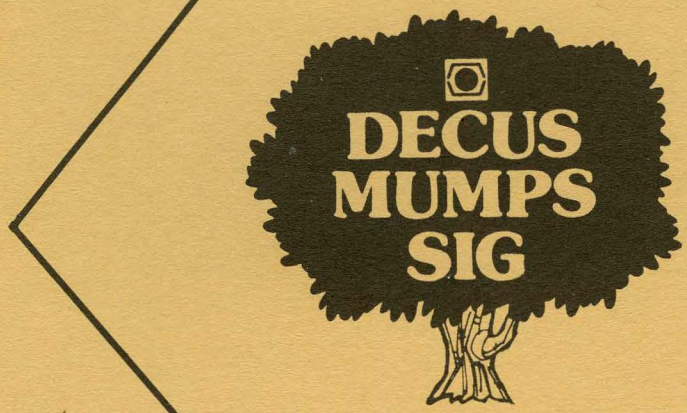
THE HEAP

STRUCTURED LANGUAGES



May 1984 Issue

Printed in the U.S.A.

The MUMPS SIG

Did not submit material for this issue

Send your submissions for the next issue to:

Jim Bernard

Data Processing
Kettering Medical Center
3935 Southern Blvd.
Kettering, OH 45429

The HEAP

From the Editor

John R. Barr, University of Montana, Missoula, MT

This issue of the Heap isn't all I promised in the previous issue. However, it does contain some interesting material and continues to be mainly articles submitted by Kathy Hornbach. Is anyone reading this newsletter?? I would like to see more articles on languages and their application. Our Modula-2 coordinator is putting together a list of sources for Modula-2 compilers that run on DEC equipment and operating systems. Is anyone out there using Pascal?? What are you doing with it that you couldn't do with FORTRAN?? How about Ada? Surely everyone will be using Ada as soon as the compilers and programming support environments are available. Any comments??

I have included a list of all the people invloved with the LTSIG to the best of my knowledge. There may be others, but I don't know about them. If you need information about languages and tools, don't call me, call the people listed in the LTSIG leadership directory. I will just refer you to them.

The subscription fee structure will be changed for next year (July 84 to June 85). Each SIG newsletter will be separately priced according to the projected number of issues and page counts. We are planning to produce four newsletters next year with each newsletter containing approximately 40 pages. Your subscription cost will be $10.00 for the year. Look for the new subscription order form to appear in DECUSCOPE.

Finally, we need a new newsletter editor. I am going to resign as newsletter editor in June. We need someone who likes to edit and write to take over as editor. Please call Jim Livingston or me if you are interested.

John R. Barr
Department of Computer Science
University of Montana
Missoula, MT 59812
(406) 243-4807

LTSIG Leadership Directory

Chairman:
James W. Livingston, Jr.
Measurex Corporation
One Results Way
Cupertino, CA 95014
(408) 255-1500 x4468

Operating System Coordinator:
Alan Rizzuto
Emc Controls, Inc.
P.O. Box 242
Cockeysville, MD 21030
(301) 667-4800

IAS Coordinator: open

RSTS Coordinator: open

RSX-11M/M+ Coord: Alan Rizzuto

RT-11 Coordinator:
Michele Wong
DISC
3336 Bradshaw Road, Suite 340
Sacramento, CA 95827
(916) 363-7385

Unix Coordinator:
Rod Creason, Jr.
DISC
3336 Bradshaw Road, Suite 340
Sacramento, CA 95827
(916) 363-7385

VAX/VMX Coordinator:
Louise M. Wooley
Measurex Corporation
One Results Way
Cupertino, CA 95014
(408) 255-1500 x4452

Symposia Coordinator:
J. R. Westmoreland
Utah Power and Light
Systems and Computer Services, Room 184
1407 W. North Temple
Salt Lake City, Utah 84116
(801) 535-2387

Menu Coordinator:
Alan L. Folsom, Jr.
Enertec, Inc.
19 Jenkins Avenue
Lansdale, PA 19446

Tools Coordinator:	Kathy Hornbach Lear Siegler/Instrument Division 4141 Eastern SE, MS 121 Grand Rapids, MI 49508 (616) 241-8800
Member Services:	John R. Barr 810 Continental Way Missoula, MT 59803 (406) 728-0062 (evenings)
Librarian:	James Triplett Intermetrics, Inc. 701 Concord Avenue Cambridge, MA 02138 (617) 661-1840
Session Notes:	Mark Katz GTE Sylvania 77 A Street Needham, MA 02194 (617) 449-2000 x635
Newsletter:	(Soon to be open)
User Training:	open
Modula-2 Coordinator:	Jack R. Davis NAP Consumer Electronics 9041 Executive Park Drive, Suite 612 Knoxville, Tennessee 37923

Toolside Chat

If you want to learn about Software Tools, Spring DECUS '84 in Cincinnati (June 4-8) is the place to be. The theme for the Symposium is "Increased Programmer Productivity", which is exactly what Software Tools are intended to do. A Preliminary list of sessions indicates that the tools area is well-represented. DEC is sponsoring many sessions - on specific tools like CMS and MMS; on how they use tools internally; and "crystal balling" on tools they are thinking about developing.

