

Guide to the MMK Make Utility

March, 2013

This manual describes the MMK Make Utility, 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; OpenVMS IA64 V8.2 or later

Software Version: MMK V5.0

**Endless Software Solutions
Perth, Western Australia**

07 March 2013

Copyright ©2008 Matthew Madison.

Copyright ©2013 Endless Software Solutions.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright owner nor the names of any other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions of the software were adapted from other open source projects. Refer to the Release Notes for copyright and license information.

The following are trademarks of Hewlett-Packard Development Company, LP:

AXP	DEC	OpenVMS
VAX	VMS	

UNIX is a registered trademark of USL, Inc.

Contents

PREFACE	vii
---------	-----

CHAPTER 1 INTRODUCTION	1-1
-------------------------------	------------

1.1 OVERVIEW	1-1
1.2 INVOKING MMK	1-1

CHAPTER 2 DESCRIPTION FILES	2-1
------------------------------------	------------

2.1 DESCRIPTION FILE COMPONENTS	2-1
2.2 USING INFERENCE RULES	2-1
2.3 DEFINING INFERENCE RULES	2-2
2.4 FORCED SETUP/TEARDOWN ACTIONS IN INFERENCE RULES	2-3
2.5 MODIFYING THE SUFFIX LIST	2-3
2.6 USING CONDITIONALS	2-4
2.6.1 GNU Make Conditionals	2-5
2.7 MACRO DEFINITIONS	2-6
2.7.1 Command Assignment	2-6
2.7.2 Evaluated Assignment	2-7
2.8 DEFERRING MACRO SUBSTITUTION	2-7
2.9 MACRO STRING SUBSTITUTION	2-8

Contents

2.9.1	Suffix Substitution _____	2-8
2.9.2	General String Substitution _____	2-8
<hr/>		
2.10	BULTIN FUNCTIONS	2-9
	ADDPREFIX	2-10
	ADDSUFFIX	2-11
	AND	2-12
	BASENAME	2-13
	CALL	2-14
	COLLAPSE	2-15
	DIR	2-16
	DIRECTORY	2-17
	FILENAME	2-18
	FILETYPE	2-19
	FILEVERSION	2-20
	FILTER	2-21
	FILTER-OUT	2-22
	FINDSTRING	2-23
	FIRSTWORD	2-24
	FOREACH	2-25
	IF	2-26
	JOIN	2-27
	LASTWORD	2-28
	NOTDIR	2-29
	OR	2-30
	ORIGIN	2-31
	PATSUBST	2-32
	SORT	2-33
	STRIP	2-34
	SUBST	2-35
	WILDCARD	2-36
	WORD	2-37
	WORDLIST	2-38
	WORDS	2-39
<hr/>		
CHAPTER 3	USING DEC/CMS WITH MMK	3-1
<hr/>		
3.1	THE /CMS QUALIFIER	3-1
<hr/>		
3.2	EXPLICIT CMS ELEMENT REFERENCES	3-1
3.2.1	Specifying the Element Generation _____	3-1
<hr/>		
3.3	INFERENCE RULES FOR CMS FILES	3-2
3.3.1	CMS and Prefixed Inference Rules _____	3-2
	MMK	3-3

APPENDIX A DIFFERENCES BETWEEN MMK AND DEC/MMS A-1

A.1 DEC/MMS FEATURES NOT SUPPORTED IN MMK A-1

A.2 MMK EXTENDED FEATURES A-1

A.3 OTHER DIFFERENCES A-3

APPENDIX B BUILT-IN DEPENDENCY RULES B-1

FIGURES

B-1 MMK default dependency rules - VAX _____ B-1

B-2 MMK default dependency rules - Alpha _____ B-5

B-3 MMK default dependency rules - IA64 _____ B-10

TABLES

2-1 Conditional Expression Operators _____ 2-5

2-2 Macro Assignment Operator _____ 2-6

2-3 Macro Definition Origins _____ 2-31

2-4 PATSUBST Pattern Operators _____ 2-32

Preface

This guide explains how to install and use 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.

Further information regarding MMK, including release details and support agreements can be found at the Endless Software Solutions website:

<http://www.endlesssoftware.com.au>

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

The following documents might also be helpful when using MMK:

- The *MMK Release Notes* contain further information related to new and deprecated features of MMK.
- The *Guide to the Module Management System* (order number: AA-P119J-TE) contains further information related to DEC/MMS and can be found at <http://h71000.www7.hp.com/doc/decset.html>.
- The *GNU Make Manual* contains information related to *make* and can be found at <http://www.gnu.org/software/make/manual/>.

Conventions

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

- **MMK** refers to 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 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.

Introduction

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.

Description Files

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 dependencies 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.

Description Files

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`, `.ELSIF` and `.ENDIF`. The `.IFDEF` directive checks if the specified macro is defined and `.IFNDEF` is the logical inverse, the syntax of which is:

```
.IFDEF VEGETABLE
...
.ENDIF
```

or,

```
.IFNDEF VEGETABLE
...
.ENDIF
```

The `.IF` directive supports more general comparison (and thus greater power) by allowing the specification of a boolean expression, like so:

```
.IF expression
...
[.ELSEIF expression]
[...]
[.ELSE]
[...]
.ENDIF
```

In the example above, *expression* can be replaced by any operations consisting of the operators (both MMS and MMK operators are listed) listed in Table 2-1.

Table 2–1 Conditional Expression Operators

MMS Syntax	MMK Syntax	Example
Boolean Operators		
.AND	AND	expression .AND expression
.NOT	NOT	.NOT expression
.OR	OR	expression .OR expression
Comparison Operators		
.EQ	EQL	expression .EQ expression
.GE	GEQ	expression .GE expression
.GT	GTR	expression .GT expression
.LE	LEQ	expression .LE expression
.LT	LSS	expression .LT expression
.NE	NEQ	expression .NE expression

All expressions evaluate to a boolean result. After all macro substitution is performed and the the operand is not an empty string it is the equivalent of a boolean TRUE. All comparisons are performed without regard to upper/lower case.

