Guide to MadGoat Make Utility

December, 1998

This manual describes the MadGoat Make Utility (MMK), a ‘‘make’’ utility for VMS systems.

Revision/Update Information:	This is a revised manual.
Operating System and Version:	VAX/VMS V5.2 or later; OpenVMS Alpha V1.5 or later
Software Version:	MMK V3.8

Matthew Madison
MadGoat Software

28 December 1998

Contents

Preface

This guide explains how to install and use the MadGoat Make Utility (MMK).

Intended Audience

This manual is intended for all MMK users, primarily programmers who need to build software systems.

MMK is patterned after VAX DEC/Module Management System (DEC/MMS), which is in turn based on the UNIX *make* utility. The reader is assumed to have at least cursory knowledge of *make* or DEC/MMS.

Note: This document does not provide a general tutorial on *make* utilities. New users are advised to learn more about description files (makefiles) by reviewing either DEC/MMS documentation or books on the UNIX *make* utility.

Document Structure

tbs

Related Documents

tbs

Conventions

In this document, the following convention will be used for the names of the three similar utilities:

- **MMK** refers to MadGoat Make, the package described in this document.
- **DEC/MMS** refers to DEC/Module Management System, a product of Digital Equipment Corporation.
- ***make*** refers to the UNIX *make* utility.

1 Introduction

This chapter describes MadGoat Make (MMK). It includes an overview of MMK and basic information on its use.

1.1 Overview

MMK is a tool for building a "software system;" that is, a collection of one or more executable images or other types of files that are related to a particular project. Building a complex system by hand can be a difficult and time-consuming task; using command procedures can make the task easier, but it may still be time-consuming.

With MMK, you create a file called a *Makefile* or *MMS description file* to describe your software system: the *objects* (i.e., source files, object files, etc.) that comprise the system, the *dependencies* between those objects, and the commands used to build the system. When you invoke MMK, it performs the following steps:

1 MMK reads and parses the description file, constructing a tree from the objects and dependencies listed in the file.

2 It then identifies the object to be built (called the *target*).

3 The tree of dependencies is traced from the target, and the revision dates for the files in that path are compared. If an object doesn't exist or is older than the object it depends on, the commands to build the object are executed in a subprocess. This continues until all objects along the dependency path have been checked and the target has been brought completely up-to-date.

In this way, MMK can execute the commands to rebuild only those pieces of your software system that need rebuilding due to a change that you have made. This can drastically reduce development time for a project.

1.2 Invoking MMK

Provided that MMK has been installed using the steps laid out in the installation instructions (file AAAREADME.INSTALL in the kit), you can invoke MMK from DCL as a foreign command:

```
$MMK
```

Full command syntax is given in MMK. By default, MMK looks for a description file called DESCRIP.MMS in the current directory; if that file does not exist, it then looks for a file called MAKEFILE. If it cannot find that file, an error is signaled. You can use the /DESCRIPTION qualifier to specify a different name for your description file, if needed.

MMK starts by reading the description file and constructing a tree from the *objects* listed in the description file (e.g., source files, include files, object files, etc.) and a tree of *dependencies* between those objects. It then identifies the *target* object to be built, and traverses the dependency tree to identify those objects that need to be built (called *intermediate targets*) in order to build the target.

MMK compares each target's revision date/time against the objects on which it depends and executes the actions for that building the target only if needed. You can force a complete rebuild by using the /FROM_ SOURCES qualifier on the MMK command.

2 Description Files

The key to successfully building your software system with MMK is the creation of a complete and accurate description file. This chapter describes the format for a description file and its components.

2.1 Description File Components

A description file is a collection of the following components:

- **Dependencies**, which describe how one object depends on one or more other objects.
- **Actions**, which are commands to be executed when an object needs to be built.
- **Macro definitions**, for defining symbols that may be used in rules or actions.
- **Inference rule definitions**, which are rules based on suffixes (and possibly directories as well), from which MMK can infer dependencies and actions without you having to list them explicitly in your makefile.
- **MMK directives**, which provide a means for adding commands to be executed before or after all other actions, provide a simple conditional-build mechanism, and other directives for modifying MMK's behavior.

Here is an example of a simple description file:

```
PROGRAM.EXE : MAIN.OBJ,SUBROUTINES.OBJ
    LINK/EXEC=PROGRAM.EXE MAIN.OBJ,SUBROUTINES.OBJ
MAIN.OBJ : MAIN.FOR
    FORTRAN MAIN
SUBROUTINES.OBJ : SUBROUTINES.MAR
    MACRO SUBROUTINES
```

This is a simple collection of dependencies and actions for building an image called PROGRAM.EXE. PROGRAM.EXE depends on two object files, called MAIN.OBJ and SUBROUTINES.OBJ; MAIN is a FORTRAN module and SUBROUTINES is a MACRO module.

MMK accepts either a colon or the DEC/MMS DEPENDS_ON keyword to separate a target object from its sources. In either case, the separator must be surrounded by blanks—this differs from *make*, but is consistent with DEC/MMS syntax.

2.2 Using Inference Rules

MMK includes a collection of built-in inference rules and actions for most VMS programming languages. The rules are driven by the file type suffix attached to the object name; you must use the default file types in order to make use of the default rules.

For example, the description file in the last section could be simplified to just:

```
PROGRAM.EXE : MAIN.OBJ,SUBROUTINES.OBJ
    LINK/EXEC=PROGRAM.EXE MAIN.OBJ,SUBROUTINES.OBJ
MAIN.OBJ : MAIN.FOR
SUBROUTINES.OBJ : SUBROUTINES.MAR
```

MMK's built-in inference rules automatically define the actions for building a .OBJ file from a .FOR (using the FORTRAN command) and for building a .OBJ file from a .MAR file (using the MACRO command).