Tools are also well represented in sessions presented by users. These include sessions on experiences with DEC tools, with home-grown tools, and with third-party tools. And there is a pre-symposium seminar (given by yours truly) on "Implementing a Software Development Environment". We have certainly come a long way from just two years ago, when the only session specifically on tools was one birds-of-a-feather!

In addition to the planned sessions, there will be plenty of opportunities for you to talk to DEC people and expert tool users informally - the LTSIG will have both a campground and a suite.

A partial list of the sessions a tools person would be interested in:

CMS and MMS - many sessions, including presentations from DEC on usage, features and futures, and user presentations/panels on experiences with these tools.

ADA (ADA is a trademark of the Department of Defense) - a whole afternoon of sessions on ADA and the ADA environment, both by DEC and by people from the ADA community.

Specific sessions include:

- Real World Use of Software Development Tools
- Languages and Tools Panel (DEC)
- VAX/VMS Software Development Environment - Productivity thru Methods and Tools (DEC)
- Programmer Productivity - Tools are not Enough (DEC)
- Software Tools User's Group Update
- Define the Software Process, then find the VMS Tools (DEC)
- Teams of Tools (DEC)
- Testing Software: Methods, Procedures and Practices

- LR(1) Parser Generator
- EMACS Editor
- Automating an Assurance Testing System
- Promoting the Acquisition and Use of Software Development Tools
- Computer-Aided Systems Analysis and Design Tools
- SPSS

Hope to see you there!

Problem and Fix for DECUS C Compiler

James B. Van Bokkelen, Manager, Software Development
Perception Technology Corporation
50 Shawmut Road
Canton, Massachusetts 02021
(617) 821-0320

I would like to report a problem and fix for IOGET.MAC, version 11, and IOGETC.MAC, version 6. Under RT11 emulation on RSTS/E, any use of ungetc() on a terminal will crash the program. This is because of the interaction between two fixes (in IOFOPA, for buffered RT11 terminal input, and in IOGETC, fo un-edited RSTS terminal input), resulting in there being no buffer for ungetc() to put the character in.

My approach was to take the RSTS native mode terminal READ code out of IOGETC, and put it in IOGET, as its fuctionality implied. In the process, I altered it to use the standard I/O buffering mechanism, which fixed ungetc(), and to trim all the normal RSTS terminator characters. This last was in an effort to duplicate the effect of .GTLIN under native RT11.

I understand DECUS has installed a VAX, and that you are on an internal DEC network. I have access to the ARPANET (jbvb@ml), and can (with some effort) get mail onto the DEC Engineering Network. If I had an address, I could send either just the diff output, or the complete modified modules via that means. I would also appreciate some feedback on the currency of the problems I am reporting. My distribution was obtained through the MIT PDP11 LUG in Summer, 1983. The compiler is at "Patch Level 8".

(Editor's note: I have the listings of the fixes and need to get them to someone who will include them in the next C release. Anyone interested should send me a self-addressed envelope or contact Jim directly.)

GETTING THE MOST OUT OF VAX-11 DEC/MMS

Operating System and Version: VAX/VMS Version 3.0 or later.

Software Version: DEC/MMS Version 1.0

CONTENTS

ABOUT THIS DOCUMENT

This is an informal document specifically prepared for the 1983 spring session of U.S. DECUS. It cannot be ordered from DIGITAL. It can, however, be copied, as long as the source is credited. There will be no updates to or revisions of this document. However, some information from this document may be included in the next version of the VAX-11 DEC/MMS User's Guide.

This document is aimed at users familiar with DEC/MMS. It is intended to supplement the information provided in the VAX-11 DEC/MMS User's Guide. The information presented here is not intended to help you get MMS to run; it is assumed that you are familiar with MMS, and that you have used it without major difficulties. It is also assumed that you have a more than cursory familiarity with DCL.

The primary objectives of this document are to present information to help you use MMS more effectively, and to give you a perspective on MMS broader than that provided in the User's Guide. This document suggests approaches to some specific tasks for which you might not have considered using MMS. You can use these approaches as they appear in this document, or you can modify them for other purposes. You are limited only by your own creativity.