2.6.1 GNU Make Conditionals

Supplemental to the MMS/MMK conditionals, MMK also supports the specification of the GNU Make-style conditionals *ifeq*, *ifneq*, *ifdef*, *ifndef*, *else* and *endif*. The *ifdef* and *ifndef* conditionals behave exactly as their MMS/MMK counter-parts. However, *ifeq* and *ifneq* have a slightly different syntax:

```
ifeq (arg1, arg2)
ifeq 'arg1' 'arg2'
ifeq "arg1" "arg2"
ifeq "arg1" 'arg2'
ifeq 'arg1' "arg2"
```

Although the example lists *ifeq*, it can be replaced with its logical inverse, *ifneq*. These two directives simply compare their arguments (ignoring case) and test for equality. The following is an example adapted from the GNU Make manual:

```
ifeq ($(strip $(foo)),)
# Come in here if foo is an empty string
...
endif
```

In order to use the conditionals it is necessary to first enable them. This can be done from the command line with the specification of the `/EXTENDED_SYNTAX=GNU_SYNTAX` qualifier or with the `.GNU_SYNTAX` directive in the description file.

2.7 Macro Definitions

MMK supports a range of operators for assigning values to macros. To simplify description files it is possible to concatenate and test before assignment. Table 2–2 defines the available assignment operators.

Table 2–2 Macro Assignment Operator

Operator	Description	Example
=	Assign value.	<i>FOO = BAR</i>
+=	Concatenate to existing value.	<i>FOR += ,NEXT</i>
?=	Assign if macro is undefined.	<i>FOO ?= FIRST</i>
!=	Assign command results.	<i>FOO != DIRECTORY</i>
~=	Completely evaluate value before assignment	

Two of these assignment operators behave a little differently than the others. They are both described in more detail below.

2.7.1 Command Assignment

Command assignment is the `!=` operator. The assigned value is taken as a command and executed, with the results being assigned to the macro. Using this mechanism it would be possible to determine some details of the running system, like so:

```
FOO != SHOW SYSTEM/NOPROCESS
ALL :
    @ WRITE SYS$OUTPUT "$(FOO)"
```

the results of which might look something like:

```
$ MMK
OpenVMS V7.3 on node BENDER 15-FEB-2013 16:05:10.03 Uptime 37 15:54:40
```

Note:

The `!'` character is already a comment delimiter in MMK. To resolve this, the MMK description file parser now tests for a `'=` character immediately following a `!'`. Usually when detecting a `!'` character the remainder of the line (including the comment delimiter) will be stripped from the output. However, in this special case the delimiter and the following characters will not be stripped and instead parsed as an assignment. This may create issues for existing procedures that contain comments that match this pattern.

However, it is worth noting that this restriction only applies to the `!'` comment delimiter. It is still possible to use `'#` in any place and achieve the desired result.

2.7.2 Evaluated Assignment

Evaluated assignment is the `~=` operator. This assignment operator is special in that it completely evaluated the value before assigning it to the macro.

All records in the description file are evaluated and an attempt is made to resolve all macros as they are read in to the parser. However, if an unresolved macro, or a builtin function call is found it is not evaluated to the empty string, rather it is retained for a time when those macros are available. An example from the default definitions within MMK is:

```
BASIC          = BASIC
BASFLAGS      = /NOLIST/OBJECT=$(MMS$TARGET_NAME)$ (OBJ)
```

The definition of `BASFLAGS` will not be evaluated fully as the description file is processed, leaving `MMS$TARGET_NAME` and `OBJ` unresolved until `BASFLAGS` is resolved as a part of executing the build actions that include this definition (shown below).

```
.BAS$(OBJ) :
    $(BASIC)$ (BASFLAGS) $(MMS$SOURCE)
```

Sometimes this is not what is wanted and it is necessary to evaluate the definition completely. The following is an excerpt from the procedure that builds MMK's PCSI Product Definition File.

```
DOCUMENTATION ~= $(ADDPREFIX $(KITDIR),$(WILDCARD $(KITDIR)*.HTML)) -
    $(ADDPREFIX $(KITDIR),$(WILDCARD $(KITDIR)*.PDF)) -
    $(ADDPREFIX $(KITDIR),$(WILDCARD $(KITDIR)*.PS)) -
    $(ADDPREFIX $(KITDIR),$(WILDCARD $(KITDIR)*.TXT))
```

Without being able to evaluate the macro definition for `DOCUMENTATION` the list documentation files would actually evaluate to the function calls, not the results of the builtin function calls.

2.8 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.

2.9 Macro String Substitution

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

2.9.1 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 occurrences 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.

2.9.2 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 occurrences 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.

2.10 Builtin Functions

As well as symbol substitution it is also possible to call a number of builtin functions that can process text to automate and simplify build definitions. The following reference describes what each of these functions does, with examples.

ADDPREFIX

ADDPREFIX—Prefix List Elements

`$(ADDPREFIX prefix,text[...])`

DESCRIPTION

ADDPREFIX treats **text** as a whitespace delimited list of words. Each word is prefixed with **prefix** and returned in a space delimited list.

EXAMPLE

The following examples show how to call the ADDPREFIX builtin function and demonstrate the expected output.

```
LIST = CAT, DOG, SECRET, HEAVY
ALL :
  @ WRITE SYS$OUTPUT "Unprefixed = $(LIST)"
  @ WRITE SYS$OUTPUT "Prefixed   = $(ADDPREFIX TOP ,$(LIST))"
```

Description file to demonstrate the ADDPREFIX function.

```
Unprefixed = CAT, DOG, SECRET, HEAVY
Prefixed   = TOP CAT, TOP DOG, TOP SECRET, TOP HEAVY
```

The output the above description file would generate.