The description file could even be simplified further, to just:

```
PROGRAM.EXE : MAIN.OBJ,SUBROUTINES.OBJ
    LINK/EXEC=PROGRAM.EXE MAIN.OBJ,SUBROUTINES.OBJ
```

MMK automatically searches the *suffixes list* when constructing the dependency tree and locates inference rules for the .OBJ files automatically. This illustrates the second use for inference rules: they are used not only for inferring actions for a dependency that omits them, but they may also be used for inferring dependencies themselves based on a combination of source and target suffixes. This second purpose can greatly simplify your makefiles, and makes the build process more automatic.

2.3 Defining Inference Rules

You can define your own inference rules, either to extend or replace the ones built into MMK. You may include these rule definitions in your makefile, or in a separate file called a *rules file*. Rules files can be included by the use of a logical name or through the /RULES qualifier on the MMK command; see the MMK for further information.

MMK supports two types of inference rules: *generic* and *prefixed*. Generic rules are based solely on suffixes (file types), as in:

```
.C.OBJ :
    CC/OBJECT=$(MMS$TARGET) $(MMS$SOURCE)
```

which says, "to build *filename*.OBJ from an existing file called *filename*.C, use the **CC** command." In general, generic rules work best when the source and target files reside in the same directory.

Prefixed inference rules are based on both suffixes and "prefixes"—device and directory specifications. This provides a way to have MMK automatically infer dependenices between files that reside in different directories. For example: For example, the prefixed rule:

```
{SRC$:}.C{OBJ$:}.OBJ :
    CC/OBJECT=$(MMS$TARGET) $(MMS$SOURCE)
```

tells MMK, "to build OBJ$:*filename*.OBJ from an existing file called SRC$:*filename*.C, use the **CC** command." This works like the generic rule above, but with the additional provision of having the source and target reside in different locations.

You can have more than one prefixed rule for a particular pair of suffixes; you may also mix generic rules and prefixed rules for a pair of suffixes. When attempting to infer a dependency, MMK will first use the prefixed rules, then fall back to using the generic rule.

In prefixed rules, the curly braces ("{" and "}") are required. One of the two prefixes may be null, but specifying two null prefixes is equivalent to defining a generic rule.

In order to match a prefixed rule, file specification *as it exists in the description file* must match the prefix in the rule; MMK performs no logical name translation on prefixes, nor can it identify equivalencies between two prefixes that reference the same directory using different syntax.

The first inference rule for a pair of suffixes, whether it is generic or prefixed, must specify an action list; subsequent rules for the same pair of suffixes (with different prefixes) may have the action list omitted, in which case MMK will use the action list from the first rule. For example, MMK already has a built-in generic rule for .C.OBJ, which is:

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

If you are simply adding a set of prefixed rules for the .C.OBJ suffix pair, you do not need to specify an action list on those rules; MMK will use the action list from the built-in generic rule.

2.4 Forced Setup/Teardown Actions in Inference Rules

MMK recognizes two special modifiers on action lines specified for inference rules. The *setup* modifier, "<", forces the execution of an action prior to any unmodified action. The *teardown* modifier, ">", forces the execution of an action after all other actions. Setup and teardown actions are performed for all dependencies matching the inference rule, even if a dependency includes explicit actions.

For example, the inference rule

```
.C.OBJ :
    < DEFINE/USER DECC$SHR V6_ROOT:[SYSLIB]DECC$SHR
      $(CC)$(CFLAGS) $(MMS$SOURCE)
```

would cause the logical name DECC$SHR to be defined prior to the invocation of the C compiler for all compilations into .OBJ files. This would apply even on dependencies containing explicit actions, such as

```
FRED.OBJ : FRED.C
  $(CC)$(CFLAGS)/DEFINE=FRED $(MMS$SOURCE)
```

2.5 Modifying the Suffix List

MMK uses a *suffix list* to determine the inference rules it should search for inferring a dependency. MMK has a built-in suffix list which goes with its list of built-in inference rules; see Appendix B for more information on the built-in rules and suffix list.

You can augment or replace the built-in suffix list with your own suffixes by using the .SUFFIXES directive in a rules file or a makefile.

For example, let's say you have a Modula-2 compiler on your system, whose source files have a file type (suffix) of .MOD. MMK has no built-in inference rules for this file type; you could add one with the following sequence:

```
.SUFFIXES : .MOD

.MOD.OBJ :
    MODULA2/OBJECT=$(MMS$TARGET) $(MMS$SOURCE)
```

The .SUFFIXES directive above adds the .MOD suffix to the end of the suffix list. This is followed by the inference rule for creating an object file from a Modula-2 source file.

Specifying the .SUFFIXES directive with nothing to the right of the colon clears the current suffix list. You can do this to prevent MMK from using any inference rules for the current build, or to follow it with another .SUFFIXES directive that specifies only those suffixes for which you want inference rules to be enabled.

2.6 Using Conditionals

MMK provides several directives that can be used to modify the build sequence based on conditions. These directives are .IF, .IFDEF, .IFNDEF, .ELSE, and .ENDIF. The .IFDEF, .ELSE, and .ENDIF directives work the same as for MMS; the .IF and .IFNDEF directives are MMK extensions. The .IFNDEF directive provides the logical inverse to .IFDEF; the check succeeds if the specified macro is *not* defined. The .IF directive supports more general comparisons. Its syntax is

```
.IF "expression1" comparison "expression2"
```

where *expression1* and *expression2* can be any string that does not contain quotation marks. Macro references may be used in the expressions as long as the macro values do not contain quotation marks. The *comparison* operator is either **EQL** (equals) or **NEQ** (not equals). The quotation marks around the two expressions are required. Comparisons are performed without regard to upper/lower case.

2.7 Deferring Macro Substitution

MMK provides a way to defer the resolution of a macro that is referenced in the right-hand side of a macro definition, as an extension to MMS. Macros are normally referenced using the $(*name*) syntax, which causes the value of the macro to be substituted immediately when a line is parsed (except for MMK's "special" macros, such as MMS$SOURCE and MMS$TARGET).