CHAPTER 1

SIMPLE TRICKS

This chapter describes some simple tasks for which you can use MMS. Some of these tasks do not even require a description file.

1.1 CHECKING WHETHER FILES ARE UP-TO-DATE

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

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

MMS will go to the CMS library, check the time of FOO.BLI, and report whether FOO.EXE is up-to-date with respect to FOO.BLI. (The /SKIP qualifier ensures that the outcome of /CHECK is unaffected if any intermediate files do not exist.) Use this command line when the source is in your default CMS library, and specify the target file (FOO.EXE in this example) as the parameter to the MMS command.

1.2 USING MMS TO "FETCH AND BUILD"

Suppose you have a CMS library that contains DSR (DIGITAL Standard Runoff) source (.RNO) files, and you want to create a .MEM file from one of those source files. Instead of fetching the .RNO file from the CMS library and running DSR to create the .MEM file, you can simply type the MMS/CMS command and specify the .MEM file-to-be as the parameter on the command line. For example, suppose you want to create a memo, and the source file MEMO.RNO exists in your CMS library. To use MMS to build MEMO.MEM, you would type the following command line:

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

MMS will fetch MEMO.RNO from the CMS library, run DSR, and create MEMO.MEM. No description file is necessary. Remember to specify the file to be created as the parameter on the MMS command line.

NOTE

If you have a description file (DESCRIP.MMS or MAKEFILE.) containing the .MEM file-to-be as a target, MMS will also parse that description file.

1.3 USING THE /SKIP QUALIFIER

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

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

The description file for building MYPROG.EXE looks like this:

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

You have fetched A.BLI from a CMS library that contains the source files, and you have edited this file. To update the system to reflect the change to A.BLI, type this command:

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

MMS will update MYPROG.EXE. It will also create A.OBJ, since A.BLI has a time newer than that of the corresponding module in MYPROG.OLB. However, MMS will not create any other .OBJ files (such as B.OBJ and C.OBJ) because /SKIP was specified on the command line. If /SKIP was not specified, MMS would have fetched B.BLI and C.BLI, compiled them, and added the .OBJ files to MYPROG.OLB, even though they did not need to be updated.

NOTE

Require files must be present in the working directory for MMS to execute compiles. For example, suppose the description file contains the following rule:

```
A.OBJ : A.BLI DEFS.REQ
        BLISS A
```

If both A.BLI and DEFS.REQ are in the CMS library, but you fetch and change only A.BLI, MMS will not fetch DEFS.REQ. Therefore, A.OBJ will not be updated and the compile will have errors.

You may want to delete intermediate files once the main target is updated. See Section 2.6 for information on selectively deleting files.

CHAPTER 2

BEING CREATIVE WITH MMS

This chapter describes some creative things you can do with MMS.

2.1 GATHERING STATISTICS

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

2.1.1 Finding Out What is Missing from a CMS Library

Suppose that your sources for a particular software system are contained in a CMS library, and you want to know whether everything you need is there. To get a list of missing files, you could put a default action such as the following in your description file, using the .DEFAULT reserved keyword:

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

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

2.1.2 Creating a Checkpoint File

You can use MMS to create a checkpoint file that indicates when MMS completed targets. For example, suppose that your directory contains the source files FOO.C, BAR.C, and BAS.C. You want MMS to create .EXE files for each of these sources, and also to inform you when it completed each build. A description file to accomplish these tasks looks like the following example. This description file builds FOO.EXE, BAR.EXE, and BAS.EXE. It also creates a file called CHECK.PNT, which indicates the time the executable files were completed.

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

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

MAIN_TARGET : FOO.PNT BAR.PNT BAS.PNT
     MAIL CHECK.PNT <your process> -
     /SUBJECT="Build summary of $* ending at ''f$time()'"
     DELETE CHECK.PNT;
```

NOTE

> The executables will be built before the .PNT files are processed. Also, the .PNT files never really exist. They are simply a trick to allow the actions that produce the file to be localized in one place (the .EXE.PNT rule).

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

```
CC /NOLIST FOO.C
LINK /TRACE FOO.OBJ
IF "''F$SEARCH("CHECK.PNT")'" .EQS. "" THEN OPEN/WRITE CHECK CHECK.PNT
IF "''F$SEARCH("CHECK.PNT")'" .NES. "" THEN OPEN/APPEND CHECK CHECK.PNT
WRITE CHECK "Completed build of FOO.EXE at ''f$time()'"
CLOSE CHECK
CC /NOLIST BAR.C
LINK /TRACE BAR.OBJ
IF "''F$SEARCH("CHECK.PNT")'" .EQS. "" THEN OPEN/WRITE CHECK CHECK.PNT
IF "''F$SEARCH("CHECK.PNT")'" .NES. "" THEN OPEN/APPEND CHECK CHECK.PNT
```

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

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

