

FILEID**VMSINSTAL

L 9

++
Facility: VMSINSTAL, The Software Product Installation Procedure
Abstract: This procedure is used to install software products on a VAX/VMS system. It can install VMS updates and upgrades, and also layered products. Multiple products can be installed with one invocation.
Environment: The document "Developer's Guide to the VAX/VMS VMSINSTAL Procedure" describes this procedure in detail. Product kits are created with the SPKITBLD procedure or an equivalent.
Parameters: P1 List of products to be processed.
(all optional) P2 Device for distribution volumes.
P3 OPTIONS
P4 List of single-letter options.
A - create or use auto-answer file
B - invoked during system boot for recovery
C - produce callback trace
D - VMSINSTAL debug mode
G - get product savesets and save on disk
K - product kit debug mode
L - log file activity
Q - quality assurance mode
(turns on the TELL_QA callback,
and inhibits asking if the system
disk backup was made).
R - install to alternate system root
S - produce statistics report
P5 Device/directory if G option, or alternate system root
if R option.
P6 Optional qualifiers to first backup command if G option.

Author: Paul C. Anagnostopoulos
Created: 11 June 1982

Modifications:

- V04-001 BLS0350 Benn Schreiber 3-SEP-1984
Final tailoring corrections. Cleanup pass on symbol
definitions and upcase logical names in lexicals. Add
ability to specify new file which goes on library disk.
If using common system disk, allow ability to specify new
file which goes on system specific directory. Fix marker file
manipulation to account for 39 character product name.
Save additional state in marker file.
- V03-117 HWS0105 Harold Schultz 29-Aug-1984
When searching for logical names which are to be deassigned
if found, specify the table in the search in order to know
which table to deassign from if the name is found.
- V03-116 HWS0102 Harold Schultz 10-Aug-1984
Fix premature exit if error in directory specification.
- V03-115 HWS0101 Harold Schultz 08-Aug-1984
Change minimum BYTLM requirement from 20480 to 18000.
- V03-114 HWS0086 Harold Schultz 19-Jul-1984
1. Add GENERATE SDL DEFINITIONS callback to allow

2. generation of SDL require files.
3. Fix unbalanced ')' and arguments in fao function for version output.
4. Increase minimum quota requirements.
5. Add explanatory text to product input on how to exit.
6. Add /ABSOLUTE option to PATCH_IMAGE.
7. Give statistics demon BYPASS privilege.
8. Don't remove '' from device names.
9. Change maximum length for product name from 6 to 36 (from 9 to 39 if vvu included)
10. Add explanatory text when asking for next volume in RESTORE_SAVESET.

V03-113 BLS0324 Benn Schreiber 12-JUN-1984
 Correct creation of system directories.

V03-112 BLS0323 Benn Schreiber 31-MAY-1984
 More of 111. Also correct f\$file usage.

V03-111 BLS0322 Benn Schreiber 30-MAY-1984
 More tailoring.

V03-110 BLS0314 Benn Schreiber 7-MAY-1984
 Minor corrections for tailoring.

V03-109 BLS0293 Benn Schreiber 30-MAR-1984
 Allow G option to foreign media. Couple small bug fixes. Support gt 26 savesets. Add support for rights data base to create_account callback.

V03-108 BLS0286 Benn Schreiber 11-MAR-1984
 Enhance restore saveset to move to next piece of media if saveset not found on current one.

V03-107 WHM0001 Bill Matthews 28-Feb-1984
 Add support for installing to a common system root.

V03-106 BLS0271 Benn Schreiber 15-FEB-1984
 Fix problems in statistics demon. Convert f\$log to f\$Strnlm. Changes for V4 tailoring. In Q/A mode, don't ask questions re other users on the system and backup done ok. Correct f\$parsr usage when parsing saveset location. Fix shutdown problems with statistics demon. Add file size to demon directory listing.

V03-105 BLS0259 Benn Schreiber 10-Jan-1983
 Set directory protection before deleting. Fix quotes in SUMSLP_TEXT.

V03-104 MCN0139 Maria del C. Nasr 16-Nov-1983
 Fix bug with F\$PARSE statement in CREATE_ACCOUNT.