You can defer this substitution in MMK by using the syntax ${*name*} instead. However, this syntax is only recognized on the right-hand side of a macro definition. This can be useful when defining macros in a rules file that rely on macros that do not get defined until another rules file or a description file gets processed. For example, you might have the following definition in a rules file:

```
CFLAGS = /OBJECT=$(MMS$TARGET)/NOLIST/DEFINE=(VMS_BUILD,${MOREDEFINES})
```

then in your description file, you can define the MOREDEFINES macro:

```
MOREDEFINES = ANOTHER_C_DEFINE
```

This will complete the CFLAGS macro value when it is referenced later in the description file.

2.8 Macro String Substitution

MMK provides two mechanisms for causing string substitution to occur when resolving a macro reference: suffix substitution and general string substitution.

2.8.1 Suffix Substitution

When a string contains a list of file specifications, you can replace the file type suffixes on each file specification with a different suffix. The general form of this type of substitution is:

$(*macro-name***:***old-sfx***=***new-sfx***)**

which causes the replacement of all occurences of the file type suffix *old-sfx* with *new-sfx*. Both suffixes must begin with a dot.

For example, in these macro definitions:

```
SOURCES = FIRST.C, SECOND.C, THIRD.C
OBJECTS = $(SOURCES:.C=.OBJ)
```

the OBJECTS macro would have the value “FIRST.OBJ, SECOND.OBJ, THIRD.OBJ”. This form of substitution works with file specification lists separated by either blanks, commas, or both. The substitution rule following the colon may also contain blanks, which are ignored. Substitutions are case-insensitive.

2.8.2 General String Substitution

General string substitution in macro references is an extended feature of MMK. It looks very similar to suffix substitution, but uses a double colon (“::”) instead of a single colon and allows the substitution to occur anywhere within the string. The syntax is:

$(*macro-name***::***old-str***=***new-str***)**

which causes the replacement of all occurences of the string *old-str* with *new-str*. You may use a backslash “\” as a “literal-next” escape when one of the strings contains an equals sign. Neither string may contain a right parenthesis character (“)”), even quoted with a backslash, although this restriction will be lifted in a future release of MMK. For example, the following macro definitions:

```
SOURCES = FIRST.C,SECOND.C,THIRD.C
SOURCEPLUS = $(SOURCES::,=+)
```

would cause SOURCEPLUS to contain the list of filenames separated with plus signs (“+”) rather than commas.

General string substitutions in macro references are case-insensitive, but *do not* ignore blanks in the macro value or in the substitution rule. For example, in the following definitions:

```
TEST      = Xyz xYz xyZ
REPLACED  = $(TEST::YZ =YZ,)
```

the REPLACED macro would have the value “XYZ,xYZ,xyZ”, due to the case-blind comparisons and the inclusion of the space in the *old-str* specification.

3 Using DEC/CMS with MMK

This chapter describes the use of Digital's DEC/Code Management System (DEC/CMS) with MMK.

3.1 The /CMS Qualifier

The MMK command supports a /CMS qualifier, which activates the automatic use of the currently set DEC/CMS library, or another DEC/CMS library that you specify, for the current build. This causes source files to be fetched out of the DEC/CMS library automatically, if needed. In addition, the MMK description file will automatically be fetched out of the DEC/CMS library if it does not exist.

The built-in suffix list and dependency rules in MMK include default rules for fetching source files out of DEC/CMS libraries. Suffixes ending in a tilde character ("~") signify DEC/CMS library elements. The built-in DEC/CMS element rules are used only if /CMS is specified on the MMK command.

3.2 Explicit CMS Element References

You can explicitly reference a CMS library element in your MMK description file by adding a tilde to the end of the file specification. For example:

```
MAIN.FOR  :  MAIN.FOR~
```

You can also explicitly name the CMS library from which the element should be fetched, by specifying a device and/or directory name:

```
MAIN.FOR  :  SOURCE_DISK:[CMS_SOURCE]MAIN.FOR~
```

If you do not explicitly name the CMS library, the currently set CMS library (set with CMS SET LIBRARY) will be used.

3.2.1 Specifying the Element Generation

By default, MMK uses the qualifier /GENERATION=1+ on all CMS FETCH operations, to get the highest-numbered generation of a particular element, or whichever generation you specify on the MMK /GENERATION qualifier. If you need to build a dependency on a specific generation of an element, you may do so by specifying the /GENERATION qualifier on the file name:

```
MAIN.FOR  :  MAIN.FOR~/GENERATION=37
```

The above example would cause generation 37 of the MAIN.FOR file in the current CMS library to be used for the build.

3.3 Inference Rules for CMS Files

MMK comes with built-in inference rules for fetching source files from a CMS library. Like DEC/MMS, MMK uses these rules *only* when you specify the /CMS qualifier on the MMK command. This allows you to have a makefile like the following:

```
TEST.EXE : TEST.OBJ
    $(LINK)$(LINKFLAGS) $(MMS$SOURCE)

TEST.OBJ : TEST.FOR
```

If you have a CMS library set and you specify the /CMS qualifier on the MMK command, MMK will automatically check to see if TEST.FOR resides in the CMS library and will fetch it out of the library if needed.

However, MMK also allows you to omit the second dependency in the makefile, and will automatically "double-infer" the existence the .FOR file, even if it has not yet been fetched out of the CMS library.

3.3.1 CMS and Prefixed Inference Rules

You can have MMK automatically search specific CMS libraries for source files by using prefixed inference rules. For example, if you were working on a cross-platform development project which used two CMS libraries - one for OS-specific source code and another for common source code - you might use the following prefixed rules:

```
{CMSSRC:[VMS_SPECIFIC]}.FOR~{}.FOR :
{CMSSRC:[COMMON]}.FOR~{}.FOR :
```

This sequence would cause MMK to automatically search the CMSSRC:[VMS_SPECIFIC] CMS library for a FORTRAN source file, then search the CMSSRC:[COMMON] library. If the file were not located in either library, MMK would fall back to using the currently set CMS library. You must still have a CMS library set and you must specify the /CMS qualifier for prefixed CMS inference rules to be tried.

MMK

Invokes the MMK utility to build a software system.