ADDSUFFIX—Suffix List Elements

`$(ADDSUFFIX suffix,text[...])`

DESCRIPTION

ADDSUFFIX treats **text** as a whitespace delimited list of words. Each word is suffixed with **suffix** and returned in a space delimited list.

EXAMPLE

The following examples show how to call the ADDSUFFIX builtin function and demonstrate the expected output.

```
LIST = OX VAX
SUFFIX = EN,
ALL :
  @ WRITE SYS$OUTPUT "Unsuffixd = $(LIST)"
  @ WRITE SYS$OUTPUT "Suffixd    = $(ADDSUFFIX $(SUFFIX),$(LIST))"
```

Description file to demonstrate the ADDSUFFIX function. Notice that the suffix to be appended to each word is a symbol. This is so that the text *EN*, will recognize the ',' character as part of the suffix and not an argument delimiter in the builtin function.

```
Unsuffixd = OX VAX QUICK LENGTH STRENGTH
Suffixd    = OXEN, VAXEN, QUICKEN, LENGTHEN, STRENGTHEN,
```

The output the above description file would generate.

AND—Logical And

`$(AND condition[,condition[,...]])`

DESCRIPTION

The AND function provides a logical and operation. Each of the **condition** arguments are expanded until one evaluates to the empty string. The arguments are evaluated individually and consecutively, so if one should fail (evaluate to an empty string) then the remaining conditions will **not** be evaluated.

If all conditions evaluate to a non-empty string, then the result of this function is the expansion of the final condition. Otherwise this function will return the empty string.

EXAMPLE

The following examples show how to call the AND builtin function and demonstrate the expected output.

```
A = A
B = B
C = C
D =
FOO = $(AND $(A),$(B),$(C))
BAR = $(AND $(A),$(B),$(C),$(D))
ALL :
    @ WRITE SYS$OUTPUT "FOO = $(FOO)"
    @ WRITE SYS$OUTPUT "BAR = $(BAR)"
```

Description file to demonstrate the AND function.

```
FOO = C
BAR =
```

The output the above description file would generate.

BASENAME—Get File Specification Without Type

`$(BASENAME text[...])`

DESCRIPTION

BASENAME returns the device, directory and file name portions of the specifications found in **text**. The result does not include the file type. If multiple specifications are supplied, they will each be evaluated and returned in a space delimited list.

EXAMPLE

The following examples show how to call the BASENAME builtin function and demonstrate the expected output.

```
FOO = $(BASENAME SYS$LOGIN:LOGIN.COM)
ALL :
  @ WRITE SYS$OUTPUT "FOO = $(FOO)"
```

Description file to demonstrate the BASENAME function.

```
FOO = USER$:[FLYNN]LOGIN
```

The output the above description file would generate.

CALL—Call User-defined Function

`$(CALL macro[,param[,...]])`

DESCRIPTION

The CALL function makes it possible for a user to develop their own macro functions. To call a macro function, specify the name in **macro**, not the substitution, and pass up to 31 arguments in **param**.

Before **macro** is call (looked up and resolved) the arguments passed in **param** are resolved and defined as symbols \$(1)-\$(31), with \$(0) being **macro**.

It is important to remember certain escaping rules when defining callable macros. MMK supports the concept of deferred substitution (see Section 2.8 by using `$(` and `)` as the substitution delimiters. It is necessary to use this method of substitution when specifying the macro arguments outside of any builtin function call, for example:

```
REVERSE = ${2} ${1}
FOO = $(CALL REVERSE,A,B)
```

EXAMPLE

The following examples show how to call the CALL builtin function and demonstrate the expected output. Another example can be found under the documentation for FINDSTRING.

```
ORIGINS = $(FOREACH VAR,$(1),$(ORIGIN $(VAR)))
ECHO = WRITE SYS$OUTPUT
ALL :
    @ $(ECHO) "$(CALL ORIGINS,MMSALPHA MMSIA64 MMSVAX)"
```

This description file shows the definition of the user defined function **ORIGINS**, which is a multi-argument version of the ORIGIN builtin.

This example also demonstrates that because the \$(1) substitution is made within the FOREACH call, there is no need to use deferred substitution.

```
UNDEFINED UNDEFINED SPECIAL
```

The output the above description file would generate when run on a VAX system.

COLLAPSE—Collapse Whitespace

`$(COLLAPSE text)`

DESCRIPTION

The COLLAPSE function removes all whitespace from **text**.

EXAMPLE

The following examples show how to call the COLLAPSE builtin function and demonstrate the expected output.

```
FOO = $(COLLAPSE 1 2 3 4 5 6 7) 8 9 10
ECHO = WRITE SYS$OUTPUT

ALL :
  @ $(ECHO) "FOO = $(FOO)"
```

Description file to demonstrate the COLLAPSE function.

```
FOO = 1234567 8 9 10
```

The output the above description file would generate.

DIR—Get Device and Directory

`$(DIR text[...])`

DESCRIPTION

DIR returns the device and directory portion of the specifications found in **text**. If multiple specifications are supplied, they will each be evaluated and returned in a space delimited list.

To retrieve only the directory, please see **DIRECTORY**.

EXAMPLE

The following examples show how to call the DIR builtin function and demonstrate the expected output.

```
FOO = $(DIR SYS$LOGIN)
ECHO = WRITE SYS$OUTPUT
ALL :
    @ $(ECHO) "FOO = $(FOO)"
```

Description file to demonstrate the DIR function.

```
FOO = USER$:[FLYNN]
```

The output the above description file would generate.

