMS uses the insertion time of a form in an FMS library to determine whether a source is newer than the target. You cannot use the /REVISE_DATE qualifier with references to FMS forms. (See *Chapter 5, "Command Dictionary"* for a description of /REVISE_DATE.)

4.5. Accessing Definitions in Oracle CDD/Repository

If the Oracle CDD/Repository is installed on your system, you can use MMS to access records and other definitions stored in Oracle CDD/Repository, as long as the definitions have the revision time attribute. You should be familiar with Oracle CDD/Repository before reading this section.

In a dependency rule, you follow the path name of an Oracle CDD/Repository definition with the caret (^) to inform MMS that the source is stored in Oracle CDD/Repository. For example:

```
A.OBJ : A.PAS, CDD$TOP.B.C.D.E^ ! CDD record referred to in A.PAS
PASCAL A.PAS
```

In this example, the target A.OBJ resides in your current directory.

MMS uses the Oracle CDD/Repository path specification to find the source and check its revision time against that of the target, A.OBJ. Then, MMS retrieves the revision time for the specified Oracle CDD/Repository definition. The definition must have the revision time attribute. If the definition is not found in Oracle CDD/Repository, MMS attempts to locate it at the specified path by using the Oracle CDD/

Repository compatibility interface. In this case, MMS assumes that the definition is in a dictionary or subdirectory that has not yet been upgraded to Oracle CDD/Repository. If MMS cannot find the definition by using the Oracle CDD/Repository compatibility interface, it returns an error.

The following restrictions apply to Oracle CDD/Repository access:

- You cannot use the /REVISE_DATE qualifier with references to Oracle CDD/Repository entities. (See *Chapter 5, "Command Dictionary"* for a description of the /REVISE_DATE qualifier.)
- Information produced with the /AUDIT qualifier is not examined during Oracle CDD/Repository node comparisons.
- The /NOACTION qualifier has no effect on the /AUDIT qualifier. That is, if you have suppressed the execution of action lines with the /NOACTION qualifier, the remark you supply with /AUDIT is still written to the Oracle CDD/Repository history file.

4.5.1. /AUDIT Qualifier

Oracle CDD/Repository maintains a history list that includes the date and time that an Oracle CDD/Repository definition was accessed and an optional remark that you supply to document the access. To insert a remark in the Oracle CDD/Repository history list when MMS accesses an Oracle CDD/Repository definition, use the /AUDIT qualifier after the caret in the Oracle CDD/Repository specification. Follow the /AUDIT qualifier with a quoted string that contains the remark to be inserted in the Oracle CDD/Repository 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 Oracle CDD/Repository history list for the specified definition. You must place the /AUDIT qualifier after the caret character; separate the qualifier from the remark with an equal sign or colon. You cannot use /AUDIT on the MMS command line.

MMS also provides the default macro CDDFLAGS. This macro is defined to be the null string, but you can redefine it so the same remark is written to the history file for all accesses to Oracle CDD/Repository entities.

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 Oracle CDD/Repository, it writes the string that is the value of CDDFLAGS into the history file.

4.5.2. /CHANGE and /DEFINE Qualifiers

Prior to CDD/Repository Version 4.3, commonly known as Data Management Utility (DMU), the software propagated revision dates between dependent entities. For example, when the CDD CHANGE command modified a field, the revision dates for records containing that field were also updated. Entities

created under DMU continue to be updated in this manner. In CDD/Repository Version 4.3 and later, commonly known as Common Dictionary Operator (CDO), the software no longer propagates these new revision dates. The /CHANGE qualifier specified on an Oracle CDD/Repository entity directs MMS to also examine the revision times of the nested components.

The CDD DEFINE command creates a new version of an Oracle CDD/Repository entity. However, this does not revise the dependent entities that are using old versions of the entity. Oracle CDD/Repository attaches a "message" to the dependent entities, so you can find and update them when appropriate. If the /DEFINE qualifier is specified on an Oracle CDD/Repository entity, MMS treats the entity as "new" if such a message is attached to it.

The defaults for these qualifiers are /NOCHANGE and /NODEFINE. The CDDFLAGS macro can be used to change these defaults, as shown in the following examples. An explicit qualifier on an entity always overrides these defaults.

In this example, the source file includes a record definition from the Oracle CDD/Repository:

```
A.OBJ : A.PAS, CDD$TOP.B.C.D.E^/CHANGE/AUDIT="Example"
   $(PASCAL) $(PFLAGS) $(MMS$SOURCE)
```

The following is equivalent to the first example:

```
CDDFLAGS = /CHANGE/NODEFINE/NOAUDIT
A.OBJ : A.PAS, CDD$TOP.B.C.D.E^/AUDIT="Example"
   $(PASCAL) $(PFLAGS) $(MMS$SOURCE)
```

The following example demonstrates rebuilding a record if there are new versions of its fields.

```
CDD$TOP.REC1^ : CDD$TOP.REC1^/DEFINE REC1.CDO
   dictionary operator define record ...
```

Table 4.1, "Summary of Oracle CDD/Repository Qualifiers" offers a brief summary of the /CHANGE and /DEFINE qualifiers.

Table 4.1. Summary of Oracle CDD/Repository Qualifiers

Qualifier	Definition
/CHANGE	MMS examines the nested components of the named entity and uses the most recent revision time as the revision time for the entity.
/DEFINE	MMS treats the named entity as "new" if a message is attached to it. See the documentation for the Oracle CDD/Repository DEFINE command and the CDD$CHECK_MESSAGES routine.

4.6. Using MMS with SCA

When you specify the /SCA_LIBRARY qualifier, MMS generates an SCA library during the build process. *Example 4.6, "Using MMS with the /SCA_LIBRARY Qualifier"* demonstrates how to use MMS with the /SCA_LIBRARY qualifier.

Example 4.6. Using MMS with the /SCA_LIBRARY Qualifier

```
$ SET DEFAULT [SYSTEM1]
$ SCA CREATE LIB [.SCALIB] ❶
%SCA-S-NEWLIB, SCA Library created in DISK1$:[SYSTEM1.SCALIB]
%SCA-S-LIB, your SCA Library is DISK1$:[SYSTEM1.SCALIB]
```

```
$
$ TYPE SCA.MMS ❷
PROG.EXE : PROG.OBJ
PROG.OBJ : PROG.C
$
$ TYPE PROG.C ❸
main ()
   {
    int total;
    total = 2 + 2;
   }
$
$ MMS/SCA_LIBRARY/DESCRIPTION=SCA PROG.EXE ❹
CC /NOLIST/OBJECT=PROG/ANALYSIS_DATA=PROG PROG.C
mms$scalib = F$TRNLNM( "SCA$LIBRARY")
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "SCA$LIBRARY:" .NES.-
 "SCA$LIBRARY:" THEN mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "SCA$LIBRARY:" .NES.-
 "SCA$LIBRARY:" .AND. mms$scalib .NES. "SCA$LIBRARY:" THEN
 mms$scasetlib = 3
IF F$SEARCH("SCA$LIBRARY:SCA$EVENT.DAT") .EQS. "" THEN
 mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN SCA CREATE LIBRARY SCA$LIBRARY:
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN SCA SET LIBRARY SCA$LIBRARY:
$ SCA LOAD PROG ❺
%SCA-S-LOADED, module PROG loaded
%SCA-S-COUNT, 1 module loaded (1 new, 0 replaced)
IF mms$scasetlib THEN SCA SET LIBRARY 'mms$scalib'
LINK /TRACE/NOMAP/EXEC=PROG PROG.OBJ
$
```

Key to the example:

❶ Set the default to your system directory, and initialize the SCA library. A side effect of initializing the library is that the logical name SCA$LIBRARY is now defined.

❷ This is the MMS description file describing the system dependencies.

❸ This is the source code file that MMS uses to compile and link the target and sources.

❹ MMS/SCA_LIBRARY is invoked with the target PROG.EXE specified.

❺ SCA loads the SCA data file into the SCA library.

4.7. Other Objects

MMS has built-in rules for establishing the date and time of sources and targets, if they are OpenVMS files, CMS elements, library elements, or CDD entities. For any other kind of object, rules must be supplied in the description file to instruct MMS how the date and time of this object can be determined.

Nonstandard objects referenced in a dependency list must be identified as such by following every reference by a grave accent (`) and a name used to uniquely define the kind of object.

The following example demonstrates the use of this in a simple dependency list:

```
MYDOC.TXT DEPENDS_ON CHAPTER1‘DOCTYPE, CHAPTER2‘DOCTYPE
```

```
... ! actions
```

You must supply a rule in the description file named `kind_GETTIME, which enables MMS to determine the date and time associated with these objects. The action lines associated with this rule must write the required date and time to SYS$OUTPUT (in a valid OpenVMS date and time format). Within these action lines, the macro "$<" is used to reference the name of the object.

For example, suppose CHAPTER1 and CHAPTER2 represent OpenVMS files whose type is .DOC:

```
‘DOCTYPE_GETTIME :
WRITE SYS$OUTPUT F$FILE_ATTRIBUTES("$<.DOC","RDT")
```

The date and time associated with the objects CHAPTER1 and CHAPTER2 is the creation date and time of the OpenVMS files CHAPTER1.DOC and CHAPTER2.DOC.

You can also include the `kind_FIRST rule, whose actions will be performed before the first `kind_GETTIME actions. This provides a mechanism for performing any initialization steps required for the `kind_GETTIME actions to function correctly. Note that MMS uses a separate subprocess for each object kind that has a `kind_FIRST rule.

For example, suppose CHAPTER1 and CHAPTER2 actually reside in "disk:[directory]:"

```
‘DOCTYPE_FIRST :
   SET DEFAULT disk:[directory]
‘DOCTYPE_GETTIME :
   WRITE SYS$OUTPUT F$FILE_ATTRIBUTES("$<.DOC","CDT")
```

The date and time associated with the objects CHAPTER1 and CHAPTER2 is the creation date and time of the OpenVMS files "disk:[directory]CHAPTER1.DOC" and "disk:[directory]CHAPTER2.DOC."

Because the previous example uses OpenVMS files, precisely the same effect could be achieved simply as follows:

```
MYDOC.TXT DEPENDS_ON disk:[directory]CHAPTER1.DOC,-
                     disk:[directory]CHAPTER2.DOC
... ! actions
```

The following example shows rebuilding based on the timestamp field of an SQL object. The ‘SQL_first target allows MMS to run SQLDBA only once to get the dates of the SQL objects. MMS executes the action lines in a second subprocess.

```
all : emptab.txt invtab.txt
write sys$output "Build complete"
emptab.txt : emptab‘SQL
write sys$output "Fetching $* from SQL into $@"
!...
invtab.txt : invtab‘SQL
write sys$output "Fetching $* from SQL into $@"
!...
invtab‘SQL : invtab.sqlsrc
write sys$output "Using $< to rebuild SQL table $*"
!...
TIMESTAMP_IN_VMS_FORMAT = to_char(to_date(max(timestamp),-
’YYYY-MM-DD:HH24:MI:SS’),’YYYY-MM-DD:HH24:MM:SS’)
‘SQL_first : ! start the SQLDBA utility
sqldba mode=line
```

```
connect system/manager
‘SQL_gettime : ! a request sent to SQLDBA
select $(TIMESTAMP_IN_VMS_FORMAT) from user_objects
where object_name=’$<’;
‘SQL_last : ! exit the SQLDBA utility
exit
```

Chapter 5. Command Dictionary

This section describes MMS command syntax and contains a detailed description of the MMS qualifiers. The qualifiers are arranged in alphabetical order.

MMS

MMS — The MMS command invokes the Module Management System. By default, it searches in your current directory for the description file DESCRIP.MMS. If DESCRIP.MMS does not exist, MMS searches for a description file named MAKEFILE., then for a file named *target-name*.MMS (see *Section 1.2, "Invoking MMS"* and the /DESCRIPTION qualifier for more information).