FORMAT

MMK *[target-name ...]*

Command Qualifiers	Defaults
/[NO]ACTION	*/ACTION*
/[NO]CHECK_STATUS	*/NOCHECK_STATUS*
/[NO]CMS	*/NOCMS*
/CMS_LIBRARY=(dir-spec...)	
/DESCRIPTION=file-spec	
/DUMP	
/[NO]FORCE	*/NOFORCE*
/[NO]FROM_SOURCES	*/NOFROM_SOURCES*
/GENERATION=string	*/GENERATION="1+"*
/IDENTIFICATION	
/[NO]IGNORE[=level]	*/NOIGNORE*
/[NO]LOCAL_RULES	*/LOCAL_RULES*
/[NO]LOG	*/NOLOG*
/MACRO=file-spec | definition...	
/OUTPUT=file-spec	
/[NO]OVERRIDE	*/NOOVERRIDE*
/[NO]RULES_FILE=file-spec...	
/[NO]SKIP_INTERMEDIATES	*/NOSKIP_INTERMEDIATES*
/[NO]VERIFY	*/VERIFY*
/WORKING_DIRECTORY=dir-spec	

PARAMETERS

target-name

Name of the target to be built. The target name must be listed in the description file. If no target name is specified, MMK builds the first target it finds in the description file. Multiple targets may be specified as a comma-separated list.

DESCRIPTION

The MMK utility builds a software system from the objects and dependencies listed in a description file. See the documentation for additional information.

QUALIFIERS

/[NO]ACTION

Determines whether action lines are executed or just displayed. Specifying /NOACTION causes MMK to display the action lines that would be executed to build the target, without actually executing them.

/[NO]CHECK_STATUS
Directs MMK to determine whether a target is up-to-date, without executing any action lines to do so. MMK sets the global symbol MMS$STATUS to 0 if the target requires updating, or 1 if the target is up-to-date. This qualifier overrides the /ACTION qualifier.

/[NO]CMS
Determines whether a DEC/Code Management System (CMS) library is automatically searched for the MMK description file and for any source files. The default is /NOCMS, which omits any CMS checks.

/CMS_LIBRARY=(dir-spec...)
Specifies one or more DEC/Code Management System (CMS) libraries to be searched for the MMK description file and for any source files that do not have explicit CMS library specifications. Effective only when /CMS is also specified. If omitted, the default CMS library or libraries (from the logical name CMS$LIB or the CMS SET LIBRARY command) are used.

/DESCRIPTION=file-spec
Specifies an alternative name for the MMK description file. The default description file name is DESCRIP.MMS (in the current default directory), with MAKEFILE. being used if DESCRIP.MMS does not exist.

/DUMP
Causes MMK to dump the suffix list, all currently defined macros, all inference rules, and all dependencies to the current output before starting the build. This qualifier is useful in debugging problems in rules files and makefiles.

/[NO]FORCE
Specifying /FORCE causes MMK to execute only the action lines from the dependency rule for the target, without performing any revision date checks and without building any intermediate targets.

/[NO]FROM_SOURCES
Specifying /FROM_SOURCES causes MMK to perform a complete build of the target, ignoring revision dates. All actions to build all intermediate targets are executed.

/GENERATION=string
Specifies the default generation to be used when MMK fetches elements out of a CMS library. If omitted, the default generation is “1+”, which fetches the highest-numbered generation of an element. You can use this qualifier in combination with CMS classes to have MMK build a specific version of your software system, provided that all source code for the system is fetched from CMS during the build.

/IDENTIFICATION
Specifying /IDENTIFICATION causes MMK to display its revision information and a copyright message, without performing any other action.

/[NO]IGNORE[=level]
By default, MMK stops when an executed action line results in a warning, error, or fatal error status. You can override this by specifying /IGNORE. Using /IGNORE or /IGNORE=FATAL causes all errors to be ignored;

specifying /IGNORE=ERROR causes errors and warnings to be ignored; specifying /IGNORE=WARNING causes only warnings to be ignored.

/[NO]LOCAL_RULES

Controls whether site-specific inference rule definitions are read in. By default, they are if the logical name MMK_LOCAL_RULES is defined and points to a readable description file. Specifying /NOLOCAL_RULES prevents this from occurring.

/[NO]LOG

Controls whether MMK logs a detailed description of its activity. By default, it does not.

/MACRO=file-spec | definintion...

Defines one or more macros that can be referenced by the description file. If a name is specified with no equals sign ("="), it is first assumed to be a file specification; if the file exists, macro definitions are read from the file. A file type of .MMS is assumed if no file type is specified. If the file cannot be found, the name is treated as macro definition, and a value of **1** is assigned to the macro by default.

If an equals sign is present, the macro definition is taken directly from the command line.

Macro definitions contained in a file should have the same syntax as macro definitions in description files, with the added restrictions that comments and line continuations are not allowed.

/OUTPUT=file-spec

Directs MMK output to a location other than the default, SYS$OUTPUT.

/[NO]OVERRIDE

Determines the order in which macro references are resolved. The default order is to resolve macros from command-line definitions, followed by definitions in the description file and any rules files, followed by MMK built-ins. If a macro cannot be resolved from any of these sources, a check is made for a DCL symbol with the same name as the macro, and if present, the symbol's value is used.

When /OVERRIDE is specified, DCL symbols are checked before the macro definitions in the description and rules files, and before the MMK built-in macros.

/[NO]RULES_FILE[=file-spec...]

Specifies the name of one or more description files containing inference rules. If /RULES_FILE is specified with no file specification, the name MMS$RULES is used by default (this can be a logical name or can reference a file called MMS$RULES.MMS in the current directory).

If /NORULES_FILE is specified, the compiled-in default rules are not loaded when MMK is started, nor is any personal rules file (pointed to by the logical name MMK_PERSONAL_RULES). /NORULES_FILE does not prevent the loading of local rules; you must also specify /NOLOCAL_RULES to prevent local rules from being loaded.

/[NO]SKIP_INTERMEDIATES

By default, MMK attempts to build missing source files if they can be built through the application of dependency or inference rules. Specifying /SKIP_INTERMEDIATES causes MMK to treat these missing sources as if they exist and have the same revision date/time stamp as the target that depends on them.