```
From:    <your process>  3-MAY-1983 08:48
To:      <your process>
Subj: Build summary of MAIN_TARGET ending at  3-MAY-1983 08:48:06.85

Completed build of FOO.EXE at  3-MAY-1983 08:47:32.99
Completed build of BAR.EXE at  3-MAY-1983 08:47:49.65
Completed build of BAS.EXE at  3-MAY-1983 08:48:06.33
```

2.2 USING DCL COMMAND PROCEDURES IN DESCRIPTION FILES

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

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

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

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

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

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

NOTE

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

ANSWER.IN corresponds to the P1 parameter, and 15 is the polling interval (in minutes) that corresponds to the P2 parameter. ANSWER.IN might modify the environment in some way -- for example, it might set a CMS library at a point where MMS cannot find the right CMS library.

The $(MY_PROC) macro is not defined in the description file. This is because it is a DCL symbol. Section 2.3 describes how to use DCL symbols as macros.

2.3 USING DCL SYMBOLS AS MACROS

You can use DCL symbols to do things cleanly by invoking DCL symbols as macros in your description file.

One way of using DCL symbols as macros is to define a symbol for your process in your LOGIN.COM file. Then use that symbol as a macro in description files when you want MMS to send mail to you. See the second example in Section 2.2. The reason for defining a symbol for your process is to make sure that mail gets sent to the right place. If you have any subprocesses or batch jobs running, and you do not have a symbol used as a macro in your description file, MMS may not send mail to the process you wanted it sent to.

Another way of using DCL symbols is illustrated in Section 2.4.1.

2.4 CREATING AND USING TIME STAMPS

You can use MMS to create time stamps for such purposes as finding out whether anything has changed since the last time you built a system, and tracking the progress of the system.

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

2.4.1 Creating a Time Stamp File Using DCL Symbols

The following example description file creates the file CMSMODS.RPT. CMSMODS.RPT reports the number of project sources modified as indicated by replaces in the CMS library.

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

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

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

! Primary Target
MODS : INIT $(PROJECT_SOURCES)
        IF "''F$SEARCH("CMSMODS.RPT")'" .EQS. "" -
        THEN OPEN/WRITE CHECK CMSMODS.RPT
        IF "''F$SEARCH("CMSMODS.RPT")'" .NES. "" -
        THEN OPEN/APPEND CHECK CMSMODS.RPT
        WRITE CHECK "''MODS' MODIFICATIONS DETECTED AT ''F$TIME()'"
        CLOSE CHECK

INIT :
        MODS = 0
```

MODS is a DCL symbol used as a counter.

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

It is suggested that a description file such as the one in the example be run on or nearly on a daily basis. You may want to put the appropriate MMS command in your LOGIN.COM file.

2.4.2 Creating a Time Stamp File Using Included Files

Suppose you have the following directories and files:

[DIR1] contains FILE1.X

[DIR2] contains FILE2.Y

[DIR3] contains FILE3.Z

You want MMS to build a file reporting changes to these files. The following example description file creates the file CHANGES.DOC, which reports when changes were made to the source files.

```
.SILENT :

RECORD_CHANGE = INCLUDE CHANGE.REC

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

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

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

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

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

Note the use of the .SILENT reserved keyword. .SILENT prevents MMS from displaying all the action lines. Therefore, only two things will be displayed when MMS processes this description file:

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

2. If changes were made to the files, MMS will type out CHANGES.DOC as instructed in the REPORT_CHANGE action line. CHANGES.DOC will list the files that were changed, and the times the changes were made.

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

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

You must indent the lines from column 1 of the file by at least one space or tab. This is so MMS will interpret them as action lines when it includes the file into the description file. Also, when you use the $(RECORD_CHANGE) macro as an action line, you need not indent it because the lines are indented in the included file.

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

```
.SILENT :

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

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

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

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

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

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

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

2.5 CHECKING FOR REPLACEMENT OF CMS ELEMENTS

If more than one programmer is working on your project, you may want to wait for someone else to replace an element in the project CMS library before you do a particular task. Rather than asking that programmer every ten minutes whether he/she is done yet, you can have MMS automatically check for element replaces. To do so, create a command procedure that checks the CMS library at specified intervals to determine when the element is replaced. You also need to create a description file that tells MMS which element to look for, and to notify you when the element has been replaced. Such a description file might look like this:

```
THERE.TIM : NEEDED.FOR~  ! Or whatever the element's name is
        IF "''F$SEARCH("THERE.TIM")'" .NES "" -
        THEN MAIL NL: $(MY_PROC) -
        /SUBJECT="$< is back in the CMS library."
        DIR 1234567890 ! Causes MMS to abort with $STATUS = failure
```