V03-103 MCN0003 Maria del C. Nasr 20-Sep-1983
 If the console is used to install the kit, VMSINSTAL loops in the cleanup procedure because it is trying to read a file that has been closed. The fix for this problem is to close the files after the REMOUNT_CONSOLE procedure is executed.

V03-102 MCN0002 Maria del C. Nasr 15-Sep-1983

When asking for next product to be installed, add EXIT
as a response to finish processing. ^Z was the only
being accepted, which is not obvious for the user.

V03-101 MCN0001 Maria del C. Nasr 15-Sep-1983
Take out special code for TDMS and ACMS since it is
not needed anymore.

V03-100 PCA1010 Paul C. Anagnostopoulos 21-Mar-1983
Major modifications and enhancements for VMS V4.

If VMIS\$INSTALLING is not equated, then we were invoked to do an
installation. Otherwise, this is a recursive callback.

if f\$type(vmi\$installing) .eqs. "" then goto STEP_1

Set up the CTRL/Y and error environment for the callbacks.
If we are not tracing them, then just go off and do one.

on control_y then goto CONTROL_Y

on error then exit \$status

if f\$trnlnm('VMIS\$CALL_FILE') .eqs. "" then goto 'p1

We are tracing callbacks, so format this callback nicely.

l = f\$fao("!24AS<!AS> <!AS> <!AS> <!AS> <!AS> <!AS> <!AS>",p1,p2,p3,p4,p5,p6,p7,p8)

c = false

Scb10:
 write vmi\$call file f\$fao("!* !#AS!#<"!>",c*3,78-c*24,l,f\$len(l) .gt. 78-c*24)
 l = f\$ext(78-c*24,999,l)
 c = true

if l .nes. "" then goto cb10

goto 'p1

On CTRL/Y we just bag the entire installation.

CONTROL_Y:
 vmi\$msg f ctrl Y "Installation cancelled via CTRL/Y."
 exit vmi\$_failure

OVERVIEW OF CALLBACKS

All callback handlers appear first in VMSINSTAL, in order to minimize label search time. They are grouped into three categories, as follows:

1. FIND_FILE and MOVE_FILE, because they represent over 50% of the callbacks requested.
2. Standard callbacks, in alphabetical order.
3. Internal callbacks, in alphabetical order.

If a callback requires a parameter which is a logical name or symbol to be set, that parameter should be the first one.

Callback parameters which specify a file should require a complete file spec, except when the file can ONLY be in the kit's working directory. Wildcarding should be not allowed unless absolutely necessary, since the files can't be FINDFILEd first.

Most callbacks that manipulate files accept an options parameter. This is a list of single-character codes, which must all be unique. Other callbacks which accept an options parameter may use any codes.

- E set the specified ECOs when providing an image.
- I add the provided sharable image to IMAGELIB.
- J produce a Journal file when patching an image.
- K Keep old versions of the file; do not purge.
- L If tailored system, put file on library disk.
- O If common system disk, use system specific root.
- R Reinstall an image before purging it.

File manipulation callbacks must be careful to allow the same file to be manipulated more than once.

Callbacks which set various options should be of the form "SET option ..." (two parameters), to be consistent with the DCL SET command.