For example, if the target is a .EXE file which depends on a .OBJ file, and that .OBJ file in turn depends on a .C file, by default MMK will build the .OBJ file if it is missing, and then in turn build the .EXE. If /SKIP_INTERMEDIATES is specified, the missing .OBJ file will not trigger a build; the build will only occur if the .C file is newer than the .EXE file.

/[NO]VERIFY

Controls whether MMK echoes action lines to SYS$OUTPUT. Enabled by default.

/WORKING_DIRECTORY=dir-spec

Causes MMK to SET DEFAULT to the specified directory before processing the description file. This can be useful when using the /NOACTION qualifier for a build in a directory containing subdirectories with their own description files, since only actions invoking $(MMS) get executed when /NOACTION is used. By replacing the actions:

```
SET DEFAULT [.subdirectory]
$(MMS) $(MMSQUALIFIERS)
```

with the action

```
$(MMS) $(MMSQUALIFIERS)/WORKING_DIRECTORY=[.subdirectory]
```

the entire multi-directory build can be tested with /NOACTION successfully.

A Differences between MMK and DEC/MMS

MMK is patterned after DEC/MMS, but does not fully implement all DEC/MMS functionality and provides other extended functionality. This appendix lists some of the differences between MMK and DEC/MMS.

Besides the differences in features, there are some differences in processing between MMK and DEC/MMS which may lead to different results or syntax errors in MMK for description files which operate properly under DEC/MMS. If possible, please report any such differences to the author so that they can be fixed.

A.1 DEC/MMS Features Not Supported in MMK

MMK does not support the following DEC/MMS features:

- MMK does not support FMS forms libraries or CDD/Repository libraries.
- MMK does not honor the "; *action*" syntax on dependency rule lines that can be used with DEC/MMS. Make sure all actions are on separate lines.
- MMK requires the leading dot on the .INCLUDE directive.
- MMK does not handle wildcard dependency rules.
- MMK does not support all of the command qualifiers supported by DEC/MMS. In addition, the MMK's /GENERATION qualifier is completely different from DEC/MMS's /GENERATE qualifier.
- MMK does not have a DECwindows interface.
- MMK does not automatically load the MMS$RULES file if that logical name is defined or that file exists in the current directory; you must specify /RULES on the MMK command to have it loaded. Use the MMK_LOCAL_RULES or MMK_PERSONAL_RULES logical names to have rules automatically loaded by MMK.

A.2 MMK Extended Features

MMK includes the following features not found in DEC/MMS:

- MMK gives you more options for rules files, and is set up to allow multiple rules files to be present. Rule file processing follows this sequence:

  1 The default rules compiled into MMK are loaded automatically unless /NORULES_FILE is specified on the MMK command.

  2 A site-defined local rules file is loaded automatically if the logical name MMK_LOCAL_RULES is defined (use /NOLOCAL_RULES to override).

3 If the /RULES_FILE qualifier is specified, any rules files listed there are loaded; if none are listed, the default is to load the file MMS$RULES.MMS (or the file pointed to by the logical name MMS$RULES).

If the /RULES_FILE qualifier is omitted, a personal rules file is loaded if the logical name MMK_PERSONAL_RULES is defined. MMS$RULES is *not* loaded in this case.

If /NORULES_FILE is specified, neither MMS$RULES nor the personal rules file is loaded.

These rules-processing features, coupled with the ability to redefine macros defined in rules files, make it easier to customize MMK's behavior when needed.

- MMK trims blanks out of $(MMS$SOURCE_LIST).
- MMK includes support for the following special local macros:
  - $(MMS$SOURCE_LIST_SPACES) - source list with spaces as separators instead of commas.
  - $(MMS$CHANGED_LIST_SPACES) - list of changed sources with spaces as separators instead of commas.
  - $(MMS$SOURCE_NAME) - like $(MMS$TARGET_NAME), but for $(MMS$SOURCE).
  - $(MMS$TARGET_FNAME) - like $(MMS$TARGET_NAME), but does not include the device/directory specification, just the filename.
  - $(MMS$TARGET_MODULE) - name of the module being replaced in a text, help, macro, or object library.
- MMK will display activity in the subprocess while action lines are being executed when you press CTRL/T.
- MMK allows you to redefine macros.
- MMK, in most cases, has more flexible syntax rules for its description files, allowing blanks where MMS does not (e.g., in library module specifications).
- MMK pre-defines the macros __VAX__ for builds on VAX systems and __AXP__ for builds on Alpha systems.
- MMK supports prefixed inference rules (described in Section 2.3).
- When used with DEC/CMS, MMK will "double-infer" a dependency on a non-existent source file, if that file currently resides in a CMS library.
- MMK includes a /DUMP qualifier for debugging problems with makefiles.
- MMK provides a /GENERATION qualifier on the MMK command for specifying the default CMS generation to be used when elements are checked for revisions and fetched out of CMS.
- MMK provides the /WORKING_DIRECTORY qualifier.

- MMK allows "generic" targets (those that do not refer to an actual file) to have null action lists. MMS requires all targets to have an action list.
- MMK adds the ${name} syntax for deferring resolution of macros on the right-hand side of a macro definition.
- MMK adds a generic string-susbstitution function for macro references.
- MMK adds the .IFNDEF and .IF directives to make it easier to conditionalize builds.

You should avoid using these extended features if you need to maintain compatibility with DEC/MMS.

A.3 Other Differences

Besides the feature differences alrady mentioned, MMK operates somewhat differently from DEC/MMS in some of its processing. In most cases, these differences are not significant, but they are worth remembering if you need to port DEC/MMS description files to or from MMK.

- MMK allows any rule, including built-in rules, to override the .DEFAULT actions. DEC/MMS lets .DEFAULT actions override built-in rules.
- When a build action does not update a target, MMK will issue an information message, except for generic targets. DEC/MMS only issues such messages in certain cases.
- MMK explicitly builds dependency rules for files on which library modules depend, even if those files are not mentioned in the description file. This may lead to MMK behaving differently from DEC/MMS, although if the description file is correct, the end result will be the same.
- MMK parses comments and continuation lines differently, so that a hyphen at the end of a comment is not considered a continuation of the comment.