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

```
$ CMS SET LIBRARY [YOURCMSLIB] ! The CMS library
$ SET DEFAULT [WORKINGDIR]     ! Your working directory
$ IF "''F$SEARCH("THERE.TIM")'" .EQS. "" THEN COPY NL: THERE.TIM
$ LOOP:
$ MMS
$ IF .NOT. $STATUS THEN EXIT
$ WAIT 0:5  ! or some interval
$ GOTO LOOP
```

When submitted to the batch queue, this command procedure will run MMS, which will check to see whether the element in the CMS library has a newer time than THERE.TIM. If it does not (that is, if the element has not been replaced in the CMS library), $STATUS will be 1, and MMS will wait the specified interval before trying again. If the element has been replaced, the first bit in $STATUS will be set to 0, and MMS will mail the message "NEEDED.FOR is back in the CMS library."

This procedure is doubly useful in that it will detect whether another person has reserved and replaced an element while you are waiting for the first person to finish with the one you are waiting for.

You can run this procedure in a subprocess (instead of submitting it to the batch queue) by typing the following command:

```
$ SPAWN/NOWAIT @FILENAME
```

FILENAME is the name of the command procedure.

2.6 SELECTIVELY DELETING FILES

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

1. Create a command procedure
2. Use a macro definition

These methods are described in the following sections.

2.6.1 Creating a Command Procedure

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

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

NOTE

> Usually, you will want to modify only the .OBJ.OLB rule to include these actions. However, you can modify all the rules you use to delete everything; just be extremely careful that you are deleting only what you want deleted.

The modified .OBJ.OLB rule will therefore look like this:

```
.OBJ.OLB :
     IF "''F$SEARCH("$@")'" .EQS. "" THEN $(LIBR)/CREATE $@
     $(LIBR) $(LIBRFLAGS) $@ $<
     IF "''F$SEARCH("DELETE.COM")'" .EQS. "" -
     THEN COPY NL: DELETE.COM
```

```
OPEN/APPEND DEL_FILE DELETE.COM
WRITE DEL_FILE "$ DELETE $<;"
```

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

```
DELETE : MYPROG.EXE ! Or whatever your main target is
       - @DELETE.COM
```

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

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

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

2.6.2 Using a Macro Definition

There are two ways of using macros to selectively delete files:

1. Use a macro definition on the MMS command line

2. Use a DCL symbol as a macro

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

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

Thus, the .OBJ.OLB rule would look like this:

```
.OBJ.OLB :
     IF "''F$SEARCH("$@")'" .EQS. "" THEN $(LIBR)/CREATE $@
     $(LIBR) $(LIBRFLAGS) $@ $<
     IF "$(CLEAN)" .NES "" THEN DELETE $<;
```

The command line would be:

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

The string you define the macro to be (that is, where the second CLEAN appears in the command line) can be anything you like. All MMS needs is something to expand the $(CLEAN) macro to so that it will not be null.

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

```
.OBJ.OLB :
     IF "''F$SEARCH("$@")'" .EQS. "" THEN $(LIBR)/CREATE $@
     $(LIBR) $(LIBRFLAGS) $@ $<
     IF "''CLEAN'" .NES "" THEN DELETE $<;
```

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

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

2.7 DOING PARALLEL PROCESSING

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

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

TARG1 :
        MMS/CMS/OUT=TARG1.OUT FOO.EXE
        PROCESS TARG1.OUT   ! Creates TARG1.COM
        SUBMIT $*

TARG2 :
        MMS/CMS/OUT=TARG2.OUT BAR.EXE
        PROCESS TARG2.OUT   ! Creates TARG2.COM
        SUBMIT $*
    .
    .
    .