DIRECTORY—Get Directory

`$(DIRECTORY text[...])`

DESCRIPTION

DIRECTORY returns only the directory portion of the specifications found in **text**. This is different to DIR which returns both the device and directory.

If multiple specifications are supplied, they will each be evaluated and returned in a space delimited list.

EXAMPLE

The following examples show how to call the DIRECTORY builtin function and demonstrate the expected output.

```
FOO = $(DIRECTORY SYS$LOGIN)
ECHO = WRITE SYS$OUTPUT
ALL :
    @ $(ECHO) "FOO = $(FOO)"
```

Description file to demonstrate the DIRECTORY function.

```
FOO = [FLYNN]
```

The output the above description file would generate.

FILENAME

FILENAME—Get Filename

`$(FILENAME text[...])`

DESCRIPTION

`FILENAME` returns only the file name portion of the specifications found in **text**. If multiple specifications are supplied, they will each be evaluated and returned in a space delimited list.

EXAMPLE

The following examples show how to call the `FILENAME` builtin function and demonstrate the expected output.

```
FOO = $(FILENAME SYS$LOGIN:LOGIN.COM)
ECHO = WRITE SYS$OUTPUT
```

```
ALL :
  @ $(ECHO) "FOO = $(FOO)"
```

Description file to demonstrate the `FILENAME` function.

```
FOO = LOGIN
```

The output the above description file would generate.

FILETYPE—Get File Type

`$(FILETYPE text[...])`

DESCRIPTION

FILETYPE returns only the file type portion of the specifications found in **text**. If multiple specifications are supplied, they will each be evaluated and returned in a space delimited list.

EXAMPLE

The following examples show how to call the FILETYPE builtin function and demonstrate the expected output.

```
FOO = $(FILENAME SYS$LOGIN:LOGIN.COM)
ECHO = WRITE SYS$OUTPUT
```

```
ALL :
  @ $(ECHO) "FOO = $(FOO)"
```

Description file to demonstrate the FILETYPE function.

```
FOO = .COM
```

The output the above description file would generate.

FILEVERSION—Get File Version

`$(FILEVERSION text[...])`

DESCRIPTION

FILEVERSION returns only the file version portion of the specifications found in **text**. If multiple specifications are supplied, they will each be evaluated and returned in a space delimited list.

FILEVERSION retrieves the version information calling the \$PARSE and then the \$SEARCH system services. In the event that the file does not exist, the version number is returned as ";" so that a missing element does not disrupt the list.

EXAMPLE

The following examples show how to call the FILEVERSION builtin function and demonstrate the expected output.

```
FOO = $(FILEVERSION SYS$LOGIN:LOGIN.COM SYS$LOGIN:DOESNOTEXIST.TXT)
ALL :
  @ $(ECHO) "FOO = $(FOO)"
```

Description file to demonstrate the FILEVERSION function.

```
FOO = ;17 ;
```

The output the above description file would generate.

FILTER—Filter Words

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

DESCRIPTION

FILTER treats **text** as a whitespace delimited list for words. It compares each word against **pattern**, which itself is a whitespace delimited list of patterns, to determine if it should be included in the result. If the pattern matches the word, the word is included in the space delimited result list. The pattern wildcard characters are `'%'` and `'*'`.

EXAMPLE

The following examples show how to call the FILTER builtin function and demonstrate the expected output.

```
SOURCES = FOO.C BAR.C BAZ.S UGH.H
ALL :
  @ WRITE SYS$OUTPUT "$(FILTER *.C *.S,$(SOURCES))"
  @ WRITE SYS$OUTPUT "$(FILTER *.H,$(SOURCES))"
  @ WRITE SYS$OUTPUT "$(FILTER *.T,$(SOURCES))"
```

Description file to demonstrate the FILTER function.

```
FOO.C BAR.C BAZ.S
UGH.H
```

The output the above description file would generate.

FILTER-OUT—Filter Out Words

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

DESCRIPTION

`FILTER` treats **text** as a whitespace delimited list for words. It compares each word against **pattern**, which itself is a whitespace delimited list of patterns, to determine if it should be included in the result. If the pattern matches the word, the word is not included in the space delimited result list. The pattern wildcard characters are `'%'` and `'*'`.

EXAMPLE

The following examples show how to call the `FILTER-OUT` builtin function and demonstrate the expected output.

```
SOURCES = FOO.C  BAR.C  BAZ.S  UGH.H
ALL :
@ WRITE SYS$OUTPUT "$(FILTER-OUT *.C *.S,$(SOURCES))"
@ WRITE SYS$OUTPUT "$(FILTER-OUT *.H,$(SOURCES))"
@ WRITE SYS$OUTPUT "$(FILTER-OUT *.T,$(SOURCES))"
```

Description file to demonstrate the `FILTER-OUT` function.

```
UGH.H
FOO.C BAR.C BAZ.S
FOO.C BAR.C BAZ.S UGH.H
```

The output the above description file would generate.

FINDSTRING—String Search

`$(FINDSTRING find,text)`

DESCRIPTION

FINDSTRING searches **text** for any occurrence of **find**. If **find** is found then the result is **find**. Otherwise the result is an empty string.

EXAMPLE

The following examples show how to call the FINDSTRING builtin function and demonstrate the expected output.

```
ISPRESNT = ${IF $(FINDSTRING $(1),$(2)),YES,NO}
TEXT = KERMIT PIGGY FOZZIE
ALL :
  @ WRITE SYS$OUTPUT "LIST = $(LIST)"
  @ WRITE SYS$OUTPUT "1. KERMIT? $(CALL ISPRESNT,KERMIT,$(TEXT))"
  @ WRITE SYS$OUTPUT "2. GONZO? $(CALL ISPRESNT,GONZO,$(TEXT))"
```

Description file to demonstrate the FINDSTRING function. In this example FINDSTRING is being called indirectly via a macro call. For further information on this, please see the documentation for the CALL builtin function.

```
LIST = KERMIT PIGGY FOZZIE
1. KERMIT? YES
2. GONZO? NO
```

The output the above description file would generate.