Format

`MMS` [/qualifier …] [target, …]

Qualifiers	**Defaults**
/[NO]ACTION	/ACTION
/CHANGED=(source1, source2,...)	See text
/[NO]CHECK_STATUS	/NOCHECK_STATUS
/[NO]CMS	/NOCMS
/[NO]DESCRIPTION[=filespec...]	/DESCRIPTION[=filespec...]
/[NO]EXTENDED_SYNTAX	/NOEXTENDED_SYNTAX
/[NO]FORCE	/NOFORCE
/[NO]FROM_SOURCES	/NOFROM_SOURCES
/GENERATE [/qualifier[/...]] source_filespec[,...]	See text
/HELP[="topic"]	See text
/IDENTIFICATION	See text
/[NO]IGNORE[=options]	/NOIGNORE
/INTERFACE[=options]	/INTERFACE=CHARACTER_CELL
/[NO]LIST[=filespec]	/NOLIST
/[NO]LOG	/NOLOG
/MACRO=filespec | "macro"	See text
/OUTPUT=filespec	See text
/[NO]OVERRIDE	/NOOVERRIDE
/[NO]REVISE_DATE	/NOREVISE_DATE
/[NO]RULES[=filespec]	/RULES[=filespec]
/[NO]SCA_LIBRARY[=library-name]	/NOSCA_LIBRARY
/[NO]SKIP_INTERMEDIATE	/NOSKIP_INTERMEDIATE
/[NO]VERIFY	/VERIFY

Parameters

/qualifier

MMS qualifiers modify the MMS command. You can place qualifiers anywhere on the command line after the MMS command. The notation (D) following a qualifier indicates the default form.

You can abbreviate all MMS qualifiers and their parameters, but you must be sure that the abbreviations are unique so they will not be confused with other command-line interface (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.

target

The name of a target, which can be either an OpenVMS file specification, a logical name, a library specification enclosed in quotes, or a mnemonic name.

Unless you use the /NODESCRIPTION qualifier on the command line, you need not type the qualifiers and targets you want to use. MMS assumes default qualifiers and updates the first target in the description file whenever you type the MMS command, or if you specified a target on the command line, MMS updates the target itself.

Qualifiers

/ACTION (D)
/NOACTION

Controls whether MMS executes the action lines in a description file. These qualifiers affect only the execution of action lines, not the behavior of MMS.

The /ACTION qualifier displays action lines as they are invoked. MMS does not display any information on dependencies.

If you specify /NOACTION, MMS does not execute the action lines, but instead writes them to an output file (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 (see the description of the /OUTPUT qualifier).

/NOACTION overrides the Silent action-line prefix (@) described in *Section 2.6.6, "Silent Prefix (@)"*. Note that the $(MMS) reserved macro is executed even if you specify /NOACTION. Therefore, you can see what actions MMS would have executed in the subprocess. See *Section 3.3, "Invoking MMS from a Description File"* for information about the $(MMS) macro.

/NOACTION does not affect the /AUDIT qualifier that you can provide with references to CDD records. That is, if you suppress the execution of action lines with the /NOACTION qualifier, the remark you supply with /AUDIT is still written to the CDD history file. The /AUDIT qualifier and CDD records are described in *Section 4.5, "Accessing Definitions in Oracle CDD/Repository"*.

/CHANGED=(source1, source2,...)

Directs MMS to treat only the specified sources as having been changed, regardless of their actual modification times. No date checking is performed at all; MMS simply rebuilds any

targets that depend on one or more of the specified sources. This qualifier affects the behavior of MMS but not the execution of action lines.

/CHECK_STATUS
/NOCHECK_STATUS (D)

Controls 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.

When you specify the /CHECK_STATUS qualifier, MMS checks whether a target is up-to-date by determining whether any actions would be executed if the /ACTION qualifier was 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.

/CHECK_STATUS has precedence over both the /ACTION and /REVISE_DATE qualifiers if they appear on the same command line. In this case, 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)

Controls whether MMS looks for source files, description files, and include files in the current default CMS library as well as in the specified directories. CMS must be installed on your system. See *Section 4.2, "Using MMS with CMS"* 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. For information on using the /CMS qualifier with the description-file generator, refer to the /GENERATE qualifier.

When you specify the /CMS qualifier and the source in the CMS library is newer, MMS fetches it from the CMS library. If the source in the CMS library is older, MMS instead uses the source in the specified directory.

/CMS also directs MMS to look in the current default CMS library for a description file and any files included with the .INCLUDE directive, or specified with the /RULES qualifier. If MMS does not find a description file in either the specified directory or the current default CMS library, it aborts execution.

The /CMS qualifier also directs MMS to apply CMS built-in rules where appropriate. (See *Section C.9, "Built-In Rules for CMS Access"* for information about CMS built-in rules.)

The /NOCMS qualifier directs MMS not to look in the current default CMS library for source files, description files, rules files, or include files. However, if any file specifications in the description file are followed by a tilde (~) to indicate specific CMS elements, MMS looks for the files in the CMS library regardless of whether /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[=filespec...](D)
/NODESCRIPTION target

Controls 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.

The filespec is an OpenVMS file specification or a logical name that identifies the description file. The default file type is .MMS. If a tilde (~) follows the file specification, MMS fetches the description file from the default CMS library even if the description file exists in the default directory. The target is an OpenVMS file specification or a mnemonic name that designates the target to be built.

When you specify more than one description file, separate the file specifications with either commas (,) or plus signs (+) and enclose them in parentheses or quotation marks.

If you use commas, the description files are processed separately and the list of files must be enclosed in parentheses. For example:

```
$ 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. For example:

```
$ MMS/DESCRIPTION="A + B"
```

You can combine separate description files with description files to be concatenated and processed as one file. For example:

```
$ MMS/DESCRIPTION=("A + B", CLEANUP)
```

This command line directs MMS to process A.MMS and B.MMS as one file, and CLEANUP.MMS as another. In this case, there are two default targets: the first one is in either A.MMS or B.MMS (depending on the contents of the two files) and the second one is 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. For example:

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

In this case, the rules for updating X, Y, and Z must appear in both description files, A.MMS and B.MMS.

If you specify a concatenated list of description files and a list of targets, the rules for updating all the listed targets must occur in the concatenated description file. For example:

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

In this case, the description file formed by the concatenation of A.MMS and B.MMS must contain the rules for updating X, Y, and Z.

If you specify the /DESCRIPTION qualifier without a file specification or if you do not specify /DESCRIPTION, MMS looks first for the default description file DESCRIP.MMS. If it cannot locate that file, it looks for one called MAKEFILE.; if it cannot find MAKEFILE., it looks for target-name.MMS. If MMS finds target-name.MMS, it does not update the first target in the description file, but instead attempts to directly update the target indicated by target-name.MMS. For example:

```
$ MMS MAIN.EXE
```

In this example, if DESCRIP.MMS and MAKEFILE. are not present, MMS looks for a file named MAIN.MMS. If MAIN.MMS exists, MMS directly processes the target you specified

on the command line, MAIN.EXE. See *Section 2.1.2, "Specifying the Target on the Command Line"* for more information.

If MMS cannot find any one of these files, it attempts to use built-in rules to build the target.

If you use the /NODESCRIPTION qualifier, you must specify a target on the command line. /NODESCRIPTION directs MMS to ignore all description files and to build the target specified on the command line.

For information on using the /DESCRIPTION qualifier with the description-file generator, refer to the /GENERATE qualifier.

/EXTENDED_SYNTAX
/NOEXTENDED_SYNTAX (D)

The /EXTENDED_SYNTAX qualifier instructs MMS to enable the extension of MMS syntax, providing for the following:

- Use of predefined MMS functions (refer to *Section 2.3.4, "Using Predefined Functions"*)
- Macro redefinition (refer to *Section 2.3.7, "Redefining Macros in a Description File"*)
- Nested macro expansion (refer to *Section 2.3.3, "Nested Macro Expansion"*)

To enable the /EXTENDED_SYNTAX qualifier in DECwindows MMS, check the Extended Syntax check box in the Build Definitions/Directives Options dialog box.

The /EXTENDED_SYNTAX qualifier will be used to include all future extensions to the MMS syntax. The inverse qualifier, the default /NOEXTENDED_SYNTAX, will continue to support the syntax of MMS Version 3.2.

/FORCE
/NOFORCE(D)

Controls whether MMS executes the action lines necessary to update one specific target. These qualifiers affect the behavior of MMS but not the execution of action lines.

When you specify the /FORCE qualifier, MMS does not check whether the target or its sources are up-to-date, but simply rebuilds the specified target by executing the action lines. MMS also executes any .FIRST and .LAST directives associated with the target.

The /FORCE qualifier is useful for quickly rebuilding a single target.

/FROM_SOURCES
/NOFROM_SOURCES (D)

Directs MMS to build a target from its sources, regardless of whether the target is already up-to-date. This qualifier affects the execution of action lines and the behavior of MMS.

When you specify the /FROM_SOURCES qualifier, 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 the /CMS and /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 uses the sources found in the specified directory.

The /FROM_SOURCES qualifier overrides the /SKIP_INTERMEDIATE qualifier.

/GENERATE [/qualifier[/...]] source_filespec[,...]

Controls whether MMS automatically generates a description file. The source_filespec specifies the source files to inspect for dependencies and from which the target will be built. The first file specified must contain the main module from which the target will be built. Otherwise, the source files can be specified in any order and wildcard characters can be used. Unless the /CMS qualifier has been specified, the description-file generator scans all the specified files for dependencies and generates an MMS description file for the target defined by the main module.

For example, if the target is built from C files in the current default directory and from Bliss files in the subdirectory [.BLISS], and the file MAIN.C contains the main module, the following command generates the description file in DESCRIP.MMS:

```
$ MMS/GENERATE MAIN.C,*.C,[.BLISS]*.BLI
```

When you specify the /GENERATE qualifier, the following additional qualifiers are valid for the MMS command:

/[NO]BUILTIN_RULES_APPLY

Controls whether MMS only includes the dependency lines for compilations in the generated description file. The qualifier /NOBUILTIN_RULES_APPLY is the default. Since action lines are not included, MMS uses the built-in rules when processing the description file. Note that action lines for the LINK command are not effected by this qualifier.

The /SWITCHES=COMPILE qualifier cannot be used when /BUILTIN_RULES_APPLY is specified.

/[NO]CMS

Controls whether MMS looks for source files and include files in the current default CMS library. CMS must be installed on your system.

If a directory-spec is associated with a file, that is the directory into which MMS fetches the file when building the target. /NOCMS is the default.

/DESCRIPTION[=filespec]

Specifies the file to which the generated description is written. When not specified, the description is written to the file DESCRIP.MMS in the current default directory.

/FMS_LIBRARY=forms-library-name

Specifies an FMS library. This qualifier defaults to MMS$FLB.FLB.

/[NO]INCLUDES

Indicates whether C include files are scanned for dependencies. This qualifier gives the same functionality as the Scan Include Files toggle button.

/LINK_LIBRARY=filespec[,...]

Specifies additional object libraries to be included in the LINK command in the description file.

/MAIN_MODULE=module-name

Specifies the main module to be included from the object library during the Link process. The main module is that defined in the first source file specified and, by default, the module name is the same name as the file. This qualifier lets you specify a module name that is different from the file name. Note that if an appropriate language-dependent module name directive is detected within the first source file, the value specified by this qualifier is ignored.

/OBJECT_LIBRARY=filespec

Specifies the object library to be included in the LINK command in the description file. Object files from the compilations are inserted into this object library.

/OPTIONS_FILE=filespec[,...]

Specifies user-written options files to be included in the LINK command in the description file.

/[NO]SHOW_DESCRIPTION_FILE

Controls whether MMS displays the current description file in the MMS description file area. When you specify the /SHOW_DESCRIPTION_FILE qualifier, if the DECwindows version of LSE is not presently invoked, MMS displays the current description file in the MMS description file area as read-only for browsing purposes. If the DECwindows version of LSE is currently invoked, MMS displays the description file in the MMS description file area in a modifiable LSE buffer.

The /NOSHOW_DESCRIPTION_FILE qualifier directs MMS to hide the current description file from the MMS description file area. The default qualifier is /SHOW_DESCRIPTION_FILE.

/SWITCHES=([COMPILE="/sw1[/sw2...]"] [,LINK="/sw3[/sw4...]"])

Directs MMS to include the specified qualifiers on all compile and link commands in the generated description file. This provides the same function as the Additional Linker Switches buttons and text fields in the DECwindows Motif interface.

The /SWITCHES=COMPILE qualifier cannot be used when /BUILTIN_RULES_APPLY is specified.

/TARGET=name

Specifies a name for the build target. If this qualifier is not specified, the target defaults to the first file specified. This lets you specify an executable name that is different from the file name.

/HELP["topic"]

Provides information about MMS and its qualifiers. The topic is an MMS topic on which you want information.

The /HELP qualifier displays information about MMS on your terminal. If you specify the /HELP qualifier without a topic, MMS displays general information and a list of qualifiers. To get help on a specific topic, type /HELP with an equal sign (=) and the topic. The topic must be enclosed in quotation marks. For example:

```
$ MMS/HELP="/RULES"
```

See *Section 1.3, "Getting Help"* for more information.

/IDENTIFICATION

Directs MMS to display an informational message with the version number and copyright date of the MMS image you are running. MMS does not process any description files or qualifiers; it simply displays an informational message on your screen.

You should include the version number and copyright date with any MMS problem reports you submit to VSI.

/IGNORE[=options]
/NOIGNORE (D)

Directs MMS to specify the severity levels of errors that MMS normally ignores when it executes action lines. The parameters correspond to the DCL severity levels W, E, and F. The /NOIGNORE qualifier directs MMS to abort execution when it finds an error. These qualifiers affect the execution of action lines but not the behavior of MMS.

The options field can contain the keyword WARNING, ERROR, or FATAL.

WARNING directs MMS to ignore W errors and continue processing, but to abort execution if it finds an E or F error. If you specify the /IGNORE qualifier without parameters, WARNING is the default. ERROR directs MMS to ignore both W and E errors, but to abort execution if 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.

When you specify the /IGNORE qualifier, the errors that MMS ignores are those generated by the execution of action lines. The only fatal errors MMS will not ignore are syntax errors. The /IGNORE qualifier does not stop MMS error messages from being generated or displayed. Informational messages are always displayed, regardless of whether you use the /IGNORE qualifier.

Note

Take caution when executing MMS with the /IGNORE qualifier; if errors occur during processing, the target might 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. However, instead of typing them on the command line, you include them in the description file. *Section 2.6.5, "Ignore Prefix (-)"* and *Section 2.7.2, ".IGNORE Directive"* describe the Ignore action-line prefix and the .IGNORE directive in detail.

To override the .IGNORE directive contained in a description file, type the /IGNORE[=WARNING], /IGNORE=ERROR, or /IGNORE=FATAL qualifier explicitly on the MMS command line. You cannot override the Ignore actionline prefix on the MMS command line.

/INTERFACE[=DECWINDOWS]
/INTERFACE[=CHARACTER_CELL] (D)

Controls whether MMS invokes the DECwindows Motif user interface or the character-cell interface. The CHARACTER_CELL qualifier is the default.

/LIST[=filespec]
/NOLIST (D)

Controls whether MMS writes dependencies and action lines to an output file as it processes the description file. These qualifiers affect the behavior of MMS, but not the execution of action lines.

When you specify the /LIST qualifier, MMS creates a complete listing of all dependencies, dependents, and actions that need to be processed to update the target. MMS creates this listing as it processes the description file and writes the listing to the output file, or to SYS$OUTPUT if you did not specify a file.

The /LIST qualifier is useful during the debugging of description files. You can also use /LIST in combination with the /NOACTION qualifier to display the dependency list and action lines without executing any actions.

/LOG
/NOLOG (D)

Controls whether MMS displays informational messages as it processes the description file. These qualifiers affect the behavior of MMS, but not the execution of action lines.

The /LOG qualifier directs MMS to write all informational messages to your terminal screen while it processes the description file. The /LOG qualifier is useful for debugging your description files. These messages indicate what MMS finds and what it assumes as it processes the description file. You should include these messages with any MMS problem reports you submit to VSI. To save these messages in a file, type the following:

```
$ DEFINE SYS$OUTPUT MYFILE.LOG
$ MMS/LOG
.
.
.
$ DEASSIGN SYS$OUTPUT
```

The /NOLOG qualifier prevents MMS from displaying informational messages.

However, if you specify /NOLOG/CHECK_STATUS on the same command line, MMS does display the informational message that reports the value of MMS$STATUS. (See the description of the /CHECK_STATUS qualifier for more information about MMS$STATUS.)

/MACRO=filespec | "macro", . . .

Directs MMS to add to or override the macro definitions in the description file. This qualifier affects the behavior of MMS but not the execution of action lines. The filespec is an OpenVMS file specification or a logical name that identifies a file of macro definitions. The default file type is .MMS. The macro string is a macro definition enclosed in quotation marks. The definition of the macro should always assign a value to the macro. Use the same format as for macro definitions in description files, that is, name = “string”.

With the /MACRO qualifier, you can specify a macro definition on the MMS command line. You can also specify a file of macro definitions to use in your description file. *Section 2.3, "Defining Your Own Macros"* describes 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. For example:

```
$ MMS/MACRO=("A=MAC1", MACROS)
```

/OUTPUT=filespec

Directs MMS to write action lines and output to the specified file. Error messages preceded by "%MMS" are not written to this output file, but instead are written to SYS$ERROR. The /OUTPUT qualifier affects the behavior of MMS, but not the execution of action lines.

The filespec is an OpenVMS file specification or a logical name that identifies the output file. The default file type is .LOG. If you do not specify the /OUTPUT qualifier on the MMS command line, MMS writes all action lines, messages, and output to SYS$OUTPUT.

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. Thus, you can use the file generated by /OUTPUT as a DCL command procedure.

/OVERRIDE
/NOOVERRIDE (D)

Controls the order in which MMS applies definitions when it processes macros.

These qualifiers affect the behavior of MMS, but not the execution of action lines.

When you specify the /OVERRIDE qualifier, MMS overrides the macro definitions in the description file with CLI symbol definitions. To find the macro definitions that should have precedence, MMS looks at symbols defined by the CLI assignment statement, scanning the CLI symbol table for the body of the macro. If the body of the macro is not in the CLI symbol table, MMS substitutes a null string for all invocations of the macro.

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

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

Once MMS finds a definition for a macro, it does not search those locations farther down the list for more definitions. If MMS finds more than one definition in the same location (such as on a command line), it uses the last definition it processed, unless the location is a description file. MMS issues an error message if a macro is defined more than once in a description file. The /NOOVERRIDE qualifier imposes the following order, which is the default hierarchy:

1. Command-line
2. Description file
3. Built-in

4. CLI symbol

/REVISE_DATE
/NOREVISE_DATE (D)

Controls whether MMS changes only the revision dates of all targets that need updating, or performs the update. These qualifiers affect the behavior of MMS, not the execution of action lines.

When you specify the /REVISE_DATE qualifier, MMS changes only the revision dates of targets that need updating; it does not direct MMS to execute the action lines that actually do the updating. If any files are missing,

/REVISE_DATE causes MMS to create them. If MMS cannot create a missing file, or if it cannot update the revision date of an existing file, it issues an error message.

The /REVISE_DATE qualifier is useful for reducing the number of superfluous compilations, for example, when you change only a comment line in a required file. However, /REVISE_DATE can defeat the purpose of using MMS, so use this qualifier with caution.

As it changes the revision times, MMS writes the name of the revised files to an output file (or to SYS$OUTPUT if no file is specified). If you specify the /REVISE_DATE and /NOVERIFY qualifiers, the names of revised files are suppressed. (*Section 2.7.4, ".SILENT Directive"* describes the .SILENT directive.)

Unless you specify a target on the command line, the /REVISE_DATE qualifier causes MMS to revise the first target (and its sources) in the description file. 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$STATUS. (See *Section 2.6.3, "MMS$STATUS and MMS$SEVEREST_STATUS"* for information about MMS$STATUS.)

The /REVISE_DATE qualifier has precedence over the /ACTION qualifier if they both appear on the same command line. In that case, only /REVISE_DATE is processed.

The /NOREVISE_DATE qualifier directs MMS to build the system by updating targets as necessary (as long as the /CHECK_STATUS qualifier is not specified on the same command line).

/RULES[=filespec] (D)
/NORULES

Controls whether MMS applies user-defined built-in rules and a suffixes precedence list when it builds a system. These qualifiers affect the behavior of MMS, but not the execution of action lines.

The filespec is an OpenVMS file specification or a logical name that identifies the file of user-created rules that MMS is to use. When you supply a file specification with the /RULES qualifier, MMS replaces the built-in rules it normally uses with the built-in rules and suffixes list in the file you specify.

The file specified with /RULES has precedence over the file represented by MMS$RULES.

If you specify /RULES without a file specification, MMS translates the logical name MMS$RULES to locate the user-defined built-in rules file. If MMS$RULES is not defined, MMS uses its own built-in rules.

The /NORULES qualifier prevents MMS from using its built-in rules or the suffixes precedence list. It also prevents MMS from applying user–defined rules and default macros. When you specify /NORULES, MMS applies only the dependency rules contained in the description file.

/SCA_LIBRARY[=library–name]
/NOSCA_LIBRARY (D)

Controls whether MMS generates an SCA library during the build process When you specify a library name with the /SCA_LIBRARY qualifier, MMS defines the macro $(SCALIBRARY) to be that library name. If you use /SCA_LIBRARY without specifying a library name, SCA$LIBRARY is the value of $(SCALIBRARY). If you do not specify /SCA_LIBRARY, /NOSCA_LIBRARY is the default.

The macro $(SCA) is defined to be SCA regardless of the setting of the /SCA_LIBRARY qualifier.

The macro $(MMSQUALIFIERS) contains the setting of the /SCA_LIBRARY qualifier.

When you specify the /SCA_LIBRARY qualifier, built-in rules for BASIC, BLISS-32, C, C++, COBOL, Fortran, MACRO, Pascal, PL/I (Alpha and VAX only), and SCAN change. *Table C.2, "The /SCA_LIBRARY Qualifiers Default Macros"* lists the macros that change when you specify the /SCA_LIBRARY qualifier.

Note

You might choose to defer loading modules into the SCA library until after all compilations are completed. In this case, define the default rules for compilation in your description file to be the same as the default rule provided by MMS when the /NOSCA_LIBRARY qualifier is specified. You should also include a .LAST directive, which then loads the SCA database. For example:

```
.LAST :
$(SCA) SET LIBRARY $(SCALIBRARY)
$(SCA) LOAD *
```

/SKIP_INTERMEDIATE
/NOSKIP_INTERMEDIATE (D)

Controls whether MMS builds intermediate source or target files. These qualifiers affect the behavior of MMS, but not the execution of action lines. The /SKIP_INTERMEDIATE qualifier directs MMS to determine whether a target is up-to-date without rebuilding intermediate files, unless they need to be updated.

MMS first checks the target date against the dates of its sources. If the target is newer than its sources, MMS determines that the target does not need to be rebuilt; if MMS cannot find some intermediate files, it acts as though they already exist, and skips over them to check their sources, and so on.

For example, if 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, the /SKIP_INTERMEDIATE qualifier prevents MMS from building the .OBJ file and 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.

If the target is older than its sources, MMS determines that the target does need to be rebuilt. It then ensures that all of the target's immediate sources exist; if any do not, MMS works from the bottom up by first rebuilding the missing sources, then rebuilding the target. If the sources contain include files that have changed, are located in a CMS library, or both, MMS also fetches the include files and recompiles the source files, then rebuilds the system. For example:

```
!
```

```
! SYSTEM2.MMS
!
SYSTEM2 : MAIN.EXE, MOD.EXE
MAIN.EXE : MAIN.OBJ
MAIN.OBJ : MAIN.C, DEFS1.H, DEFS2.H
MOD.OBJ : MOD.C, DEFS2.H
```

If the include file DEFS1.H changes, MMS does the following when you specify the /SKIP_INTERMEDIATE qualifier:

1. Determines that one of the target's sources is newer than the target, and that the target must be rebuilt.
2. Verifies that MAIN.OBJ depends on MAIN.C, which contains the include files DEFS1.H and DEFS2.H.
3. Fetches MAIN.C, DEFS1.H, and DEFS2.H from the CMS library and recompiles MAIN.C.
4. Because MAIN.OBJ is now newer than MAIN.EXE, MMS rebuilds MAIN.EXE.
5. Because none of the sources for MOD.EXE have changed, MMS does not need to fetch them from CMS, and target SYSTEM2 is now up-to-date.

The /NOSKIP_INTERMEDIATE qualifier directs MMS to ensure that all intermediate source files exist and are up-to-date. If any intermediate source files do not exist, MMS builds them. This is the default.

/VERIFY (D)
/NOVERIFY

Controls whether MMS displays action lines before executing them. These qualifiers affect the behavior of MMS, but not the execution of action lines.

The /VERIFY qualifier directs MMS to display each action line before executing it. MMS writes action lines either to SYS$OUTPUT or to a file you specify on the /OUTPUT qualifier.

If you specify the /REVISE_DATE qualifier in combination with the /VERIFY qualifier, MMS displays the names of files whose dates have been revised.

When you specify the /NOVERIFY qualifier, MMS 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 the /REVISE_DATE and /NOVERIFY qualifiers 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 action-line prefix and the .SILENT directive (see *Section 2.6.6, "Silent Prefix (@)"* and *Section 2.7.4, ".SILENT Directive"*, respectively). If a description file contains the .SILENT directive, to override it you must type the /VERIFY qualifier explicitly on the MMS command line.

You cannot override the Silent action-line prefix from the MMS command line.

Appendix A. Error Messages

This appendix lists messages produced by MMS. The messages are accompanied by explanations and suggested actions to recover from errors.

A.1. Message Display

MMS messages are issued on the current output device identified by the logical name SYS$OUTPUT. If you are running MMS interactively, this device is a terminal; if you are running MMS in batch mode, messages are written into the log file.

A.2. Severity Levels

The severity level of a message is included in the status message and indicates the general nature of the message.

Informational (I) messages inform you of MMS actions during the systembuilding process. You can control the display of these messages by specifying the /[NO]LOG qualifier on the command line (some informational messages are displayed regardless of whether you specify /LOG).

Warning (W) messages indicate that MMS has encountered a minor error. If the error occurs during the execution of an action line, processing stops unless you specify the /IGNORE=FATAL, /IGNORE=ERROR, or /IGNORE=WARNING qualifiers on the command line.

Fatal (F) messages indicate that MMS is terminating because of a problem that prevents it from continuing any further.

MMS also generates Success (S) or Error (E) messages.

A.3. MMS Messages

This section lists all MMS messages along with brief descriptions and recommended user actions. Items enclosed in single quotation marks indicate 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.

ABORTF, For target 'target name, returned abort severity 'severity': (foreign status is 'n').

Explanation: Execution of an action line with an associated user-defined severity directive returned a status value of fatal, error, or warning. By default, MMS aborts execution.

User Action: Correct the error in the action line.

ALINSUFFLST, Type '.type' is already in suffixes precedence list in file 'filespec', line 'n'.

Explanation: You have attempted to add a type that already exists in the suffixes precedence list. MMS moved the type from its previous position in the list to the position you specified.

User Action: Check whether this is the action you intended; if not, correct the description file.

ASDILLCHR, Action status name 'name' should not include '@' or '__' in file 'filespec', line 'n'.

Explanation: The name defined by a .ACTION_STATUS directive contains non-valid characters.

User Action: Locate and correct the problem in your description file.

ASDILLNUM, Illegal number 'status' for status value in file 'filespec', line 'n'.

Explanation: A non-valid number was used to define a status value for an .ACTION_STATUS directive.

User Action: Locate and correct the problem in your description file.

ASDMSKDEF, More than one value of mask in file 'filespec', line 'n'.

Explanation: Only one definition of mask is allowed in an .ACTION_STATUS directive.

User Action: Locate and correct the problem in your description file.

ASDMSKOTH, OTHERS is not a valid value for mask in file 'filespec', line 'n'.

Explanation: OTHERS is not a valid value for mask in an .ACTION_STATUS directive.

User Action: Locate and correct the problem in your description file.

ASDMSKSEV, 'text' is not .MASK or .severity in file 'filespec', line 'n'.

Explanation: While processing an .ACTION_STATUS directive, MMS expected a value of .MASK, .SUCCESS, .INFORMATION, .WARNING, .ERROR, or .FATAL.

User Action: Locate and correct the problem in your description file.

ASDNAMDEF, Action status name 'name' is already defined in file 'filespec', line 'n'.

Explanation: The name defined by an .ACTION_STATUS directive has already been defined by a previous directive.

User Action: Locate and correct the problem in your description file.

ASDNOTDEF, Action status 'name' is not defined.

Explanation: You tried to associate the user-defined severity directive 'name' with an action line, but a directive with this name does not exist.

User Action: Locate and correct the problem in your description file.

ASDOTHDEF, OTHERS defined for more than one severity in file 'filespec', line 'n'.

Explanation: OTHERS can only be specified as the value for one severity in an .ACTION_STATUS directive.

User Action: Locate and correct the problem in your description file.

ASDOTHVAL, Status values and OTHERS defined for 'severity', values ignored in file 'filespec', line 'n'.

Explanation: Both specific status values and OTHERS have been specified for the same severity in this .ACTION_STATUS directive. MMS ignores the specified values.

User Action: Locate and correct the problem in your description file.

ASDPREDEF, Status definition already defined for this action line.

Explanation: A user-defined severity directive is already associated with the specified action line.

User Action: Locate and correct the problem in your description file.

ASDVALDEF, Status value 'status' has been defined more than once in file 'filespec', line 'n'.

Explanation: The specified status value has already been defined by an .ACTION_STATUS directive.

User Action: Locate and correct the problem in your description file.

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.

BUILDDONE, Build completed.

Explanation: Informational message from the DECwindows User Interface (UI).

User Action: No user action required.

CDDACCERR, CDD access error on path 'path specification.'

Explanation: The Oracle CDD/Repository 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 an Oracle CDD/Repository element specification, but you did not supply a remark to be included in the Oracle CDD/Repository audit history file.

User Action: Edit the description file to remove the /AUDIT qualifier or to include a remark with it.

CDDNOTFIND, MMS cannot access CDD, or CDD is not installed.

Explanation: You are trying to access records or definitions using Oracle CDD/Repository, but either it is not installed or the dictionary is not accessible.

User Action: Oracle CDD/Repository must be installed on your system and you must have the appropriate access rights to the dictionary.

CDDNOTIME, CDD path 'name' has no time attribute.

Explanation: The Oracle CDD/Repository path specification in your description file is not associated with a revision time. Therefore, MMS cannot determine whether the Oracle CDD/Repository element is newer than your target.

User Action: You cannot use an Oracle CDD/Repository record that is not associated with a revision time. Correct the description file to specify a different Oracle CDD/Repository element.

CDDPLSERR, Error returned from Oracle CDD/Repository.

Explanation: An error occurred in the processing of your description file when MMS tried to access an Oracle CDD/Repository element. This message is preceded by a message from Oracle CDD/Repository to help you locate the error.

User Action: Correct the condition that caused the first error and try again.

CDDPRIERR, Prior severe Oracle CDD/Repository error has occurred.

Explanation: An error occurred earlier in the processing of your description file when MMS tried to access an Oracle CDD/Repository element. This message is preceded by one of the other MMS error messages that pertain to the Oracle CDD/Repository and by a message from the Oracle CDD/Repository 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 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. See the CMS documentation for more information.

CMSBADFLAGS, Error parsing CMSFLAGS string 'string-value'.

Explanation: Either the value supplied for the CMSFLAGS macro is invalid or the generation-expression associated with a reference to an element is incorrect.

User Action: Correct the command line or description file.

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.

CMSCALL, Callable CMS has returned an error.

Explanation: Callable CMS, used in conjunction with CMS libraries, has returned an error to MMS. The specific error is printed on the next line.

User Action: Refer to the Code Management System documentation for more information on the CMS error returned.

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.

CMSNOSUP, MMS cannot access CMS, or CMS is not installed.

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

User Action: CMS must already be installed on your system before you install MMS if you want access to CMS libraries.

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.

CMSSRCHLISCTX, CMS library search list context resolved.

Explanation: Informational message from MMS.

User Action: No user action required.

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, MMS is installed with support for FMS.

Explanation: You can access forms stored in 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.

DRVLISFIL, Using listing file 'file.'

Explanation: MMS is outputting dependencies and action lines to the specified file.

User Action: None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

DRVNOFMSSUP, MMS is installed without support for FMS.

Explanation: You cannot access forms stored in 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, possibly resulting 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, possibly resulting 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, possibly resulting from insufficient quotas or a mailbox problem. Check with your system manager.

EXEDELPROC, Subprocess terminated abnormally.

Explanation: The subprocess terminated unexpectedly, possibly because you used illegal commands like STOP or LOGOUT in your description file, or because the subprocess was stopped by another process.

User Action: Remove any invalid commands from the description file.

EXEDELSES, Cleanup of subprocess %X'value' failed.

Explanation: The $DELPRC system service could not delete the subprocess that was executing action lines.

User Action: Type the DCL command SHOW SYSTEM/SUB to determine whether the subprocess still exists. If it does, type the STOP command to delete it: STOP/ID='value.' If the subprocess does not still exist and this message was preceded by the message %MMS-F-EXEDELPROC, the subprocess was probably deleted by a user command such as LOGOUT. If this is the case, remove the offending command from the description file.

EXENCRE, Could not create subprocess for executing action lines.

Explanation: MMS could not create the subprocess for executing action lines.

User Action: Check your quotas, and raise them if necessary. This message could also indicate a system problem. Check with 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 identifier 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.

FILEMPTY, File 'filespec' is empty.

Explanation: Informational message from the DECwindows UI.

User Action: No user action required.

FMSNOSUPP, MMS is installed without FMS support.

Explanation: Your description file specifies a form in an FMS library, but you installed MMS without FMS support.

User Action: FMS must already be installed on your system before you install MMS if you want access to FMS forms.

FMSNOTFND, MMS cannot access FMS, or FMS is not installed.

Explanation: Your description file specifies a form in an FMS library, but you installed MMS without FMS support.

User Action: 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 FMS library access.

Explanation: You cannot use a wildcard character in the specification of an FMS form.

User Action: Correct the description file to specify the forms you want MMS to access.

GENBEGIN, MMS description file generation started.

Explanation: Success message from the Automatic Description File Generator.

User Action: No user action required.

GFBMULTACTS, Actions for 'target' are redefined in file 'filespec', line 'n.'

Explanation: More than one set of actions is defined for the specified target.