! The rules building the parts of your system
FOO.EXE : FOO.OBJ
        ACTION

BAR.EXE : BAR.OBJ
        ACTION
    .
    .
    .
```

PROCESS (in the TARG1 and TARG2 action lines) is a program that you supply to insert a dollar sign ($) in front of each line in the file, thus creating the command procedures.

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

CHAPTER 3

GENERAL USAGE

This chapter answers some commonly-asked questions about the workings and general usage of DEC/MMS.

3.1 HOW CAN USING MMS HELP A PROJECT?

A software project on which more than one programmer is working usually has one CMS library or project source directory containing the pieces of the system. In addition, each programmer usually has his/her own copies of everything that is in the CMS or source library. Each programmer makes different changes to the files and puts them back in the library. This procedure can result in the following problems:

1. Forgetting which modules have changed.

2. Conflicting changes to the same module(s), resulting in modules that do not work.

3. Wasted disk space.

MMS can help solve these problems in the following ways:

1. MMS determines what has changed, relieving programmers of that responsibility.

2. MMS can be used to build and test modules first locally, and then against the modules in the source library. Thus, the modules that are replaced in the library will always work.

3. Use of the /SKIP qualifier avoids unnecessary building of intermediate files, thus saving space.

3.2 WHY NOT JUST USE DCL COMMAND PROCEDURES?

A function of MMS that is not easily found in DCL is that MMS supplies a conditional for determining whether one file is older than another (or whether it exists at all with relation to another file). For example, the following relationship:

```
A : B
    ACTION

     or

A : B ; ACTION
```

means, in DCL terms:

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

DCL does not easily supply this capability. You can use DCL command procedures to perform functions similar to those performed by MMS, but using them is not always the most efficient way of doing things.

MMS has some other advantages over DCL command procedures:

1. The order in which MMS processes dependency rules is determined by logic, rather than by the position of the rules in the description file. In DCL command procedures, the order of command execution is determined by the position of the command in the procedure.

2. You can leave some steps out of an MMS description file, due to the presence of built-in rules, which define certain commonly used actions. You cannot leave any steps out of a DCL command procedure.

3. MMS allows the "top-down" breakdown of a task. This means you can use target names at the beginning of a description file to specify the order in which tasks must be accomplished, and then specify the actions to accomplish those tasks further down in the description file. For example:

```
! Target names for the tasks
BLD_MY_SYSTEM : NOTIFY_SYS_MGR  FOO.EXE -
                INSTALL_THE_EXE  CLEANUP

! Actions to accomplish the tasks
NOTIFY_SYS_MGR :
        ! Action to send mail to system manager
```

```
CLEANUP :
        ! Actions to purge, delete, and so on

INSTALL_THE_EXE :
        ! Action (such as VMSINSTAL) to
        ! install the image

FOO.EXE : A.OLB($(MY_MODS))
        ! Actions to create FOO.EXE
```

In DCL, you cannot break a task down into sub-tasks in as clean a manner. You must specify the actions as you go along, thus scattering the commands and structural information throughout the command procedure. With MMS, the structural information is all in one place, giving a clearer representation of the system.

3.3 HOW DOES DEC/MMS EXECUTE COMMANDS?

When you run MMS, two processes are used. The first process, your current process, executes the actual MMS code. The second process, a spawned subprocess, executes the commands specified on action lines in the description file.

The spawned subprocess is created by MMS only when an action is to be executed. MMS creates only one subprocess to execute all actions in the description file. MMS creates this subprocess when it executes the first action; the subprocess remains until the parent MMS process terminates.

While the subprocess is executing an action (such as a DCL command), the parent process is in a hibernate state. Therefore, if you monitor the parent process, do not be alarmed to find it hibernating.

When you invoke MMS from a description file using the MMS or $(MMS) commands, another subprocess is spawned to execute the new MMS. The original subprocess is treated as a parent process for the subsequent MMS execution.

3.4 CAN THE ORDER OF COMMAND EXECUTION BE CONTROLLED?

Yes, to an extent. MMS executes commands as a result of traversing a network data structure or graph. MMS initially builds this graph from parsing the description file. The order of the dependencies in the description file may determine the position of a target (or "node") in the graph.

NOTE

This order will hold true when all the dependencies are explicitly specified in the description file. However, the order of command execution may be affected if part of the description file relies on built-in or user-defined rules to specify implied target/source relationships. The suffixes precedence list also becomes a factor in the order of command execution when built-in rules are used.

For example, suppose you have the following description file:

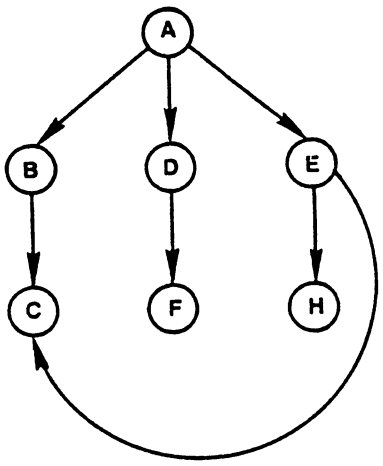
```
A : B  D  E
   ACTION 1