FIRSTWORD—Get First Word

`$(FIRSTWORD text[...])`

DESCRIPTION

FIRSTWORD fetches the first word in the whitespace separated list in **text**. This is equivalent to:

`$(word 1, text)`

EXAMPLE

The following examples show how to call the FIRSTWORD builtin function and demonstrate the expected output.

```
LIST = kermit the frog
FOO = $(FIRSTWORD $(LIST))
BAR = $(WORD 1,$(LIST))
ALL :
    @ WRITE SYS$OUTPUT "$ (FOO) = $(BAR)"
```

Description file to demonstrate the FIRSTWORD function.

```
kermit = kermit
```

The output the above description file would generate.

FOREACH—Repeat Expansion

`$(FOREACH macro,list[...text],text)`

DESCRIPTION

FOREACH provides a basic "for" loop construct by treating **list** as a whitespace delimited list of words and executing **text** for each word in **list**. The temporary variable **macro** is defined to the next word before each execution of **text**.

EXAMPLE

The following examples show how to call the FOREACH builtin function and demonstrate the expected output. An example of FOREACH can also be found in the examples for CALL.

```
LETTERS = A B C D
ALL :
  @ WRITE SYS$OUTPUT "$(FOREACH LETTER,$(LETTERS),A$(LETTER))"
```

Description file to demonstrate the FIRSTWORD function.

```
AA AB AC AD
```

The output the above description file would generate.

IF—Conditional Evaluation

`$(IF condition,then[,else])`

DESCRIPTION

The IF function evaluates **condition**. If it evaluates to a non-empty string, then the **then** argument is evaluated and returned as the result. Otherwise, if **condition** evaluates to an empty string (false) and the **else** argument is present, it is evaluated and returned as the result.

The **then** and **else** arguments are not evaluated until **condition** has been evaluated. Then only the appropriate argument is evaluated.

EXAMPLE

The following examples show how to call the IF builtin function and demonstrate the expected output.

```
TRUE = TRUE
FALSE =

FOO = $(IF $(TRUE),TRUE)
BAR = $(IF $(FALSE),,FALSE)
ALL :
    @ WRITE SYS$OUTPUT "FOO = $(FOO)"
    @ WRITE SYS$OUTPUT "BAR = $(BAR)"
```

Description file to demonstrate the IF function.

```
FOO = TRUE
BAR = FALSE
```

The output the above description file would generate.

JOIN—Merge Lists

`$(JOIN list,text)`

DESCRIPTION

JOIN merges the whitespace delimited lists in **list** and **text**. If the length of the two lists are not the same, then the remainder of the longer list is concatenated to the result.

EXAMPLE

The following examples show how to call the JOIN builtin function and demonstrate the expected output.

```
FOO = $(JOIN A B C D, 1 2 3 4)
BAR = $(JOIN A      , 1 2 3 4)
ALL :
  @ WRITE SYS$OUTPUT "FOO = $(FOO)"
  @ WRITE SYS$OUTPUT "BAR = $(BAR)"
```

Description file to demonstrate the JOIN function.

```
FOO = A1 B2 C3 D4
BAR = A1 2 3 4
```

The output the above description file would generate.

LASTWORD

LASTWORD—Get Last Word

`$(LASTWORD text ...)`

DESCRIPTION

LASTWORD fetches the last word in the whitespace separated list in **text**. This is equivalent to:

```
$(word $(words text), text)
```

EXAMPLE

The following examples show how to call the LASTWORD builtin function and demonstrate the expected output.

```
LIST = kermit the frog
FOO = $(LASTWORD $(LIST))
BAR = $(WORD $(WORDS $(LIST)),$(LIST))
ALL :
    @ WRITE SYS$OUTPUT "$ (FOO) = $(BAR)"
```

Description file to demonstrate the LASTWORD function.

```
frog = frog
```

The output the above description file would generate.

NOTDIR—Get Filename

`$(NOTDIR text[...])`

DESCRIPTION

NOTDIR is the opposite of DIR. NOTDIR returns the file name and type portion of the specifications found in **text**. If multiple specifications are supplied, they will each be evaluated and returned in a space delimited list.

To retrieve only filename, type or version, consider using FILENAME, FILETYPE or FILEVERSION respectively.

Note: NOTDIR performs a "syntax only" parse. It does not test for the existence of the file specified before parsing.

EXAMPLE

The following examples show how to call the NOTDIR builtin function and demonstrate the expected output.

```
FOO = $(NOTDIR SYS$LOGIN:A.A)
ECHO = WRITE SYS$OUTPUT
ALL :
  @ $(ECHO) "FOO = $(FOO)"
```

Description file to demonstrate the NOTDIR function.

```
FOO = A.A
```

The output the above description file would generate.

OR—Logical Or

`$(OR condition[,condition[,...]])`

DESCRIPTION

The OR function provides a logical or operation. Each of the **condition** arguments are expanded until one evaluates to a non-empty string. The arguments are evaluated individually and consecutively, so if one should succeed (evaluate to a non-empty string) then the remaining conditions will **not** be evaluated.

If all conditions evaluate to an empty string, then the result of this function is the empty string. Otherwise this function will return the result of the last condition evaluated.