D 10

```
$! FIND_FILE logical file-spec [default-spec] locate-list [symbol]
      W - kit work directory
      O - system specific directory
      S - system directories
      E - error msg if not found

$! FIND_FILE:
      f = f$parse(p3,f$parse(p4,"SYNTAX_ONLY"),'MISSING.MIS')
      if f .eqs. "", then goto ff90
      if p6 .nes. "" then 'p6 == ""

      if f$loc('W',p5) .eq. f$len(p5) then goto ff10
      m = 'New file'
      w = f$parse("vmi$kwds:",f,,,"SYNTAX ONLY")
      if f$search(w) .eqs. "" then goto ff10
      if p6 .nes. "" then 'p6 == 'W'
      define 'p2' 'w
      exit vmi$_success

$ff10: if .not. vmi$common_root .or. (f$loc('O',p5) .eq. f$len(p5)) then goto ff20
      l = f$parse('vmi$specific:',f)
      if l .eqs. "" then goto ff20
      if f$search(l) .eqs. "" then goto ff20
      if p6 .nes. "" then 'p6 == 'S
      define 'p2' 'l
      exit vmi$_success

$ff20: if f$loc('S',p5) .eq. f$len(p5) then goto ff30
      m = 'File'
      if f$search(f) .eqs. "" then goto ff25
      if p6 .nes. "" then 'p6 == 'S
      define 'p2' 'f
      exit vmi$_success

$ff25: l = f$parse('lib$sysroot:',f)
      if l .eqs. "" then goto ff30
      if f$search(l) .eqs. "" then goto ff30
      if p6 .nes. "" then 'p6 == 'S
      define 'p2' 'l
      exit vmi$_success

$ff30: if f$loc('E',p5) .eq. f$len(p5) then exit vmi$_success
      vmi$msg w nofile ""m "'p3 does not exist."
      if p6 .nes. "" then 'p6 == 'E'
      exit vmi$_failure

$ff90: vmi$msg e badspec "file spec ''p3 cannot be parsed'' -
          "using default spec ""'p4'''."
      exit vmi$_failure
```

```

$! MOVE_FILE logical full-spec [options]
$!           I - add sharable image to IMAGELIB
$!           K - keep old versions
$!           L - put file on library disk if tailored
$!           R - reinstall image before purging
$!           O - system specific directory

$! SMOVE_FILE:
$!   l = f$loc('L',p4) .ne. f$len(p4), and, (vmi$tailoring .eq. 2)
$!   o = vmi$common_root .and. (f$loc('O',p4) .ne. f$len(p4))
$!   if l then t = f$parse('LIBSSYSROOT:',p3,,,'SYNTAX ONLY')
$!   if .not. (l.or. o) then t = f$parse(p3,,,'SYNTAX ONLY')
$!   if o then t = f$parse('VMISSPECIFIC:',p3,,,'SYNTAX ONLY')
$!   w = f$parse('vmi$kwds:',t,,,'SYNTAX ONLY')
$!   if vmi$safety then goto mf20

$!   if f$search(w) ,eqs. "" then exit vmi$_failure
$!   b = f$getdvi(w,'FULLDEVNAM') .eqs. f$getdvi(t,'FULLDEVNAM')
$!   if b then rename 'w' 't'
$!   if .not. b then backup 'w' 't/new_version/owner=original
$!   ww = f$search(w)
$!   if ww .nes. "" then set prot=s=rwed 'w*
$!   if ww .nes. "" then delete 'w'

$!   install = "$install"
$!   if f$loc('R',p4) .eq. f$len(p4) then goto mf10
$!   if f$file(f$elem(0,';',t),'KNOWN') then install 't/replace
$!mf10:
$!   if vmi$purge .and. f$loc('K',p4) .eq. f$len(p4) then purge 'f$elem(0,';',t)
$!   if f$loc('I',p4) .ne. f$len(p4) then vmi$callback UPDATE_LIBRARY -
$!           vmi$ vmi$root:[syslib]imagerlib.olb share "/replace" 't
$!   define 'p2' 't
$!   exit vmi$_success

$!mf20: write vmi$defer_file p1," ",p2," ",t," ",p4
$!   define 'p2' 'w
$!   exit vmi$_success