As other differences are brought to the author’s attention, they will either be fixed or noted here.

B Built-in Dependency Rules

The dependency rules built into MMK for VAX systems is given in Figure B–1. The dependency rules built into MMK for Alpha systems is given in Figure B–2.

Figure B–1 MMK default dependency rules - VAX

```
! MMK_DEFAULT_RULES.MMS
!
!   COPYRIGHT © 1993, 1994, 1997  MADGOAT SOFTWARE.  ALL RIGHTS RESERVED.
!
!  Default build rules for use with MMK.  (for VAX systems)
!
!  Modification history:
!
!   23-DEC-1992 V1.0 Madison     Initial coding.
!   17-OCT-1993 V1.1 Madison     Elimination of intermediate libfiles.
!   11-APR-1994 V1.2 Madison     Make rules more like MMS's.
!   01-JUL-1994 V2.0 Madison     Add CMS support.
!   16-JUL-1994 V2.1 Madison     Update for V3.2.
!   22-AUG-1994 V2.1-1 Madison     Eliminate DELETE_SOURCE checks.
!   14-OCT-1994 V2.2 Madison     Add CXX support.
!   28-DEC-1994 V2.3 Madison     Make IF commands silent.
!   20-JUN-1997 V2.3-1 Madison     Add .MAR.MLB inference rule.
!

!
! These symbols can be used to distinguish MMK from DEC's DEC/MMS product
! using .IFDEF directives.
!
__MATTS_MMS__ = __MATTS_MMS__
__MMK__ = __MMK__
__MMK_V32__ = 1

!
! This symbol can be used to distinguish a VAX-based build from an
! AXP-based build. (or use .IFDEF __AXP__ .ELSE ... .ENDIF)
!
__VAX__ = 1

EXE = .EXE
OLB = .OLB
OBJ = .OBJ
OPT = .OPT
L32 = .L32

.SUFFIXES :     ! clear the suffix list first
.SUFFIXES : $(EXE) $(OLB) $(OBJ) .TLB .HLB .MLB $(L32) .C .CXX .BAS .B32 .BLI .FOR
         .COB .COR .DBL .RPG .SCN .PLI .PEN .PAS .MAC .MAR .MSG .CLD .R32 -
         .REQ .TXT .H .MEM .HLP .RNH .RNO .MMS .DAT .OPT .SDML .COM -
         .C~ .CXX~ .BAS~ .B32~ .BLI~ .FOR~ .COB~ .COR~ .DBL~ .RPG~ .SCN~ -
         .PLI~ .PAS~ .MAC~ .MAR~ .MSG~ .CLD~ .R32~ .REQ~ .TXT~ -
         .H~ .HLP~ .RNH~ .RNO~ .MMS~ .DAT~ .OPT~ .SDML~ .COM~

LINK     = LINK
LINKFLAGS   = /EXEC=$(MMS$TARGET)
```

Figure B–1 Cont'd on next page

Figure B–1 (Cont.) MMK default dependency rules - VAX

```
$(OBJ)$(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)/MODULE=$(MMS$TARGET_MODULE)

.HLP.HLB :
    @ IF F$SEARCH("$(MMS$TARGET)") .EQS. "" THEN $(LIBR)/CREATE/HELP $(MMS$TARGET)
    $(LIBR)$(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)

.MAC.MLB :
    @ IF F$SEARCH("$(MMS$TARGET)") .EQS. "" THEN $(LIBR)/CREATE/MACRO $(MMS$TARGET)
    $(LIBR)$(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)

.MAR.MLB :
    @ IF F$SEARCH("$(MMS$TARGET)") .EQS. "" THEN $(LIBR)/CREATE/MACRO $(MMS$TARGET)
    $(LIBR)$(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)

LIBR     = LIBRARY
LIBRFLAGS   = /REPLACE

.BAS$(OBJ) :
    $(BASIC)$(BASFLAGS) $(MMS$SOURCE)
BASIC     = BASIC
BASFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.BLI$(OBJ) :
    $(BLISS)$(BFLAGS) $(MMS$SOURCE)
.B32$(OBJ) :
    $(BLISS)$(BFLAGS) $(MMS$SOURCE)
BFLAGS     = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.C$(OBJ) :
    $(CC)$(CFLAGS) $(MMS$SOURCE)
CC       = CC
CFLAGS     = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.COB$(OBJ) :
    $(COBOL)$(COBFLAGS) $(MMS$SOURCE)
COBOL     = COBOL
COBFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.COR$(OBJ) :
    $(CORAL)$(CORFLAGS) $(MMS$SOURCE)
CORAL     = CORAL
CORFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.CXX$(OBJ) :
    $(CXX)$(CXXFLAGS) $(MMS$SOURCE)
CXX      = CXX
CXXFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.DBL$(OBJ) :
    $(DIBOL)$(DBLFLAGS) $(MMS$SOURCE)
DIBOL     = DIBOL
DBLFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.CLD$(OBJ) :
    $(SETCMD)$(SETCMDFLAGS) $(MMS$SOURCE)
SETCMD     = SET COMMAND
SETCMDFLAGS = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)
```