EXAMPLE

The following examples show how to call the OR builtin function and demonstrate the expected output.

```
A =  
B =  
C =  
D = D  
FOO = $(OR $(A),$(B),$(C))  
BAR = $(OR $(A),$(B),$(C),$(D))  
ALL :  
  @ WRITE SYS$OUTPUT "FOO = $(FOO)"  
  @ WRITE SYS$OUTPUT "BAR = $(BAR)"
```

Description file to demonstrate the OR function.

```
FOO =  
BAR = D
```

The output the above description file would generate.

ORIGIN—Symbol Origin

\$(ORIGIN macro)

DESCRIPTION

ORIGIN returns a string describing in which context **macro** was defined. Table 2–3 lists the possible results and what they mean.

Table 2–3 Macro Definition Origins

Type	Description
UNDEFINED	The symbol is not yet defined.
FILE	These symbols have been defined within the description file.
COMMAND LINE	These symbols have been defined through use of the /MACRO= command line qualifier.
SPECIAL	Special symbols are defined per target, these include \$(MMS\$TARGET) and \$(MMS\$SOURCE).
DEFAULT	This includes all symbols defined by the default rules.
CLI SYMBOL	This macro takes its definition from a DCL symbol.
TEMPORARY	These symbols are defined through the use of CALL, like \$(0) and \$(1), or FOREACH and the definition of the loop symbol.

EXAMPLE

The following examples show how to call the ORIGIN builtin function and demonstrate the expected output.

```

FOO = BAR
BAR = $(ORIGIN 0)
ALL :
  @ WRITE SYS$OUTPUT "MMSVAX"           = $(ORIGIN MMSVAX) "
  @ WRITE SYS$OUTPUT "MMSALPHA"        = $(ORIGIN MMSALPHA) "
  @ WRITE SYS$OUTPUT "CC"              = $(ORIGIN CC) "
  @ WRITE SYS$OUTPUT "CALL ARGUMENT 0" = $(CALL BAR) "
  @ WRITE SYS$OUTPUT "FOO"             = $(ORIGIN FOO) "
  @ WRITE SYS$OUTPUT "VARIANT"         = $(ORIGIN VARIANT) "
  @ WRITE SYS$OUTPUT "CLISYM"          = $(ORIGIN CLISYM) "

```

Description file to demonstrate the ORIGIN function.

```

MMSVAX      = SPECIAL
MMSALPHA    = UNDEFINED
CC          = DEFAULT
CALL ARGUMENT 0 = TEMPORARY
FOO         = FILE
VARIANT     = COMMAND LINE
CLISYM      = CLI SYMBOL

```

The output the above description file would generate if run on a VAX system. With the following commands:

```

$ CLISYM == 1
$ MMK/MACRO=VARIANT=1

```

PATSUBST—Pattern Substitution

\$(PATSUBST pattern, to, text)

DESCRIPTION

PATSUBST replaces all instances of **pattern** with **to** in **text**. However, unlike the function SUBST this builtin allows the specification of a pattern. Table 2–4 details the pattern operators.

Table 2–4 PATSUBST Pattern Operators

Operator	Match	Replace
*	Match any sequence of any length.	Insert the entire matched sequence.
%	Match any single character.	Insert the first character of the matched sequence.

When matching the words in **text** if a match is found, the wild-carded portions of the match are placed in a queue. Therefore, when specifying **to** it is also possible to include pattern operators which will insert the wildcarded portions back into the output.

Note: Unlike the substitution mentioned in Section 2.9, Macro String Substitution the comparison done by PATSUBST is case-sensitive.

EXAMPLE

The following examples show how to call the PATSUBST builtin function and demonstrate the expected output.

```
LIST = FIRST.C SECOND.C THIRD.C
FOO = $(PATSUBST *.C,*.OBJ,$(LIST))
BAR = $(PATSUBST *.*,*.%,$(FOO))
ALL :
  @ WRITE SYS$OUTPUT "FOO = $(FOO)"
  @ WRITE SYS$OUTPUT "BAR = $(BAR)"
```

Description file to demonstrate the PATSUBST function.

```
FOO = FIRST.OBJ SECOND.OBJ THIRD.OBJ
BAR = FIRST.O SECOND.O THIRD.O
```

The output the above description file would generate.

SORT—Sort Words

`$(SORT text[...])`

DESCRIPTION

SORT treats **text** as a whitespace delimited list of words. These words are then sorted into into lexical order, removing duplicates and returned to the caller.

EXAMPLE

The following examples show how to call the SORT builtin function and demonstrate the expected output.

```
ECHO = WRITE SYS$OUTPUT
MUPPETS = PIGGY FOZZIE KERMIT GONZO BEAKER ROWLF
DILBERT = DILBERT ALICE WALLY ASOK DOGBERT RATBERT
ALL :
  @ $(ECHO) "$(SORT $(DILBERT))
  @ $(ECHO) "$(SORT $(MUPPETS))
```

Description file to demonstrate the SORT function.

```
ALICE ASOK DILBERT DOGBERT RATBERT WALLY
BEAKER FOZZIE GONZO KERMIT PIGGY ROWLF
```

The output the above description file would generate.

STRIP—Compress Whitespace

`$(STRIP text)`

DESCRIPTION

STRIP replaces all whitespace characters with a single space as well as removing the leading and trailing whitespace from **text**.

EXAMPLE

The following examples show how to call the STRIP builtin function and demonstrate the expected output.

```
FOO = $(STRIP This has lots of space )
ALL :
@ WRITE SYS$OUTPUT "FOO = $(FOO)"
```

Description file to demonstrate the STRIP function.

```
FOO = This has lots of space
```

The output the above description file would generate.

SUBST—String Substitution

`$(SUBST from,to,text)`

DESCRIPTION

SUBST replaces all instances of **from** found in **text** with **to**.