User Action: Correct description file if necessary.

GFBTYPEMIX, Illegal single/double colon rule mix for 'item' in file 'filespec', line 'n.'

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.

GMFUTURE, Time for 'filename' is in the future: 'time.'

Explanation: MMS is attempting to access a target whose creation date is later than the current time. This can occur when system clocks are not exactly synchronized.

User Action: Adjust the date of the file and try again. If necessary, adjust the system clocks.

GMLCKRTRY, File 'filename' is locked by another user; retrying...

Explanation: MMS is unable to get the modification date and time for files that are open for exclusive access by another process (for example, an .OBJ file that is currently being generated by a compilation currently in progress). MMS attempts the retry for five minutes.

User Action: Unlock the file or wait for MMS to time out.

GMRTRYEXC, File 'filename' is locked by another user; retry limit exceeded.

MMS will assume it is current.

Explanation: MMS has attempted to access a locked file; after 5 minutes, MMS assumes the file is current and continues processing.

User Action: None. This is an informational message that appears once MMS has completed its five-minute retry.

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.

GUISTOPBLD, MMS build stopped by user.

Explanation: Informational message from the DECwindows UI.

User Action: No user action required.

GUISTOPGEN, MMS description file generation stopped by user.

Explanation: Informational message from the DECwindows UI.

User Action: No user action required.

GWKACTNOUPD, Actions didn't update 'target.'

Explanation: The target has not been updated by the actions associated with it.

User Action: Correct description file if necessary.

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.

GWKCHGEXC, $(MMS$CHANGED_LIST) exceeds maximum OpenVMS string length; setting it to NULL.

Explanation: The string for the $(MMS$CHANGED_LIST) macro has exceeded the maximum OpenVMS string size. MMS sets the macro to null.

User Action: Reduce the number of sources for the target being updated.

GWKCONECT, Target 'target name' found in circular dependency.

Explanation: The specified targets are involved in a circular, or recursive, 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.

GWKFORSTS, Severity of executed command is 'severity' (foreign status is 'n').

Explanation: MMS has executed a CLI command in an attempt to update a target. The resulting foreign status value and its associated severity are displayed in this 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 it to your VSI support representative.

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, or recursive, dependency.

User Action: Revise the description file to remove circular dependencies.

GWKMACRECUR, Recursive expansion of macro 'name'; macro not expanded.

Explanation: While expanding a macro used on an action line, MMS found a previous reference to the same macro.

User Action: Verify that this is the action you intended; if not, correct your description file.

GWKMFINVCHR, Invalid character following $(in macro/function reference.

Explanation: MMS detected that the first character of a macro name used on an action line is incorrect.

User Action: Locate and correct the problem in your description file.

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.

GWKNEWNOD, Target 'target' is newer than 'source.'

Explanation: The specified source has not been modified since the target was created.

User Action: None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

GWKNOACTS, Actions to update 'target name' are unknown.

Explanation: MMS cannot determine what actions to take in updating the specified target. This message might indicate a problem with the .SUFFIXES list or with your user-defined rules. There might be no built-in rule or user-defined rule for MMS to use. The file types in the user-defined rule might not be in the .SUFFIXES list, or they might be in the wrong order.

User Action: Revise the description file. Specify the actions needed to update the target.

GWKNOPRN, There are no known sources for the current target 'target name.'

Explanation: MMS has found no sources for the current target.

User Action: Create a source file that can update the target.

GWKNOREV, Cannot update modification time for file 'filespec.'

Explanation: MMS is unable to modify the revision time of the specified file, as directed by the /REVISE_DATE qualifier on the command line, because an error occurred. A possible reason for the error is that the file's protection prohibited write access.

User Action: Correct the file protection so write access is allowed.

GWKNUMPAR, Incorrect number of function parameters for 'name' in action line.

Explanation: You specified either an incorrect number of function parameters or null parameters in the function call.

User Action: Correct the erroneous line.

GWKOLDNOD, Target 'target name' is older than 'source names.'

Explanation: The specified target is older than the specified sources, so MS will update it.

User Action: None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier.

GWKPARMIS, Parentheses mismatched for macro/function 'name' in action line.

Explanation: Either an opening or closing parenthesis is missing in the description file.

User Action: Correct the erroneous line.

GWKSRCEXC, $(MMS$SOURCE_LIST) exceeds maximum OpenVMS string length; setting it to NULL.

Explanation: The string for the $(MMS$SOURCE_LIST) macro has exceeded the maximum OpenVMS string size. MMS sets the macro to null.

User Action: Reduce the number of sources for the target being updated.

GWKUNRFUN, Unrecognized function 'name' in action line.

Explanation: You specified a nonexistent function name in the action line.

User Action: Use a predefined function name.

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, MMS 'version' Copyright © 2019 VMS Software, Inc. (VSI), Bolton, Massachusetts, USA.

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 problem reports 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.

IFDEFIGNTXT, Text after '.IFDEF text' has been ignored in file 'filespec', line 'n'.

Explanation: The .IFDEF directive takes a single macro name as its argument; all text following the argument is ignored.

User Action: Verify that this is the action you intended; if not, correct your description file.

INITERR, Error in initialization routine.

Explanation: Failure in the Automatic Description File Generator.

User Action: Collect as much information as possible, and submit it to your VSI support representative.

INTERNERR, Internal MMS Error. Please Report Error #'number.'

Explanation: An MMS internal component failed.

User Action: Collect as much information as possible, and submit it to your VSI support representative.

INVFILE, Invalid file specification: 'filespec'.

Explanation: Severe error message from the DECwindows UI.

User Action: Check your directory and file specification.

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.

LEXELSERR, Encountered .ELSE or .ELSIF without matching .IF . . . in file 'filespec', line 'n'.

Explanation: An .ELSE or .ELSIF directive has been found in the description file for which there is no corresponding .IF or .IFDEF statement.

User Action: Correct description file.

LEXFILELOOP, Include file 'filespec' is already open.

Explanation: An include file is itself included at some deeper level. If undetected, this situation would cause an infinite sequence of include files.

User Action: Remove the second level of inclusion in the file.

LEXIFERR, Encountered .ENDIF without matching .IF . . . in file 'filespec', line 'n'.

Explanation: MMS found an .ENDIF directive in your description file but no corresponding .IFDEF or .IF directive.

User Action: Correct the description file to remove the .ENDIF directive, or add the necessary .IF or .IFDEF.

LEXILLNAME, Specified target name 'target' is illegal in file 'filespec', line 'n'.

Explanation: You used incorrect syntax to indicate the target on the specified line number of the description file.

User Action: Correct the description file.

LEXLCKRTRY, File 'filespec' is locked by another user; retrying . . .

Explanation: The specified file cannot be accessed because it is being updated by another user.

User Action: No user action required.

LEXNOENDIF, Encountered end of description file when expecting .ENDIF.

Explanation: An .IFDEF or .IF is missing a matching .ENDIF directive.

User Action: Locate the problem, and correct your description file.

LEXNULLNAME, Encountered null filename in file 'filespec', line 'n'.

Explanation: MMS found a null file name in the description file.

User Action: Correct the erroneous line.

LEXRTRYEXC, File 'filespec' is locked by another user; retry limit exceeded.

Explanation: The specified file cannot be accessed because it is being updated by another user.

User Action: Repeat when file is available.

LEXTOOLONG, Line too long in file 'filespec', line 'n'.

Explanation: A long line has been detected in the description file.

User Action: Correct the erroneous line in the description file.

LEXUNEXEND, Continuation character found at end of file 'filespec', line 'n'.

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.

MAXNESTDEPTH, Maximum nesting depth for include files exceeded.

Explanation: This is an error message from the Automatic Description File Generator. This probably means that the source files being processed contain a circular, or recursive, reference (that is, a file includes itself).

User Action: Check the sources for circular references.

MBBADMODE, Unknown mode parameter 'mode number.'

Explanation: An internal MMS component failed.

User Action: Collect as much information as possible, and submit it to your VSI support representative.

MBREDEFILL, Illegal attempt to redefine macro 'macro name.'

Explanation: You attempted to redefine the specified macro in the description file. You cannot define the same macro twice in one description file. The attempt is ignored, and the original definition applies.

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

MBREDEFMAC, Macro 'macro name' redefined from 'value' to 'value'.

Explanation: A macro has been redefined in the description file.

User Action: None. This is an informational message that appears only if you have invoked MMS with the /LOG qualifier, in addition to the /EXTENDED_SYNTAX qualifier.

MISSINGBOOLOPER, Missing '.AND' or '.OR' before 'word' in file 'filespec', line 'n'.

Explanation: The specified boolean expression following an .IF directive is incorrect, possibly due to mismatched parentheses.

User Action: Locate and correct the problem in your description file.

NAMTOOLONG, Name too long: 'name'.

Explanation: This is a warning message from the Automatic Description File Generator. The name of an include file or routine is too long.

User Action: Check the source file.

NFMACRECUR, Recursive expansion of macro 'name'; macro not expanded in file 'filespec', line 'n'.

Explanation: While expanding a macro at the position specified, MMS detected a previous reference to the same macro.

User Action: Verify that this is the action you intended; if not, correct the problem in your description file.

NFMINVCHR, Invalid character following $(in macro/function reference in file 'filespec', line 'n'.

Explanation: MMS detected that the first character of the specified macro name is incorrect.

User Action: Locate and correct the problem in your description file.

NFNUMPARAMS, Incorrect number of function parameters for 'name' in file 'filespec', line 'n'.

Explanation: You specified either an incorrect number of function parameters or null parameters in the function call.

User Action: Correct the erroneous line.

NFPARMISMAT, Parentheses mismatched for macro/function 'name' in file 'filespec', line 'n'.

Explanation: Either an opening or closing parenthesis is missing in the description file.

User Action: Correct the erroneous line.

NFUNRECFUNC, Unrecognized function 'name' in file 'filespec', line 'n'.

Explanation: You specified a nonexistent function name in the description file.

User Action: Choose a predefined function.

NOACCESS, Unable to access file 'file.'

Explanation: This message is followed by one or more messages describing why the file could not be accessed.

User Action: Modify the file protection of the inaccessible file.

NOCHAN, No more channels available.

Explanation: This is a failure in the Automatic Description File Generator. There are not enough I/O channels available.

User Action: Increase the 'Open file' quota or the SYSGEN CHANNELCT parameter.

NOFILEGIVEN, No appropriate files found for scanning.

Explanation: This is an error message from the Automatic Description File Generator. No files of a recognized language type were specified for processing.

User Action: Check the list of files supplied.

NOFILENAME, No filename in file specification 'filespec'.

Explanation: This is a warning message from the Automatic Description File Generator. An include file has no name.

User Action: Check the source.

NOLIBSPECDBL, Library module specifications not allowed as targets in double colon rules: 'filespec.'

Explanation: You used a library module specification as a target in a double-colon dependency rule.

User Action: Rewrite the dependency as a single-colon dependency using the library module specification or use only the library file name in your double-colon dependency rule. You can write the preferred single-colon syntax by using library module specifications. For example:

```
UTIL(MOD1) : MOD1.OBJ
LIBR UTIL.LIB MOD1.OBJ
UTIL(MOD2) : MOD2.OBJ
LIBR UTIL.LIB MOD2.OBJ
```

The following dependency rule is correct for the double colon use:

```
UTIL.OLB :: MOD1.OBJ
LIBR UTIL.LIB MOD1.OBJ
UTIL.OLB :: MOD2.OBJ
LIBR UTIL.LIB MOD2.OBJ
```

NOMACFIL, Cannot open macro file 'filespec.'

Explanation: You specified either an illegal or a nonexistent file in the command-line macro definitions.

User Action: Create the file, or correct the file specification.

NOOUTFIL, Cannot open output file 'filespec.'

Explanation: MMS failed to create the output file.

User Action: Verify that the file specification is legal, check your disk quota, or check the protection of an existing file of the same name as the output file.

NOREVIEW, No module to review.

Explanation: Informational message from the DECwindows UI.

User Action: No user action required.

NORIGHTPAREN, Missing right parenthesis before 'text' in file 'filespec', line 'n'.

Explanation: The specified boolean expression following a .IF directive is incorrect.

User Action: Locate and correct the problem in your description file.

NORMAL, MMS description file generation completed.

Explanation: Success message from the Automatic Description File Generator.

User Action: No user action required.

NOSOURCES, No input source file specified.

Explanation: Severe error message from the DECwindows UI.

User Action: Specify the input source file.

NOSTATUS, Unable to set MMS$STATUS and MMS$SEVEREST_STATUS.

Explanation: MMS received an error from OpenVMS when trying to set the symbols MMS$STATUS and MMS$SEVEREST_STATUS. This error can occur if you have exceeded the available space for symbols defined by your process, or if symbol scope is set to noglobal.

User Action: Remove some of your symbols, or have the system manager change the SYSGEN parameter CLISYMTBL, or set the symbol scope to global.

NOSUFFLST, .SUFFIXES_[AFTER |BEFORE] directive does not specify two or more types in file 'filespec', line 'n'.

Explanation: The list of types following either an .SUFFIXES_AFTER or .SUFFIXES_BEFORE directive should contain at least two values. MMS ignores the directive.

User Action: Verify that this is the action you intended; if not, locate and correct the problem in your description file.

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 it specifies a target.

NOTBOOLOPER, 'word' is not a valid boolean operator in file 'filespec', line 'n'.

Explanation: MMS expected a boolean operator (.EQ, .NE, .GE, .LE, .GT, or .LT) in a boolean expression following the .IF directive. This problem may be due to mismatched parentheses.

User Action: Locate and correct the problem in your description file.

NOTINSUFFLST, Type '.type' is not in suffixes precedence list in file 'filespec', line 'n'.

Explanation: This is due to one of the following conditions:

- The first type specified as an argument to a .SUFFIXES_AFTER or .SUFFIXES_BEFORE directive is not in the suffixes precedence list.

  Remaining types will be appended to existing list.

- A file type specified as an argument to the .SUFFIXES_DELETE is not in the suffixes precedence list. The type cannot be deleted.

User Action: Locate and correct the problem in your description file.

OPERNOTANDOR, Operator must be '.AND' or '.OR', not 'word', in file 'filespec', line 'n'.

Explanation: MMS expected the operator .AND or .OR following an .IF directive in a boolean expression. This problem may be due to mismatched parentheses.

User Action: Locate and correct the problem in your description file.

OUTPUTERR, Error in MMS output processing.

Explanation: Failure in Automatic Description File Generator.

User Action: Collect as much information as possible, and submit it to your VSI support representative.

RULDISCNOTYP, Rule '.type1.type2' discarded as '.type' is not in suffixes precedence list.

Explanation: MMS discarded the rule for the specified type since it is not in the suffixes precedence list.

User Action: Locate and correct the problem in your description file.

RULDISCNOTYPS, Rule '.type1.type2' discarded as types is not in suffixes precedence list.

Explanation: MMS discarded the rule for the specified type(s) since they are not in the suffixes precedence list.

User Action: Locate and correct the problem in your description file.

RULDISCORDER, Rule '.type1.type2' discarded as types incorrectly ordered in suffixes precedence list.

Explanation: MMS discarded the rule for the specified type(s) since they are not in the correct order in the suffixes precedence list.

User Action: Locate and correct the problem in your description file.

SCANNING, Scanning 'filespec'.

Explanation: Information message from the Automatic Description File Generator. The specified file is being scanned for dependencies.

User Action: No user action required.

SKIPPING, Skipping 'filespec'.

Explanation: Information message from the Automatic Description File Generator. The specified file is not being scanned because it contains no dependencies.

User Action: No user action required.

SOURCERR, Error in source file processing.

Explanation: Failure in Automatic Description File Generator.

User Action: Collect as much information as possible, and submit it to your VSI support representative.

UNABLOPEN, Cannot open output file 'filespec'.

Explanation: Error message from the Automatic Description File Generator. Unable to open the file to which the new MMS description should be output; output will be to SYS$OUTPUT instead.

User Action: Check the description file 'filespec'.

UNACCESS, Unable to access file 'filespec'.

Explanation: Warning message from the Automatic Description File Generator. Not able to access the specified file.

User Action: Check the protection of 'filespec'.

UNEXPLEFTPAREN, Unexpected left parenthesis before 'text' in file 'filespec', line 'n'.

Explanation: MMS encountered an erroneous left parenthesis following an .IF directive in a boolean expression. This problem may be due to mismatched parentheses.

User Action: Locate and correct the problem in your description file.

UNEXPRIGHTPAREN, Unexpected right parenthesis before 'text' in file 'filespec', line 'n'.

Explanation: MMS encountered an erroneous right parenthesis following an .IF directive in a boolean expression. This problem may be due to mismatched parentheses.

User Action: Locate and correct the problem in your description file.

UNEXPTYP, Unexpected file type 'filespec'.

Explanation: Warning message from the Automatic Description File Generator. The specified file is a type that can only be referenced as an include file.

User Action: Check the 'filespec'.

UNKNOWNTYP, Unknown file type 'filespec'.

Explanation: Warning message from the Automatic Description File Generator. The specified file type is not associated with a recognized language.

User Action: Check the 'filespec'.

USETOWRITE, Writing output to SYS$OUTPUT.

Explanation: Information message from the Automatic Description File Generator. MMS has been unable to open the file to which the new MMS description should be output; output will be to SYS $OUTPUT instead.

User Action: No user action required.

UTLALLOCFAIL, Failed to allocate memory for dynamic data structures.

Explanation: An MMS call to obtain more virtual memory failed. Either your description file is too large or a system service failed unexpectedly.

User Action: Try trimming your description file. If this fails, consult your system manager about increasing the size of virtual address space available to your system processes. If this fails, contact your VSI support representative.

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 it to your VSI support representative.

WILDDIRCMS, Wildcarding not permitted in [directory-spec] when using CMS.

Explanation: Warning message from the Automatic Description File Generator. A [directory-spec] associated with a CMS element defines the directory to which the element is to be fetched and must not contain wildcards.

User Action: Amend [directory-spec].

Appendix B. MMS and *make* Comparisons

This appendix briefly compares the characteristics of MMS and the UNIX make utility. It is designed to ease the transition to MMS for users already familiar with make.

Because OpenVMS and UNIX are very different operating systems, certain system-imposed changes were necessary to provide the features of make on OpenVMS systems. 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. If it cannot find DESCRIP.MMS or MAKEFILE., it looks for target-name.MMS.
- In the target or source line of a dependency rule, there must be at least one space or tab on either side of the colon or double colon that separates the list of targets from the list of sources. The space or tab prevents MMS from trying to interpret the colon or colons as part of an OpenVMS file specification.
- With MMS, you can use commas as well as spaces to separate the elements in a list of targets or sources.
- You can use either a number sign (#) or an exclamation point (!) as a comment character. On target or source lines, as well as on blank lines that separate dependency rules, you can use the number sign and the exclamation point interchangeably; however, on action lines, you can use only the exclamation point to indicate a comment.
- In MMS, subprocesses are not executed independently of one another. Therefore, it is possible to define logical names, change directories, and in general manipulate the subprocess environment at will.
- The dummy target .PRECIOUS, found in make, is not implemented in MMS.
- When invoking a macro in MMS, you must enclose the macro name in parentheses. That is, $(A) is a legal invocation of an MMS macro, but $A is not.
- MMS action lines can begin with either a space or a tab. MMS assumes that any line that begins with a space or tab is an action line unless the preceding line ends with a continuation character.
- MMS has different built-in rules from those of make. See *Table C.6, "MMS Built-In Rules"* 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 an OpenVMS library module, you can use the question mark (?) wildcard character as a synonym for the percent sign (%) wildcard character.
- MMS has an optional format for dependency rules:

```
PROG.OBJ DEPENDS_ON PROG.C
```

```
UTIL.LIB ADDITIONALLY_DEPENDS_ON MOD1.OBJ
```

In this format, you can use DEPENDS_ON in place of the colon, and ADDITIONALLY_DEPENDS_ON in place of the double colon.

For compatibility with make, MMS provides alternate 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.

- You can place a colon after the name of a directive. For example, you can specify either .SILENT or .SILENT : in a description file.

- The period preceding the .INCLUDE directive is optional.

- You can abbreviate MMS special macros to two characters (see *Table C.3, "MMS Special Macros"*).

The make utilities, such as NMAKE and GNU Make, continue to evolve with additional features and functions. The /EXTENDED_SYNTAX qualifier enables the use of predefined functions within the MMS product, as described in *Section 2.3.4, "Using Predefined Functions"*.

In addition, the /EXTENDED_SYNTAX qualifier enables you to redefine macros within the same description file for a series of conditions. The default is the /NOEXTENDED_SYNTAX state.

Appendix C. MMS Built-In Features

This appendix contains tables of MMS built-in features, including the default macros, suffixes-precedence list, and built-in rules. *Chapter 2, "MMS Description Files"* contains detailed information about how these three features work together in MMS.

The tables in this appendix are as follows:

- *Table C.1, "MMS Default Macros"* lists the default macros.
- *Table C.2, "The /SCA_LIBRARY Qualifiers Default Macros"* lists the changed default macros when you use the /SCA_LIBRARY qualifier.
- *Table C.3, "MMS Special Macros"* lists the special macros.
- *Table C.4, "Suffixes-Precedence List"* contains the suffixes-precedence list.
- *Table C.5, "MMS Directives"* lists and describes the directives used in a description file.
- *Table C.6, "MMS Built-In Rules"* contains the standard built-in rules.

Section C.7, "Built-In Rules for Library Files" describes the built-in rules for accessing OpenVMS libraries.

Section C.8, "Built-In Rules for the /SCA_LIBRARY Qualifier" lists the built-in rules that change when you use the /SCA_LIBRARY qualifier. *Section C.9, "Built-In Rules for CMS Access"* includes the built-in rules for accessing CMS library elements.

For information on using MMS to create and access elements in OpenVMS libraries, see *Section 4.1, "Creating and Accessing Files in OpenVMS Libraries"*; in CMS libraries, see *Section 4.2, "Using MMS with CMS"*.

C.1. Default Macros

MMS uses the default macros shown in *Table C.1, "MMS Default Macros"* to build your system if none are specified or redefined.

Table C.1. MMS Default Macros

Macro	Definition
ANLFLAGS	/OUTPUT=$(MMS$TARGET_NAME).ANL
AS	MACRO
BASFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
BASIC	BASIC
BFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
BLIBFLAGS1	/NOLIST
BLISS	BLISS

Macro	Definition
BLISS16	BLISS/PDP11
CC	CC
CDDFLAGS	null string
CFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
CLDFLAGS	null string
CMS	CMS
CMSCOMMENT	null string
CMSFLAGS	/GEN=$(MMS$CMS_GEN)
COBFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
COBOL	COBOL
CORAL	CORAL
CXX	C++
CXXFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
CORFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
DBLFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
DIBOL	DIBOL
FFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
FMS	FMS
FMSFLAGS	/REPLACE
FORMS	FORMS
FORMS_EXOBJ_FLAGS	/NOLIST/OUTPUT=$(MMS $TARGET_NAME).OBJ
FORMS_TRANS_FLAGS	/NOLIST/OUTPUT=$(MMS $TARGET_NAME).FORM
FORT	FORTRAN
F90	F90
F90FLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
LIBR	LIBRARY
LIBRFLAGS	/REPLACE
LINK	LINK
LINKFLAGS	/TRACE/NOMAP/EXEC=$(MMS $TARGET_NAME).EXE
MACRO	MACRO

Macro	Definition
MFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
MSGFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
PASCAL	PASCAL
PENVFLAGS	/NOLIST
PFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
PLI	PLI
PLIFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
RALFLAGS	null string
RALLY	RALLY
RFLAGS	/OUTPUT=$(MMS$TARGET_NAME).OBJ
RPG	RPG
RPGFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
RUNOFF	RUNOFF
SCA	SCA
SCAFLAGS	/LOG
SCALIBRARY1	Not defined
SCAN	SCAN
SCANFLAGS1	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ
SQL$PRE	SQL$PRE
SQL$MOD	SQL$MOD
SQLFLAGS	null string

C.2. Default Macro Changes with the /SCA_LIBRARY Qualifiers

Table C.2, "The /SCA_LIBRARY Qualifiers Default Macros" lists the default macro changes with the /SCA_LIBRARY qualifier.

Table C.2. The /SCA_LIBRARY Qualifiers Default Macros

Macro	Definition
SCA	SCA
SCALIBRARY	Library name from the /SCA_LIBRARY qualifier
BASFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA

Macro	Definition
BFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA
BLIBFLAGS	/NOLIST/ANALYSIS_DATA=$(MMS $TARGET_NAME).ANA
CFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA
COBFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA
CXXFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA
FFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA
F90FLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/XREF_DATA=$(MMS $TARGET_NAME).ANA
MFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA
PFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA
PLIFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA
SCANFLAGS	/NOLIST/OBJECT=$(MMS $TARGET_NAME).OBJ/ANALYSIS_DATA= $(MMS$TARGET_NAME).ANA

C.3. Special Macros

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

Table C.3. MMS Special Macros

Macro	Symbol	Meaning
MMS$TARGET	$@	Expands to the mnemonic name or the complete file specification of the target currently being updated.
MMS$TARGET_NAME	$*	Expands to the mnemonic name or the file name (excluding the file type) of the target

Macro	Symbol	Meaning
		being updated. The device, directory, and node information are included.
MMS$TARGET_SPEC	$>	In an OpenVMS library target dependency rule, this expands to the target OpenVMS library specification. Otherwise, it expands to the complete specification of the target being updated.
MMS$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	$%	Expands to the name of a module in an OpenVMS library and its file name, including the file type (see *Section 4.1, "Creating and Accessing Files in OpenVMS Libraries"*).
MMS$CMS_ELEMENT	$<	Expands to the implicit CMS element specification (if the source file is a CMS element).
MMS$CMS_GEN	$&	Expands to the CMS generation specified by the source file (if the source is a CMS element).
MMS$CMS_LIBRARY	$@	Expands to the CMS library specification (if the source is a CMS element).

C.4. Suffixes-Precedence List

Table C.4, "Suffixes-Precedence List" provides the MMS suffixes-precedence list.

Table C.4. Suffixes-Precedence List

.SUFFIXES	.ANL .EXE .OLB .MLB .HLB .TLB .FLB .OBJ .FORM .BLI .B32 .C .COB .F90$MOD .F90 .F77 .FOR .F .BAS .B16 .PLI .PEN .PAS .MAC .MAR .M64 .CLD .MSG .COR .DBL .RPG .SCN .IFDL .RBA .RC .RCO .RFO .RPA .SC .SCO .SFO .SPA .SPL .SQLADA .SQLMOD .RGK .RGC .MEM .RNO .HLP .RNH .L32 .REQ .R32 .L16 .R16 .TXT .H .FRM .MMS .DDL .COM .DAT .OPT .CDO .SDML .ADF .GDF .LDF .MDF .RDF .TDF .CXX .LIB .ADF~1

	.ANL~ .B16~ .B32~ .BAS~ .BLI~ .C~ .CDO~ .CLD~ .COB~ .COM~ .COR~ .DAT~ .DBL~ .DDL~ .FOR~ .F90~. F77~ .F~ .FRM~ .GDF~ .HLP~ .H~ .IFDL~ .LDF~ .MAC~ .MAR~ .M64~ .MDF~ .MMS~ .MSG~ .OPT~ .PAS~. PLI~ .R16~ .R32~ .RBA~ .RC~ .RCO~ .RDF~ .REQ~ .RFO~ .RGC~ .RNH~ .RNO~ .RPA~ .RPG~ .SC~ .SCN~ .SCO~ .SDML~ .SFO~ .SPA~ .SPL~ .SQLADA~ .SQLMOD~ .TDF~ .TXT~ .CXX~ .LIB~

C.5. Directives

Table C.5, "MMS Directives" lists and describes all the MMS directives.

Table C.5. MMS Directives

Directive	Function
.ACTION_STATUS	Directs MMS to apply a user-defined severity value to action lines that do not return a standard OpenVMS status value.
.DEFAULT	Indicates actions to be performed if MMS built-in rules or user-defined rules do not specify how to update a target.
.ELSE	Causes subsequent lines of a description file to be processed only if the macro for the preceding .IF, .IFDEF, or .ELSIF directives are undefined or false.
.ELSIF	Causes subsequent lines of a description file to be processed only if the related boolean expression is true.
.ENDIF	Terminates the set of lines in the description file whose processing is controlled by .IF, .IFDEF, .ELSE, or .ELSIF.
.FIRST	Indicates actions to be performed before MMS has executed any action lines to update the target.
.IF	Causes subsequent lines of a description file to be processed only if the related boolean expression is true.
.IFDEF	Causes subsequent lines of a description file to be processed only if the specified macro is defined.
.IGNORE	Causes MMS to ignore all errors generated by all action lines and continue processing the description file.
.IGNORE_ALL	Causes MMS to ignore warnings, errors, and fatal errors that occur during processing of the description file—for both action lines and with the description file itself.
.INCLUDE	Includes the specified file in the description file.
.LAST	Indicates actions to be performed after MMS has executed all the action lines that update the target.
.SILENT	Suppresses the writing of all action lines to the output file (whether to SYS$OUTPUT or to the file specified by the /OUTPUT qualifier).
.SUFFIXES	Appends a list of file types to the suffixes precedence list.

Directive	Function
.SUFFIXES_AFTER	Inserts a list of file types in the suffix precedence list after the specified first file type.
.SUFFIXES_BEFORE	Inserts a list of file types in the suffix precedence list before the specified first file type.
.SUFFIXES_DELETE	Removes a list of file types from the suffixes precedence list.

C.6. Built-In Rules

Table C.6, "MMS Built-In Rules" lists the sources, targets, and actions for the MMS built-in rules.

Table C.6. MMS Built-In Rules

Source	Target	Action
.B161	.OBJ	$(BLISS16) $(BFLAGS) $(MMS$SOURCE)
.B321	.OBJ	$(BLISS) $(BFLAGS) $(MMS$SOURCE)
.BAS1	.OBJ	$(BASIC) $(BASFLAGS) $(MMS$SOURCE)
.BLI1	.OBJ	$(BLISS) $(BFLAGS) $(MMS$SOURCE)
.C1	.OBJ	$(CC) $(CFLAGS) $(MMS$SOURCE)
.CLD	.OBJ	SET COMMAND /OBJECT=$(MMS $TARGET_NAME)$(CLDFLAGS) $(MMS$SOURCE)
.COB1	.OBJ	$(COBOL) $(COBFLAGS) $(MMS$SOURCE)
.COR	.OBJ	$(CORAL) $(CORFLAGS) $(MMS$SOURCE)
.CXX1	.OBJ	$(CXX) $(CXXFLAGS) $(MMS$SOURCE)
.DBL	.OBJ	$(DIBOL) $(DBLFLAGS) $(MMS$SOURCE)
.EXE	.ANL	ANALYZE/IMAGE $(ANLFLAGS) $(MMS $SOURCE)
.FOR1	.OBJ	$(FORT) $(FFLAGS) $(MMS$SOURCE)
.F901	.F90$MOD	$(F90) $(F90FLAGS) $(MMS$SOURCE)
.F901	.OBJ	$(F90) $(F90FLAGS) $(MMS$SOURCE)
.F771	.OBJ	$(FORT) $(FFLAGS) $(MMS$SOURCE)
.F1	.OBJ	$(FORT) $(FFLAGS) $(MMS$SOURCE)
.FORM	.OBJ	$(FORMS) EXTRACT OBJECT $(FORMS_EXOBJ_FLAGS) $(MMS$SOURCE)
.IFDL	.FORM	$(FORMS) TRANSLATE $(FORMS_TRANS_FLAGS) $(MMS$SOURCE)
.MAC1	.OBJ	$(MACRO) $(MFLAGS) $(MMS$SOURCE)
.MAR1	.OBJ	$(MACRO) $(MFLAGS) $(MMS$SOURCE)
.M641	.OBJ	$(MACRO) $(MFLAGS) $(MMS$SOURCE)
.MSG	.OBJ	MESSAGE $(MSGFLAGS) $(MMS$SOURCE)

Source	Target	Action
.OBJ	.ANL	ANALYZE/OBJECT $(ANLFLAGS) $(MMS $SOURCE)
.OBJ	.EXE	$(LINK) $(LINKFLAGS) $(MMS$SOURCE)
.PAS1	.OBJ	$(PASCAL) $(PFLAGS) $(MMS$SOURCE)
.PAS	.PEN	$(PASCAL) /ENVIRON=$(MMS$TARGET) $(PENVFLAGS) $(MMS$SOURCE)
.PLI1	.OBJ	$(PLI) $(PLIFLAGS) $(MMS$SOURCE)
.R161	.L16	$(BLISS16) /LIBRARY=$(MMS$TARGET)
.R321	.L32	$(BLISS) /LIBRARY=$(MMS$TARGET)
.REQ1	.L32	$(BLISS) /LIBRARY=$(MMS$TARGET)
.RGC	.RGK	$(RALLY) DEFINE KEYS $(RALFLAGS) $(MMS $SOURCE) $(MMS$TARGET)
.RNH	.HLP	$(RUNOFF) $(RFLAGS) $(MMS$SOURCE)
.RNO	.MEM	$(RUNOFF) $(RFLAGS) $(MMS$SOURCE) $(BLIBFLAGS) $(MMS$SOURCE)
.RPG	.OBJ	$(RPG) $(RPGFLAGS) $(MMS$SOURCE) $(BFLAGS) $(MMS$SOURCE) $(BFLAGS) $(MMS$SOURCE)
.SC	.OBJ	$(SQLPRE) /CC /OBJECT=$(MMS$TARGET) $(SQLFLAGS) $(MMS$SOURCE)
.SCO	.OBJ	$(SQLPRE) /COBOL /OBJECT=$(MMS$TARGET) $(SQLFLAGS) $(MMS$SOURCE)
.SFO	.OBJ	$(SQLPRE) /FORTRAN /OBJECT=$(MMS$TARGET) $(SQLFLAGS) $(MMS$SOURCE)
.SPA	.OBJ	$(SQLPRE) /PASCAL /OBJECT=$(MMS$TARGET) $(SQLFLAGS) $(MMS$SOURCE)
.SPL	.OBJ	$(SQLPRE) /PLI /OBJECT=$(MMS$TARGET) $(SQLFLAGS) $(MMS$SOURCE)
.SQLADA	.OBJ	$(SQLPRE) /ADA $(SQLFLAGS) $(MMS$SOURCE)
.SQLMOD	.OBJ	$(SQLMOD) /OBJECT=$(MMS$TARGET) $(SQLFLAGS) $(MMS$SOURCE) $(SQLFLAGS) $(MMS$SOURCE)
.SCN1	.OBJ	$(SCAN) $(SCANFLAGS) $(MMS$SOURCE)

C.7. Built-In Rules for Library Files

The following example shows how 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)
The following example shows how 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)
The following example shows how to build an object library:
.OBJ.OLB
IF "’’F$SEARCH("$(MMS$TARGET)")’" .EQS. "" -
THEN $(LIBR)/CREATE $(MMS$TARGET)
$(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
.TXT.TLB
IF "’’F$SEARCH("$(MMS$TARGET)")’" .EQS. "" -
THEN $(LIBR)/CREATE/TEXT $(MMS$TARGET)
$(LIBR) $(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
The following example shows how to build an FMS library:
.FRM.FLB
IF "’’F$SEARCH("$(MMS$TARGET)")’" .NES. "" -
THEN $(FMS)/LIBRARY $(FMSFLAGS) $(MMS$TARGET) $(MMS$SOURCE)
IF "’’F$SEARCH("$(MMS$TARGET)")’" .EQS. "" -
THEN $(FMS)/LIBRARY/CREATE $(MMS$TARGET) $(MMS$SOURCE)
```