```

```

$! ASK symbol prompt [default] [options] [help]
      B - boolean value
      D - double space
      H - display help first
      I - integer value
      N - null answer is allowed
      R - ring bell
      S - string value
      Z - CTRL/Z is allowed

SASK:
w = """
if f$loc('R',p5) .ne. f$len(p5) then w[0,8]= %x07
if f$loc('D',p5) .ne. f$len(p5) .or. f$loc('R',p5) .ne. f$len(p5) then say w
dt = 'S'
if f$loc('B',p5) .ne. f$len(p5) then dt = 'B'
if f$loc('I',p5) .ne. f$len(p5) then dt = 'I'
if p4_.nes. then p3 = p3 + "[" + p4 + "]"
p3 = "*" + p3 + f$ext(f$loc(dt,'BIS'),1,'?::') + " "
if f$loc('H',p5) .ne. f$len(p5) then goto a80

Sa10: if f$trnlm('VMISAUTOFILE') .nes. "" then -
      if vmi$auto_option .eqs. 'R' then goto a15
      read/end_of_file=a70/prompt=""'p3"" vmi$terminal_file v
      v = f$edit(v,'COMPRESS,TRIM,UNCOMMENT,UPCASE')
      goto a20
Sa15: v = "(END-OF-FILE)"
      read/end_of_file=a90 vmi$auto_file v
      if f$elem(0,"\",v) .nes. p3 then goto a90
      v = f$elem(1,"\",v)
      say p3,v

Sa20: if f$trnlm('VMISAUTOFILE') .nes. "" then -
      if vmi$auto_option .eqs. 'W' then write vmi$auto_file p3,"\",v
      if v .eqs. "?" then goto a80
      if v .eqs. "" then v = f$edit(p4,'COMPRESS,TRIM,UPCASE')
      if v .eqs. "" .and. f$loc('N',p5) .eq. f$len(p5) then goto a10
      goto a_dt

Sa_B: 'p2 == f$ext(0,1,v) .eqs. "Y"
      if f$loc(v,'YES') .eq. 0 .or. f$loc(v,'NO') .eq. 0 then exit vmi$_success
      vmi$msg e yesno "Please enter YES or NO."
      goto a10

Sa_I: 'p2 == f$int(v)
      if f$type(v) .eqs. "INTEGER" then exit vmi$_success
      vmi$msg e integer "Please enter an integer number."
      goto a10

Sa_S: 'p2 == v
      exit vmi$_success

Sa70: if f$loc('Z',p5) .eq. f$len(p5) then goto a10
      'p2 == "Z"
      exit vmi$_success

Sa80: if p6_.eqs. "" then p6 = "Sorry, no help is available."
      say ""
      if f$ext(0,1,p6) .nes. 'a' then say p6
      if f$ext(0,1,p6) .eqs. 'a' then 'p6
      goto a10

Sa90: vmi$msg f autosync "Auto-answer file is not in synch with questions." -

```

G 10

"question: \"p3\" -
"file: \"f\$elem(0,\"\",v)\"

```
$!      CHECK_NET_UTILIZATION symbol blocks
$S
$CHECK_NET_UTILIZATION:
$S      'p2 == p3 .le. vmi$free_blocks
$S      exit vmi$success
```

```
$! CHECK_VMS_VERSION symbol [version] [baselevel]
$ This callback is obsolete as of VMS V4. However, it must remain
$ here for compatibility with V3.
$CHECK_VMS_VERSION:
$p2 == false
$v := f$ext(1,999,f$getsyi("VERSION"))
if f$ext(0,1,f$getsyi("VERSION")) .nes. "V" then goto cvv20
  i = f$loc("",v)
  v = f$fao("!2ZL!AS",f$int(f$ext(0,i,v)),f$ext(i+1,1,v))
  if p3 .nes. "" .and. v .ges. p3 then 'p2 == true
  exit vmi$_success
$cvv20: if f$len(p4),,eq. 3 then p4 = p4 + "-" + p4
$  if p4 .eqs. ." .or. -
$    v .ges. f$ext(0,3,p4) .and. v .les. f$ext(4,3,p4) then 'p2 == true
$  exit vmi$_success
```

```
$!      CREATE_ACCOUNT username qualifiers
$!      UPDATE_ACCOUNT username qualifiers
$!
$CREATE_ACCOUNT:
$      m = "creates"
$      v = "ADD"
$      goto ca10
$UPDATE_ACCOUNT:
$      m = "updates"
$      v = "MODIFY"
$
$ca10: vmi$msg i account "This installation ''m an account named ''p2."
$      vmi$find sysuaf sysuaf vmi$root:[sysexe].dat s,e
$      if .not. $status then exit $status
$      authorize = "$authorize"
$      define/user netuaf 'f$parse("netuaf",f$trnlnm("SYSUAF"),,,,"SYNTAX_ONLY")
$!**  define/user rightslist 'f$parse("rightslist",f$trnlnm("SYSUAF"),,,,"SYNTAX_ONLY")
$      authorize 'v 'p2 'p3
$      deassign sysuaf
$      exit vmi$_success
```

```
$!      CREATE_DIRECTORY SYSTEM hierarchy [qualifiers]
$!          USER    directory
$!
$CREATE_DIRECTORY:
$    goto cd_p2
$!
$cd_SYSTEM:
$    vmi$msg i sysdir "This product creates system directory [''p3]."
$    create/directory vmi$root:[p3] 'p4
$    if .not. vmi$common_root then exit vmi$_success
$    create/directory vmi$specific:[p3] 'p4
$    type sys$input
```