Note: This builtin function, like PATSUBST, is case sensitive in its matching. For case-insensitive string substitution please use, Section 2.9, Macro String Substitution.

EXAMPLE

The following examples show how to call the SUBST builtin function and demonstrate the expected output.

```
ALL :
@ WRITE SYS$OUTPUT "$(SUBST ee,EE,feet on the street)"
@ WRITE SYS$OUTPUT "$(SUBST EE,ee,feet on the street)"
```

Description file to demonstrate the SUBST function.

```
fEEt on the strEEt
feet on the street
```

The output the above description file would generate.

WILDCARD—File Search

\$(WILDCARD text)

DESCRIPTION

WILDCARD searches for the files matching the specification in **text** and returns the name and type portions as a space delimited list. It is also possible to list multiple specifications in **text**, each must be separated by whitespace. The wildcard characters '%' and '*' can also be used.

It is possible to specify multiple search specifications by separating them with space.

EXAMPLE

The following examples show how to call the WILDCARD builtin function and demonstrate the expected output.

```
FOO = $(WILDCARD BLISS32_BLD:LEX*.BLI)
BAR = $(WILDCARD BLISS32_BLD:K*.BLI BLISS32_BLD:LEX*.BLI)
ALL :
  @ WRITE SYS$OUTPUT "FOO = $(FOO)"
  @ WRITE SYS$OUTPUT "BAR = $(BAR)"
```

Description file to demonstrate the WILDCARD function.

```
FOO = LEXAN.BLI LEXFUN.BLI
BAR = KFOLD.BLI KOST.BLI LEXAN.BLI LEXFUN.BLI
```

The output the above description file would generate.

WORD—Return the n'th Word

`$(WORD n, text)`

DESCRIPTION

WORD treats **text** as a space delimited (multiple consecutive whitespace characters are treated as one) array of words. The parameter **n** is an index into this array, specifying which word to return.

If you specify a value of **n** that is less than 1 or greater than the number of elements in **text**, then the result will be an empty string, "". The builtin function WORDS can be used to determine the upper limit of the array in **text**.

EXAMPLE

The following examples show how to call the WORDS builtin function and demonstrate the expected output.

```
FOO = $(WORD 2, first second third)
BAR = $(WORD 5, kermit the frog)
ALL :
  @ WRITE SYS$OUTPUT "FOO = $(FOO)"
  @ WRITE SYS$OUTPUT "BAR = $(BAR)"
```

Description file to demonstrate the WORD function.

```
FOO = second
BAR =
```

The output the above description file would generate.

WORDLIST

WORDLIST—Extract Words

`$(WORDLIST start,end,text[...])`

DESCRIPTION

WORDLIST treats **text** as a whitespace delimited list of words. The words, start with **start** and ending at **end** (inclusive) are extracted from **text** and returned as a space delimited list. The first word in **text** is *1*.

EXAMPLE

The following examples show how to call the WORDLIST builtin function and demonstrate the expected output.

```
ALL :  
  @ WRITE SYS$OUTPUT "$(WORDLIST 2, 3, FOO BAR BAZ)"
```

Description file to demonstrate the WORDLIST function.

```
BAR BAZ
```

The output the above description file would generate.

WORDS—Count Words

`$(WORDS text)`

DESCRIPTION

WORDS counts the number of words in **text**. The contents of **text** is treated as a space delimited (multiple consecutive whitespace characters are treated as one) array of words.

EXAMPLE

The following examples show how to call the WORDS builtin function and demonstrate the expected output.

```
FOO = $(WORDS first second third)
BAR = $(WORDS )
ALL :
    @ WRITE SYS$OUTPUT "FOO = $(FOO)"
    @ WRITE SYS$OUTPUT "BAR = $(BAR)"
```

Description file to demonstrate the WORDS function.

```
FOO = 3
BAR = 0
```

The output the above description file would generate.

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

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.

/EXTENDED_SYNTAX

...describe what this does, largely for MMS compatibility...

/[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 | definition...

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[=ALL]

Controls whether MMK echoes action lines to `SYS$OUTPUT`. Enabled by default. By specifying the keyword **ALL** it is possible to force MMK to echo all command output, including that which is prefixed with `@`. For debugging purposes it is possible to specify the keyword **ALL** which will cause MMK to verify all action lines, even those prefixed with `@` (silent) action line prefix. This also overrides the `.SILENT` directive.

/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).

Differences between MMK and DEC/MMS

- 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, `__AXP__` for builds on Alpha systems, and `__I64__` for builds on IA64 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-substitution function for macro references.

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

A.3 Other Differences

Besides the feature differences already 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.
- MMK is more lenient about applying inference rules. It only checks for a match on the suffix and, if present, the prefix. If you have a dependency with an explicit target and source that match by suffix (and possibly prefix), even if the base names do not match, the inference rule can be applied. MMS requires that base names match to apply an inference rule. Note that MMK's behavior may change to match MMS in a future release.

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 are shown in Figure B-1; those for Alpha systems are shown in Figure B-2; and those for IA64 systems are shown in Figure B-3.