B : C
   ACTION 2

D : F
   ACTION 3

E : H  C
   ACTION 4
```

When MMS builds a graph, it places nodes from left to right, in the order in which it encounters them in the dependency rules. Therefore, the graph for this description file would look like the following:



ZK-1208-82

MMS walks the graph in a "left first, depth first" order. That is, the nodes would be visited in the following order: A, B, C, D, F, E, H, C.

If MMS executed all the actions in this description file, it would do so in the following order: ACTION 2, ACTION 3, ACTION 4, ACTION 1.

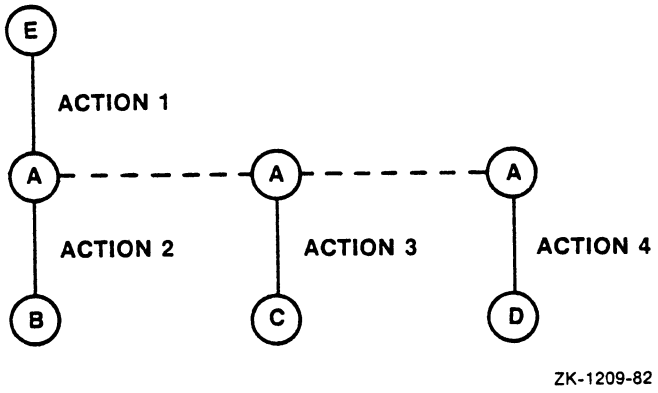
Graphs for double colon rules are different from those for regular rules. For example, suppose you have the following description file:

```
E : A
    ACTION 1
A :: B
    ACTION 2

A :: C
    ACTION 3

A :: D
    ACTION 4
```

The graph for this description file would look like the following:



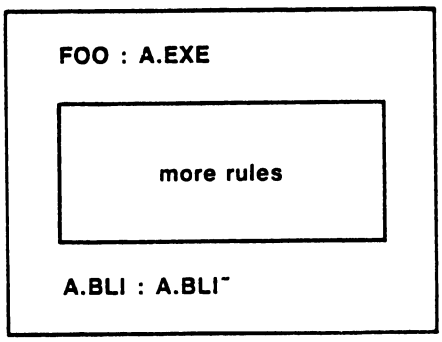
The node A is really represented in the graph as one node. However, internally, two "clones" of A are created for MMS to act upon. Each clone is executed at a different time, but MMS gives the clones the same time that the original node A had. MMS does this so that the execution of ACTION 2 does not make the second clone appear up-to-date with regard to node C.

The actions are executed in the order in which they appear in the graph, that is, left first, depth first. (Each of node A's clones is treated the same, that is, left to right.)

The order of command execution may sometimes be difficult to predict because it can be affected by the following factors:

- The order of file types in the suffixes precedence list. This factor only affects dependencies that rely on built-in or user-defined rules.

- Rules that appear between directly-related dependencies in the description file. For example, suppose your description file looks like the following:



ZK-1207-82

A.EXE depends on A.BLI, but these rules are not together in the description file. This positioning does not affect the relationship between A.EXE and A.BLI, but it can affect the construction of the graph. The nodes may not be initially connected in the graph. Two nodes may not be obviously connected when the graph is constructed, but they will be connected when built-in or user-defined rules are executed.

CHAPTER 4

LIMITATIONS OF V1.0

This chapter documents limitations of Version 1.0 of DEC/MMS.

4.1 TARGET AND SOURCE SPECIFICATIONS

For this release of MMS, targets and sources in dependency rules must be in the same directory for built-in rules to work properly. That is, you cannot have a rule with the following target/source line:

```
FOO:BAR.OBJ : BAS:BAR.C
```

MMS will not be able to use built-in rules to build the target in such a rule. (If you explicitly specify an action line, however, MMS will work properly.) You can get around this limitation by adding an action line copying the files to the correct directory.