If you intend to execute this layered product on other nodes in your VAXcluster, and you have the appropriate software license, you must prepare the system-specific roots on the other nodes by issuing the following command on each node (using a suitably privileged account):

```
$    write sys$output "      $ CREATE / DIRECTORY SYS$SPECIFIC:[''p3'] ''p4''
$    exit vmi$_success
$!
$cd_USER:
$    if f$getdvi(p3,"FULLDEVNAM") .eqs. f$getdvi("VMI$ROOT","FULLDEVNAM") then -
$        vmi$msg i sysdisk "This product creates system disk directory ''p3.''
$    create/directory 'p3 'p4
$    exit vmi$_success
```

```
$!      DELETE_FILE full-spec  
$  
$DELETE_FILE:  
$  vmi$find vmi$del 'p2 "' s,e  
$  if .not. $status then exit $status  
$  if vmi$safety then goto df20  
$  install = '$install  
$  if f$file(f$elem(0,".",f$trnlnm("VMISDEL")),'KNOWN') then install vmi$del/delete  
$  delete 'f$trnlnm("VMISDEL")'  
$  exit vmi$_success  
$  
$df20: write vmi$defer_file p1," ",f$trnlnm("VMISDEL")  
$  exit vmi$_success
```

\$! GENERATE SDL_DEFINITIONS module-file language optquals options

This callback handles SDL generation of definitions
 (Currently, there are no defined options for this callback.
 Any options supplied will signaled with an informational
 message and ignored.)

\$ First check for the existance of SYSSYSTEM:SDLNPARSE.EXE and
 VMISROOT:[SYSEXEC]STARLETSD.TLB before proceeding further
 \$ Also check for existance of options. If any found, signal options
 ignored message.

\$! GENERATE SDL_DEFINITIONS:

```
vmi$find SYSSYSTEM:SDLNPARSE.EXE "" s,e
if .not. $status then exit $status
vmi$find vmi$sdl VMISROOT:[SYSLIB]STARLETSD.TLB "" s,e
if .not. $status then exit $status
```

\$! Form output file specs from requested languages.

```
lns = ""
y = 0
$sdla:
x=f$elem(y,"",p3)
if x .eqs. "" then goto sdlb
lns = lns + "," + x + "=vmi$kwds:"
y = y + 1
goto sdla
$sdlb: lns = lns + ")" - ","
```

\$! Check if single module name or file containing module names.

```
set noon
multi = f$loc('"a',p2) .ne. f$len(p2)
if .not. multi then goto sdl2
```

\$! Have a file which contains the name of the modules to extract from the
 library. Process each name completely prior to getting the next name.

1. read module name from file
2. extract module from .TLB library
3. run SDL on the extracted module
4. delete extracted library module

```
p2 = p2 - "a"
found = 0
vmi$find vmi$list 'p2 vmi$kwds:.dat w,e
if .not. $status then goto sdl5
open/read vmi$list_file vmi$list
$sd1: read/end=sdl4 vmi$list_file p2
$sd2: library/extract = 'p2/output = vmi$kwds:'p2.sdi vmi$sdl
if $status then goto sdl3
vmi$msg i libfail "Failed to extract ''p2 from library."
if multi then goto sdl1
goto sdl5
$sd3: SDL/NOPARSE/NOHEADER/LANG='lns'"p4 vmi$kwds:'p2.sdi
if .not. $status then goto sdl5
found = 1
delete/nolog vmi$kwds:'p2.sdi;*
if multi then goto sdl1
set on
```

```
$          exit vmi$_success
$sd14: close vmi$list_file
$      set on
$      if .not. found then exit vmi$_failure
$      exit vmi$_success
$sd15: set on
$      exit $status
```

```
$!      GET_SYSTEM_PARAMETER symbol name
$!
$GET_SYSTEM_PARAMETER:
$p2==f$getsyi(p3)
$    exit vmi$_success
```