Figure B-1 MMK default dependency rules - VAX

```
! MMK_DEFAULT_RULES.MMS
!
! COPYRIGHT © 1993, 1994, 1997 MADGOAT SOFTWARE. ALL RIGHTS RESERVED.
! COPYRIGHT © 2012 ENDLESS SOFTWARE SOLUTIONS. 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.
! 22-AUG-2012 V2.4 Sneddon Add GENCAT support.
!
!
! These symbols can be used to distinguish MMK from DEC's DEC/MMS product
! using .IFDEF directives.
!
__MATTMMS__ = __MATTMMS__
__MMK__ = __MMK__
__MMKV32__ = 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 $(L32) .CAT .C .CXX .BAS .B32 .BLI
.COBI .COR .DBL .RPG .SCN .PLI .PEN .PAS .MAC .MAR .MSG .MSGX .CLD .R32 -
.REQ .TXT .H .MEM .HLP .RNH .RNO .MMS .DAT .OPT .SDML .COM -
.C~ .CXX~ .BAS~ .B32~ .BLI~ .FOR~ .COB~ .COR~ .DBL~ .RPG~ .SCN~ -
.PLI~ .PAS~ .MAC~ .MAR~ .MSG~ .CLD~ .R32~ .REQ~ .TXT~ -
.H~ .HLP~ .RNH~ .RNO~ .MMS~ .DAT~ .OPT~ .SDML~ .COM~
!
LINK = LINK
LINKFLAGS = /EXEC=$(MMS$TARGET)
```

Figure B-1 Cont'd on next page

Built-in Dependency Rules

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)

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

.CORS$(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)

.MSGX.CAT :
    $(GENCAT)$ (GENCATFLAGS) $(MMS$SOURCE) $(MMS$TARGET)
GENCAT    = GENCAT
GENCATFLAGS =

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

```

Figure B-1 Cont'd on next page

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

```
.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)

.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) $(CMSCOMM)
```

Figure B-1 Cont'd on next page

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

```
.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

.MSGX~.MSGX :
@ IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB $(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MSGX $(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

.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) $(CMSCO

.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

Figure B-2 Cont'd on next page

Built-in Dependency Rules

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

```
! MMK_DEFAULT_RULES_AXP.MMS
!
!   COPYRIGHT © 1993, 1997 MADGOAT SOFTWARE. ALL RIGHTS RESERVED.
!   COPYRIGHT © 2012 ENDLESS SOFTWARE SOLUTIONS. 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.
!   22-AUG-2012 V2.4   Sneddon      Add GENCAT support.
!
!
! This symbol can be used to distinguish MMK from DEC's DEC/MMS product
! using .IFDEF directives.
!
__MATTMMS__ = __MATTMMS__
__MMK__ = __MMK__
__MMKV32__ = 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) .CAT .C .CXX .BAS .B32 .BLI
              .COB .COR .DBL .RPG .SCN .PLI .PEN .PAS .MAC .MAR .M64 .MSG .MSGX .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)
```

Figure B-2 Cont'd on next page

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

```
.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)

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

.CORS$(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)

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

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

Figure B-2 Cont'd on next page

Built-in Dependency Rules

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

```
.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

.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)

.MSGX.CAT :
    $(GENCAT)$ (GENCATFLAGS) $(MMS$SOURCE) $(MMS$TARGET)
GENCAT    = GENCAT
GENCATFLAGS =

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) $(CMS$
```

Figure B-2 Cont'd on next page

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

```

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.COBS~.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

.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) $(CMSCOM

```

Figure B-2 Cont'd on next page

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

```
.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

.MSGX~.MSGX :
@ IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB $(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MSGX $(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

.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) $(CMSCO

.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–3 MMK default dependency rules - IA64

```
! MMK_DEFAULT_RULES_I64.MMS
!
!   COPYRIGHT © 2004 MADGOAT SOFTWARE. ALL RIGHTS RESERVED.
!   COPYRIGHT © 2012 ENDLESS SOFTWARE SOLUTIONS. ALL RIGHTS RESERVED.
!
! Default build rules for use with MMK. (for OpenVMS IA64)
!
! Modification history:
!
!   03-MAY-2004 V1.0 Madison Initial coding (from ACP)
!   22-AUG-2012 V2.4 Sneddon Add GENCAT support.
!
```

Figure B–3 Cont'd on next page

Figure B-3 (Cont.) MMK default dependency rules - IA64

```

!
! This symbol can be used to distinguish MMK from DEC's DEC/MMS product
! using .IFDEF directives.
!
__MATTMMS__ = __MATTMMS__
__MMK__ = __MMK__
__MMKV32__ = 1
!
! These symbols can be used to distinguish an IA64-based build from a
! VAX or Alpha one.
!
__IA64__ = 1
__I64__ = 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) .CAT .C .CXX .BAS .B32 .BLI
               .COB .COR .DBL .RPG .SCN .PLI .PEN .PAS .MAC .MAR .M64 .MSG .MSGX .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-3 Cont'd on next page

Built-in Dependency Rules

Figure B-3 (Cont.) MMK default dependency rules - IA64

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

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

.CORS$(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)

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

.MARS$(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-3 Cont'd on next page

Figure B-3 (Cont.) MMK default dependency rules - IA64

```

.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)

.MSGX.CAT :
    $(GENCAT)$ (GENCATFLAGS) $(MMS$SOURCE) $(MMS$TARGET)
GENCAT     = GENCAT
GENCATFLAGS =

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) $(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)

.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)

.COBI~.COBI :
@ IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB $(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).COBI $(CMSFLAGS) $(CMSCOMMENT)

```

Figure B-3 Cont'd on next page

Figure B-3 (Cont.) MMK default dependency rules - IA64

```
.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

.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

.MSGX~.MSGX :
@ IF "$(MMS$CMS_LIBRARY)" .NES. "" THEN DEFINE/USER CMS$LIB $(MMS$CMS_LIBRARY)
$(CMS) FETCH $(MMS$CMS_ELEMENT) /OUTPUT=$(MMS$TARGET_NAME).MSGX $(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
```

Figure B-3 Cont'd on next page

Figure B-3 (Cont.) MMK default dependency rules - IA64

```
.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) $(CMSCC

.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) $(CMSCC

.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) $(CMSCC

.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) $(CMSCC

.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) $(CMSCC

.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) $(CMSCC

.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) $(CMSCC
```