Figure B–1 Cont'd on next page

Figure B–1 (Cont.) MMK default dependency rules - VAX

```
.FOR$(OBJ) :
    $(FORT)$(FFLAGS) $(MMS$SOURCE)
FORT        = FORTRAN
FFLAGS      = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.MAR$(OBJ) :
    $(MACRO)$(MFLAGS) $(MMS$SOURCE)
MACRO     = MACRO
MFLAGS     = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.MSG$(OBJ) :
    $(MESSAGE)$(MSGFLAGS) $(MMS$SOURCE)
MESSAGE     = MESSAGE
MSGFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.PAS$(OBJ) :
    $(PASCAL)$(PFLAGS) $(MMS$SOURCE)
.PAS.PEN   :
    $(PASCAL)$(PENVFLAGS) $(MMS$SOURCE)
PASCAL     = PASCAL
PFLAGS     = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)
PENVFLAGS   = /ENVIRONMENT=$(MMS$TARGET_NAME).PEN/NOLIST

.PLI$(OBJ) :
    $(PLI)$(PLIFLAGS) $(MMS$SOURCE)
PLI      = PLI
PLIFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.REQ$(L32) :
    $(BLISS)/LIBR=$(MMS$TARGET_NAME)$(L32)$(BLIBFLAGS) $(MMS$SOURCE)
.R32$(L32) :
    $(BLISS)/LIBR=$(MMS$TARGET_NAME)$(L32)$(BLIBFLAGS) $(MMS$SOURCE)
BLISS     = BLISS
BLIBFLAGS   = /NOLIST

.RPG$(OBJ) :
    $(RPG)$(RPGFLAGS) $(MMS$SOURCE)
RPG      = RPG
RPGFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.RNH.HLP :
    $(RUNOFF)$(RFLAGS) $(MMS$SOURCE)
.RNO.MEM :
    $(RUNOFF)$(RFLAGS) $(MMS$SOURCE)
RUNOFF     = RUNOFF
RFLAGS     = /OUTPUT=$(MMS$TARGET)

.SCN$(OBJ) :
    $(SCAN)$(SCANFLAGS) $(MMS$SOURCE)
SCAN     = SCAN
SCANFLAGS   = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

CMS      = CMS
CMSCOMMENT  = ""
CMSFLAGS    = /GENERATION=$(MMS$CMS_GEN)

.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) $(CMSCO
```

Figure B–1 Cont'd on next page

Figure B–1 (Cont.) MMK default dependency rules - VAX

```
.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCOMM

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCOMM

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO
```

Figure B–1 Cont'd on next page

Figure B–1 (Cont.) MMK default dependency rules - VAX

```
.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSC

.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) $(CMSCO
```

Figure B–2 MMK default dependency rules - Alpha

```
! MMK_DEFAULT_RULES_AXP.MMS
!
!   COPYRIGHT © 1993, 1997  MADGOAT SOFTWARE.  ALL RIGHTS RESERVED.
!
!  Default build rules for use with MMK.  (for OpenVMS AXP)
!
!  Modification history:
!
!   23-DEC-1992 V1.0 Madison     Initial coding.
!   17-OCT-1993 V1.1 Madison     Delete intermediate libfiles.
!   11-APR-1994 V1.2 Madison     Make rules more like MMS's.
!   05-JUL-1994 V2.0 Madison     Add CMS support.
!   16-JUL-1994 V2.1 Madison     Update for V3.2.
!   22-AUG-1994 V2.1-1 Madison     Eliminate DELETE_SOURCE checks.
!   14-OCT-1994 V2.2 Madison     Add CXX support.
!   28-DEC-1994 V2.3 Madison     Make IF commands silent.
!   20-JUN-1997 V2.3-1 Madison     Add .MAR.MLB inference rule.
!
```

Figure B–2 Cont'd on next page

Figure B–2 (Cont.) MMK default dependency rules - Alpha

```
!
! This symbol can be used to distinguish MMK from DEC's DEC/MMS product
! using .IFDEF directives.
!
__MATTS_MMS__ = __MATTS_MMS__
__MMK__ = __MMK__
__MMK_V32__ = 1
!
! These symbols can be used to distinguish an AXP-based build from a
! VAX-based build.
!
__ALPHA__ = 1
__AXP__   = 1

EXE = .EXE
OLB = .OLB
OBJ = .OBJ
OPT = .OPT
L32 = .L32

.SUFFIXES :     ! clear the suffix list first
.SUFFIXES : $(EXE) $(OLB) $(OBJ) .TLB .HLB .MLB $(L32) .C .CXX .BAS .B32 .BLI .FOR
         .COB .COR .DBL .RPG .SCN .PLI .PEN .PAS .MAC .MAR .M64 .MSG .CLD -
         .R32 .REQ .TXT .H .MEM .HLP .RNH .RNO .MMS .DAT .OPT .SDML .COM -
         .C~ .CXX~ .BAS~ .B32~ .BLI~ .FOR~ .COB~ .COR~ .DBL~ .RPG~ .SCN~ -
         .PLI~ .PAS~ .MAC~ .MAR~ .M64~ .MSG~ .CLD~ .R32~ .REQ~ .TXT~ -
         .H~ .HLP~ .RNH~ .RNO~ .MMS~ .DAT~ .OPT~ .SDML~ .COM~

LINK     = LINK
LINKFLAGS   = /EXEC=$(MMS$TARGET)

$(OBJ)$(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)/MODULE=$(MMS$TARGET_MODULE)

.HLP.HLB :
    @ IF F$SEARCH("$(MMS$TARGET)") .EQS. "" THEN $(LIBR)/CREATE/HELP $(MMS$TARGET)
    $(LIBR)$(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)

.MAC.MLB :
    @ IF F$SEARCH("$(MMS$TARGET)") .EQS. "" THEN $(LIBR)/CREATE/MACRO $(MMS$TARGET)
    $(LIBR)$(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)

.MAR.MLB :
    @ IF F$SEARCH("$(MMS$TARGET)") .EQS. "" THEN $(LIBR)/CREATE/MACRO $(MMS$TARGET)
    $(LIBR)$(LIBRFLAGS) $(MMS$TARGET) $(MMS$SOURCE)

LIBR     = LIBRARY
LIBRFLAGS   = /REPLACE


.BAS$(OBJ) :
    $(BASIC)$(BASFLAGS) $(MMS$SOURCE)
BASIC     = BASIC
BASFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)


.BLI$(OBJ) :
    $(BLISS)$(BFLAGS) $(MMS$SOURCE)
.B32$(OBJ) :
    $(BLISS)$(BFLAGS) $(MMS$SOURCE)
BFLAGS     = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)
```