```
FOO:BAR.OBJ : BAS:BAR.C
        COPY BAS:BAR.C FOO:
```

MMS will then be able to build BAR.OBJ from BAR.C using built-in rules.

This limitation does not apply to elements in CMS libraries.

4.2 MULTIPLE TARGETS WITH ONE ACTION

There is a potential problem if you have more than one target on one target/source line that have the same action: MMS may create more than one version of the targets because it executes the action line once for each target.

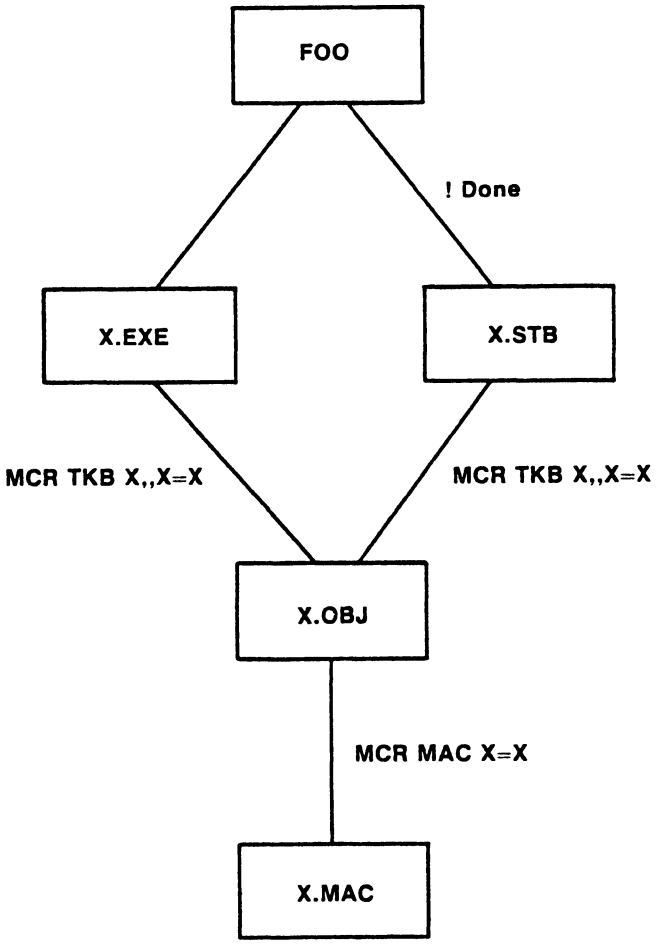
Suppose, for example, that you have the following description file:

```
FOO : X.EXE, X.STB
        ! Done

X.EXE, X.STB : X.OBJ
        MCR TKB X,,X=X

X.OBJ : X.MAC
        MCR MAC X=X
```

The graph that MMS builds from this description file looks like the following:



If FOO does not exist, MMS will create it by walking the graph left first, depth first. As it updates the nodes, MMS flags them as being updated and does not check them again, in order to avoid repeating actions.

Assuming that none of the nodes except X.MAC exists, MMS will create FOO by following these steps:

1. Execute the MCR MAC X=X action line to create X.OBJ from X.MAC.

2. Execute the MCR TKB X,,X=X action line to create X.EXE from X.OBJ. (This action also creates X.STB.)

3. Execute the MCR TKB X,,X=X action line to create X.STB from X.OBJ. (This action also creates X.EXE.)

4. Execute the !Done action line to create FOO.

As a result of steps 2 and 3, MMS creates two versions each of X.EXE and X.STB.

Now suppose that X.OBJ exists and is up-to-date. MMS will then perform the following steps:

1. Execute MCR TKB X,,X=X to create X.EXE (and also X.STB).

2. Check X.STB. It is found to be up-to-date because the MCR TKB action line was just executed. Therefore, MMS does not execute the action again.

3. Execute !Done to create FOO.

This time, MMS creates only one version of X.EXE and X.STB.

MMS does not realize that executing action lines to update one target can also update other targets. When MMS flags a node as being updated, it assumes that all nodes above that one must then be updated. It must assume this in order to avoid duplicating actions unnecessarily. For example, suppose that the action executed to update X.EXE also updates X.MAC. X.MAC is then newer than X.OBJ. Therefore, MMS updates X.OBJ, making it newer than X.EXE. Because MMS has already flagged X.EXE as being up-to-date, however, it does not try to update everything else again.

To avoid unnecessary duplication of actions, do not put more than one target with the same action line on the same target/source line. Separate them into different dependency rules.

4.3 DEFAULT FILE TYPES FOR /DESCRIPTION

MMS does not supply any default file types for the /DESCRIPTION qualifier. If you specify /DESCRIPTION=FOO and expect MMS to look for FOO.MMS, you will get an error. You must explicitly specify the file type (usually .MMS) when you use the /DESCRIPTION qualifier.

DECUS

DECUS SUBSCRIPTION SERVICE
DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
249 NORTHBORO ROAD, BPO2
MARLBORO, MASSACHUSETTS 01752

MOVING OR REPLACING A DELEGATE?

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: _____

Name: _____

Company: _____

Address: _____

_____

State/Country: _____

Zip/Postal Code: _____

Mail to: **DECUS - ATTN: Subscription Service**
249 Northboro Road, BPO2
Marlboro, Massachusetts 01752 USA