\$! MESSAGE [VMSINSTAL] severity id text ...
\$! if P2 is VMSINSTAL, this is a special internal call. All
\$! parameters bumped by 1.
\$! Text is strings enclosed in quotes. If text doesn't begin with
\$! "%" then "%" or "-" is prefixed, depending if first or later
\$! messages.

\$MESSAGE:
\$ if p2 .nes. "VMSINSTAL" then goto m5
\$ b = "%VMSINSTAL-" + p3 + "-" + p4 + ", "
\$ i = 5
\$ goto m10
\$m5: b = "%" + f\$ext(0,f\$len(vmi\$product)-3,vmi\$product) + "-" + p2 + "-" + p3 + ", "
\$ i = 4
\$
\$m10: if t\$ext(0,1,p'i) .nes. "%" then say b,p'i
\$ if f\$ext(0,1,p'i) .eqs. "%" then say p'i
\$ b[0,1]:= "-"
\$ i = i + 1
\$ if p'i .nes. "" then goto m10
\$
\$ if p2 .eqs. "VMSINSTAL" .and. p3 .eqs. "F" then exit %x10f50004
\$ exit vmi\$_success

\$! PATCH_IMAGE logical patch-name-type [image-spec] [options]
\$! D 11
\$! J - journal
\$! K - keep old versions
\$! A - PATCH/ABSOLUTE

SPATCH_IMAGE:
S vmi\$find vmi\$com 'p3 vmi\$kw: w,e
S if .not. \$status then exit \$status
S if p4 .nes. "" then goto pai15
S open/read vmi\$temp_file vmi\$com
S read vmi\$temp_file_l
S close vmi\$temp_file
S l = f\$edit(l,"T","COMPRESS,TRIM,UPCASE")
S p4 = f\$elem(0," ",l)
S p5 = p5 + "" + f\$elem(1," ",l)
Spai15: vmi\$find vmi\$exe 'p4,"",w,s,e vmi\$exe
S if .not. \$status then exit \$status
S d = f\$parse("vmi\$exe","","DEVICE","SYNTAX_ONLY") + f\$parse("vmi\$exe","","DIRECTORY","SYNTAX_ONLY")
S n = f\$parse("vmi\$exe","","NAME","SYNTAX_ONLY")
S
S vmi\$abs = ""
S if f\$loc("A",p5) .ne. f\$len(p5) then vmi\$abs = "/ABSOLUTE"
S define vmi\$jnl nl:
S vmi\$jnl == "N"
S if f\$loc("J",p5) .eq. f\$len(p5) then goto pai30
S vmi\$find vmi\$jnl ,jni 'p4 w,s vmi\$jnl
S if vmi\$jnl .eqs. "" then define vmi\$jnl vmi\$kw:
S if vmi\$jnl .eqs. "" then goto pai30
S vmi\$callback UPDATE_FILE vmi\$jnl vmi\$jnl
S if .not. \$status then exit \$status
S
Spai30: define/user sys\$input vmi\$com:
S vmi\$no_output
S patch/output=vmi\$kw:/journal=vmi\$jnl'vmi\$abs' vmi\$exe
S if vmi\$exe .eqs. "S" then vmi\$callback PROVIDE_IMAGE 'p2 'n.exe 'd 'p5
S if vmi\$exe .eqs. "S" then if .not. \$status then exit \$status
S if vmi\$jnl .eqs. "" then vmi\$callback PROVIDE_FILE vmi\$jnl 'n.jnl 'd
S if vmi\$jnl .eqs. "" then exit \$status
S exit vmi\$_success

```
$!      PRINT_FILE full-spec [copies]
$SPRINT_FILE:
$      vmi$find vmi$ppt 'p2 """ w,s,e
$      if .not. $status then exit $status
$      if p3 .eqs. """ then p3 = "1"
$      set noon
$      print/name='vmi$product/copies='p3 vmi$ppt
$      if .not. $status then vmi$msg w noprnt "File ''p2 cannot be printed."
$      set on
$      exit vmi$_success
```

\$!
\$
\$UF
\$S
\$S
\$S
\$S
\$S

\$! PRODUCT procedure:callback parameter ...