Figure B–2 Cont'd on next page

Figure B–2 (Cont.) MMK default dependency rules - Alpha

```
.C$(OBJ) :
    $(CC)$(CFLAGS) $(MMS$SOURCE)
CC        = CC
CFLAGS     = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.COB$(OBJ) :
    $(COBOL)$(COBFLAGS) $(MMS$SOURCE)
COBOL     = COBOL
COBFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.COR$(OBJ) :
    $(CORAL)$(CORFLAGS) $(MMS$SOURCE)
CORAL     = CORAL
CORFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.CXX$(OBJ) :
    $(CXX)$(CXXFLAGS) $(MMS$SOURCE)
CXX      = CXX
CXXFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.DBL$(OBJ) :
    $(DIBOL)$(DBLFLAGS) $(MMS$SOURCE)
DIBOL     = DIBOL
DBLFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.CLD$(OBJ) :
    $(SETCMD)$(SETCMDFLAGS) $(MMS$SOURCE)
SETCMD     = SET COMMAND
SETCMDFLAGS = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.FOR$(OBJ) :
    $(FORT)$(FFLAGS) $(MMS$SOURCE)
FORT        = FORTRAN
FFLAGS      = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.MAR$(OBJ) :
    $(MACRO)$(MFLAGS) $(MMS$SOURCE)
MACRO     = MACRO/MIGRATION
MFLAGS     = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.M64$(OBJ) :
    $(TASM)$(TASMFLAGS) $(MMS$SOURCE)
TASM     = MACRO
TASMFLAGS   = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.MSG$(OBJ) :
    $(MESSAGE)$(MSGFLAGS) $(MMS$SOURCE)
MESSAGE     = MESSAGE
MSGFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.PAS$(OBJ) :
    $(PASCAL)$(PFLAGS) $(MMS$SOURCE)
.PAS.PEN   :
    $(PASCAL)$(PENVFLAGS) $(MMS$SOURCE)
PASCAL     = PASCAL
PFLAGS     = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)
PENVFLAGS   = /ENVIRONMENT=$(MMS$TARGET_NAME).ENV/NOLIST
```

Figure B–2 Cont'd on next page

Figure B–2 (Cont.) MMK default dependency rules - Alpha

```
.PLI$(OBJ) :
    $(PLI)$(PLIFLAGS) $(MMS$SOURCE)
PLI      = PLI
PLIFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.REQ$(L32) :
    $(BLISS)/LIBR=$(MMS$TARGET_NAME)$(L32)$(BLIBFLAGS) $(MMS$SOURCE)
.R32$(L32) :
    $(BLISS)/LIBR=$(MMS$TARGET_NAME)$(L32)$(BLIBFLAGS) $(MMS$SOURCE)
BLISS     = BLISS
BLIBFLAGS   = /NOLIST

.RPG$(OBJ) :
    $(RPG)$(RPGFLAGS) $(MMS$SOURCE)
RPG      = RPG
RPGFLAGS    = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

.RNH.HLP :
    $(RUNOFF)$(RFLAGS) $(MMS$SOURCE)
.RNO.MEM :
    $(RUNOFF)$(RFLAGS) $(MMS$SOURCE)
RUNOFF     = RUNOFF
RFLAGS     = /OUTPUT=$(MMS$TARGET)

.SCN$(OBJ) :
    $(SCAN)$(SCANFLAGS) $(MMS$SOURCE)
SCAN     = SCAN
SCANFLAGS   = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$(OBJ)

CMS      = CMS
CMSCOMMENT  = ""
CMSFLAGS    = /GENERATION=$(MMS$CMS_GEN)

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCOMM

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO
```

Figure B–2 Cont'd on next page

Figure B–2 (Cont.) MMK default dependency rules - Alpha

```
.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCOMM

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSCO
```

Figure B–2 Cont'd on next page

Figure B–2 (Cont.) MMK default dependency rules - Alpha

```
.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) $(CMSCO

.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) $(CMSCO

.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) $(CMSC

.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) $(CMSCO
```

C Using the CROSS_ALPHA Rules

The CROSS_ALPHA.MMS rules file is included with MMK as an aid to developers working on programs that will be run on both VAX and Alpha systems. The default dependency rules built into MMK are based on the macros show in Table C–1. They are used by default on both VAX and Alpha systems.

Table C–1 MMK default suffix macros

Macro	Suffix	Meaning
$(EXE)	.EXE	Executable image
$(L32)	.L32	BLISS Library file
$(OBJ)	.OBJ	Object file
$(OLB)	.OLB	Object library
$(OPT)	.OPT	Linker options file

Since the files mentioned in the table are of a different format on OpenVMS Alpha systems, developers wishing to do both Alpha and VAX builds in the same directory need a way of preventing the differing files from interfering with each other. The CROSS_ALPHA.MMS rules redefine the macros as shown in Table C–2, eliminating the name conflict.

Table C–2 CROSS_ALPHA suffix macros

Macro	Suffix	Meaning
$(EXE)	.ALPHA_EXE	Executable image
$(L32)	.L32E	BLISS Library file
$(OBJ)	.ALPHA_OBJ	Object file
$(OLB)	.ALPHA_OLB	Object library
$(OPT)	.ALPHA_OPT	Linker options file

To make all this work, you must use the macros in your description file instead of making literal references to the file type suffixes. For example:

```
PROGRAM$(EXE) : PROGRAM$(OBJ),SUBROUTINES$(OBJ),PROGRAM$(OPT)
    $(LINK)$(LINKFLAGS) PROGRAM$(OPT)/OPTION
PROGRAM$(OBJ) : PROGRAM.C
SUBROUTINES$(OBJ) : SUBROUTINES.FOR
```

It also helps to use the macros for the commands to compile and link programs, especially if you are using DEC's Alpha Migration Tools and cross-compiling your AXP objects on a VAX.

In addition to using the CROSS_ALPHA rules, MMK provides the special macros __AXP__ and __ALPHA__, which are predefined only when MMK is running on an OpenVMS AXP system (it also provides the __VAX__ predefined macro for VAX-based builds). This allows you to conditionalize your description file with the .IFDEF directive to handle Alpha- or VAX-specific cases.