C.8. Built-In Rules for the /SCA_LIBRARY Qualifier

This section lists the changes to built-in rules when you use the /SCA_LIBRARY qualifier.

```
.BAS.OBJ :
$(BASIC) $(BASFLAGS) $(MMS$SOURCE)
   mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
   mms$scasetlib = 0
   IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:"
 THENmms$
   scasetlib = 2
   IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND.-
   mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
   IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
   mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
   IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS)
 -
$(SCALIBRARY)
   IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
   IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) ’mms$scalib’
```

```
   .BLI.OBJ :
$(BLISS) $(BFLAGS) $(MMS$SOURCE)
   mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
   mms$scasetlib = 0
   IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
   mms$scasetlib = 2
   IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND.
 -
   mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
   IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
   mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
   IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS)
 -
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.B32.OBJ :
$(BLISS) $(BFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.C.OBJ :
$(CC) $(CFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS)
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS)
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.COB.OBJ :
$(COBOL) $(COBFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
```

```
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF ((mms$scasetlib .AND. 4) .EQ. 4) THEN SPAWN/WAIT/NOSYMBOLS $(SCA) -
CREATE LIBRARY $(SCAFLAGS) $(SCALIBRARY)
IF ((mms$scasetlib .AND. 4) .EQ. 4) .OR. ((mms$scasetlib .AND. 2) .EQ. 2) -
THEN DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
.CXX.OBJ :
$(CXX) $(CXXFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM("SCA$LIBRARY")
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
SPAWN/WAIT/NOSYMBOLS $(SCA) CREATE LIBRARY $(SCAFLAGS) $(SCALIB
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
.FOR.OBJ :
$(FORT) $(FFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.F90.F90$MOD :
$(F90) $(F90FLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM("SCA$LIBRARY")
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
SPAWN/WAIT/NOSYMBOLS $(SCA) CREATE LIBRARY $(SCAFLAGS) $(SCALIB
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) IMPORT $(SCAFLAGS) $(MMS$TARGET_NAME).XREF
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
.F90.OBJ :
$(F90) $(F90FLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM("SCA$LIBRARY")
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
SPAWN/WAIT/NOSYMBOLS $(SCA) CREATE LIBRARY $(SCAFLAGS) $(SCALIB
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) IMPORT $(SCAFLAGS) $(MMS$TARGET_NAME).XREF
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
.F77.OBJ :
$(FORT) $(FFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM("SCA$LIBRARY")
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
SPAWN/WAIT/NOSYMBOLS $(SCA) CREATE LIBRARY $(SCAFLAGS) $(SCALIB
```

```
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
.F.OBJ :
$(FORT) $(FFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM("SCA$LIBRARY")
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
SPAWN/WAIT/NOSYMBOLS $(SCA) CREATE LIBRARY $(SCAFLAGS) $(SCALIB
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
.MAC.OBJ :
$(MACRO) $(MFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM("SCA$LIBRARY")
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
SPAWN/WAIT/NOSYMBOLS $(SCA) CREATE LIBRARY $(SCAFLAGS) $(SCALIB
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
.M64.OBJ :
$(MACRO) $(MFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM("SCA$LIBRARY")
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
SPAWN/WAIT/NOSYMBOLS $(SCA) CREATE LIBRARY $(SCAFLAGS) $(SCALIB
IF "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. mms$scalib .NES. "$(SCALI
DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
.MAR.OBJ :
$(MACRO) $(MFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.PAS.OBJ :
$(PASCAL) $(PFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
```

```
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.PLI.OBJ :
$(PLI) $(PLIFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.REQ.L32 :
$(BLISS)/LIBRARY=$(MMS$TARGET) $(BLIBFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.R32.L32 :
$(BLISS)/LIBRARY=$(MMS$TARGET) $(BLIBFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF (mms$scasetlib .AND. 4) .EQ. 4 THEN $(SCA) CREATE LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
IF (mms$scasetlib .AND. 2) .EQ. 2 THEN $(SCA) SET LIBRARY $(SCAFLAGS) -
$(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
IF mms$scasetlib THEN $(SCA) SET LIBRARY $(SCAFLAGS) 'mms$scalib'
.SCN.OBJ :
$(SCAN) $(SCANFLAGS) $(MMS$SOURCE)
mms$scalib = F$TRNLNM( "SCA$LIBRARY" )
mms$scasetlib = 0
IF mms$scalib .EQS. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" THEN -
mms$scasetlib = 2
IF mms$scalib .NES. "" .AND. "$(SCALIBRARY)" .NES. "SCA$LIBRARY:" .AND. -
mms$scalib .NES. "$(SCALIBRARY)" THEN mms$scasetlib = 3
IF F$SEARCH("$(SCALIBRARY)SCA$EVENT.DAT") .EQS. "" THEN -
```

```
mms$scasetlib = (mms$scasetlib .AND. .NOT. 2) .OR. 4
IF ((mms$scasetlib .AND. 4) .EQ. 4) THEN SPAWN/WAIT/NOSYMBOLS $(SCA) -
CREATE LIBRARY $(SCAFLAGS) $(SCALIBRARY)
IF ((mms$scasetlib .AND. 4) .EQ. 4) .OR. ((mms$scasetlib .AND. 2) .EQ. 2) -
THEN DEFINE/USER SCA$LIBRARY $(SCALIBRARY)
$(SCA) LOAD $(SCAFLAGS) $(MMS$TARGET_NAME).ANA
```

C.9. Built-In Rules for CMS Access

This section lists the built-in rules for CMS access. A tilde (~) after a file type indicates that the file is in a CMS library.

```
.ADF~.ADF :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).ADF -
$(CMSFLAGS) $(CMSCOMMENT)
.ANL~.ANL :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).ANL -
$(CMSFLAGS) $(CMSCOMMENT)
.B16~.B16 :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).B16 -
$(CMSFLAGS) $(CMSCOMMENT)
.B32~.B32 :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).B32 -
$(CMSFLAGS) $(CMSCOMMENT)
.BAS~.BAS :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).BAS -
$(CMSFLAGS) $(CMSCOMMENT)
.BLI~.BLI :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).BLI -
$(CMSFLAGS) $(CMSCOMMENT)
.C~.C :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).C -
$(CMSFLAGS) $(CMSCOMMENT)
.CDO~.CDO :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).CDO -
$(CMSFLAGS) $(CMSCOMMENT)
.CLD~.CLD :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).CLD -
$(CMSFLAGS) $(CMSCOMMENT)
```

```
.COB~.COB :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).COB -
$(CMSFLAGS) $(CMSCOMMENT)
.COR~.COR :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).COR -
$(CMSFLAGS) $(CMSCOMMENT)
.COM~.COM :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).COM -
$(CMSFLAGS) $(CMSCOMMENT)
.CXX~.CXX :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).CXX -
$(CMSFLAGS) $(CMSCOMMENT)
.DAT~.DAT :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).DAT -
$(CMSFLAGS) $(CMSCOMMENT)
.DBL~.DBL :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).DBL -
$(CMSFLAGS) $(CMSCOMMENT)
.DDL~.DDL :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).DDL -
$(CMSFLAGS) $(CMSCOMMENT)
.FOR~.FOR :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).FOR -
$(CMSFLAGS) $(CMSCOMMENT)
.F90~.F90 :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$ (CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).F90 -
$(CMSFLAGS) $(CMSCOMMENT)
.F77~.F77 :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).F77 -
$(CMSFLAGS) $(CMSCOMMENT)
.F~.F :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).F -
$(CMSFLAGS) $(CMSCOMMENT)
.FRM~.FRM :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
```

```
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).FRM -
$(CMSFLAGS) $(CMSCOMMENT)
.GDF~.GDF :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).GDF -
$(CMSFLAGS) $(CMSCOMMENT)
.H~.H :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).H -
$(CMSFLAGS) $(CMSCOMMENT)
.HLP~.HLP :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).HLP -
$(CMSFLAGS) $(CMSCOMMENT)
.IFDL~.IFDL :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).IFDL -
$(CMSFLAGS) $(CMSCOMMENT)
.LDF~.LDF :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).LDF -
$(CMSFLAGS) $(CMSCOMMENT)
.LIB~.LIB :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).LIB -
$(CMSFLAGS) $(CMSCOMMENT)
.MAC~.MAC :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MAC -
$(CMSFLAGS) $(CMSCOMMENT)
.MAR~.MAR :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MAR -
$(CMSFLAGS) $(CMSCOMMENT)
.M64~.M64 :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).M64 -
$(CMSFLAGS) $(CMSCOMMENT)
.MDF~.MDF :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MDF -
$(CMSFLAGS) $(CMSCOMMENT)
.MMS~.MMS :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MMS -
$(CMSFLAGS) $(CMSCOMMENT)
.MSG~.MSG :
```

```
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MSG -
$(CMSFLAGS) $(CMSCOMMENT)
.OPT~.OPT :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).OPT -
$(CMSFLAGS) $(CMSCOMMENT)
.PAS~.PAS :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).PAS -
$(CMSFLAGS) $(CMSCOMMENT)
.PLI~.PLI :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).PLI -
$(CMSFLAGS) $(CMSCOMMENT)
.R16~.R16 :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).R16 -
$(CMSFLAGS) $(CMSCOMMENT)
.R32~.R32 :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).R32 -
$(CMSFLAGS) $(CMSCOMMENT)
.RBA~.RBA :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RBA -
$(CMSFLAGS) $(CMSCOMMENT)
.RC~.RC :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RC -
$(CMSFLAGS) $(CMSCOMMENT)
.RCO~.RCO :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RCO -
$(CMSFLAGS) $(CMSCOMMENT)
.REQ~.REQ :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).REQ -
$(CMSFLAGS) $(CMSCOMMENT)
.RFO~.RFO :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RFO -
$(CMSFLAGS) $(CMSCOMMENT)
.RGC~.RGC :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RGC -
```

```
$(CMSFLAGS) $(CMSCOMMENT)
.RNH~.RNH :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RNH -
$(CMSFLAGS) $(CMSCOMMENT)
.RNO~.RNO :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RNO -
$(CMSFLAGS) $(CMSCOMMENT)
.RPA~.RPA :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RPA -
$(CMSFLAGS) $(CMSCOMMENT)
.RPG~.RPG :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RPG -
$(CMSFLAGS) $(CMSCOMMENT)
.SC~.SC :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SC -
$(CMSFLAGS) $(CMSCOMMENT)
.SCN~.SCN :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SCN -
$(CMSFLAGS) $(CMSCOMMENT)
.SCO~.SCO :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SCO -
$(CMSFLAGS) $(CMSCOMMENT)
.SDML~.SDML :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SDML -
$(CMSFLAGS) $(CMSCOMMENT)
.SFO~.SFO :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SFO -
$(CMSFLAGS) $(CMSCOMMENT)
.SPA~.SPA :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SPA -
$(CMSFLAGS) $(CMSCOMMENT)
.SPL~.SPL :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SPL -
$(CMSFLAGS) $(CMSCOMMENT)
.SQLADA~.SQLADA :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
```

```
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SQLADA -
$(CMSFLAGS) $(CMSCOMMENT)
.SQLMOD~.SQLMOD :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).SQLMOD -
$(CMSFLAGS) $(CMSCOMMENT)
.TDF~.TDF :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).RBA -
$(CMSFLAGS) $(CMSCOMMENT)
.TXT~.TXT :
IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB -
$(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).TXT -
$(CMSFLAGS) $(CMSCOMMENT)
```