This callback allows product groups to have their own special callbacks which do things useful for their product installations. The base product (e.g., CDD, RSX/VAX) provides a procedure which handles the callbacks, and then other products can use the callbacks via this special VMSINSTAL callback.

\$! PRODUCT:

```
$! vmi$find vmi$com 'f$elem(0,"",p2) vmi$root:[sysupd].com s vmi$com  
if vmi$com .nes. "S" then vmi$msg e nopsproc -  
  "Product-specific callback procedure ''f$elem(0,"",p2) does not exist."  
if vmi$com .nes. "S" then exit vmi$failure  
@vmi$com 'f$elem(1,"",p2) ""'p3""'p4""'p5""'p6""'p7""'p8""  
exit $status
```

G 11

```
$!      PROVIDE_DCL_COMMAND name-type
$SPROVIDE_DCL_COMMAND:
$      vmi$cld 'p2 vmi$kwds: w,e
$      if .not. $status then exit $status
$      vmi$find vmi$exe vmi$root:[syslib]dcltables.exe "" w,s,e vmi$exe
$      if .not. $status then exit $status
$      set command/tables=vmi$exe/output=vmi$kwds: vmi$cld
$      if vmi$exe .eqs. "S" then vmi$callback PROVIDE IMAGE vmi$exe dcltables.exe -
$          'fSparse("vmi$exe","","DEVICE","SYNTAX_ONLY")' 'fSparse("vmi$exe","","DIRECTORY","SYNTAX_ONLY")'
$      if vmi$exe .eqs. "S" then if .not. $status then exit $status
$      set command vmi$cld
$      exit vmi$_success
```

```
$! PROVIDE_DCL_HELP name-type
$ SPROVIDE_DCL_HELP:
$   vmi$callback UPDATE_LIBRARY vmi$ vmi$root:[syshlp]helplib.hlb help "/replace" vmi$kwd:'p2
$   exit $status
```

I 11

```
$! PROVIDE_FILE logical name-type directory [options]
      K - keep old versions
      L - Put on lib disk if tailored
      O - Put in system specific root
$! PROVIDE_FILE:
$!   vmi$find vmi$p3 vmi$kwd: w,e
$!   if .not. $status then exit $status
$!   o =
$!   if vmi$common_root .and. (f$loc("O",p5) .ne. f$len(p5)) then o = "O"
$!   vmi$find vmi$old 'p3' 'p4' s'o vmi$old
$!   if vmi$old .eqs. ."." then vmi$callback MOVE_FILE 'p2' 'p4''p3 k'p5''o'
$!   if vmi$old .eqs. "S" then vmi$callback MOVE_FILE 'p2' vmi$old 'p5''o'
$!   exit $status
```

```
J 11
$! PROVIDE_IMAGE logical name-type directory [options] [eco-list]
$!           E - set specified ECOs
$!           I - put in IMAGELIB
$!           K - keep old versions
$!           L - Put on lib disk if tailored
$!           O - Put in system specific root
$!
$! PROVIDE_IMAGE:
$!   vmi$find vmi$new 'p3 vmi$kw: w,e
$!   if .not. $status then exit $status
$!
$!   o = ""
$!   if vmi$common_root .and. (f$loc('O',p5) .ne. f$len(p5)) then o = "O"
$!   if f$loc('E',p5) .eq. f$len(p5) then goto pi29
$!   open/write vmi$temp_file vmi$kw:vmi.tmp
$!pi22:    write vmi$temp_file "SET ECO ",f$elem(0,'',p6)
$!            write vmi$temp_file "UPDATE"
$!            p6 = f$ext(f$loc(' ',p6)+1,999,p6)
$!            if p6 .nes. "" then goto pi22
$!   close vmi$temp_file
$!   define/user sys$input vmi$kw:vmi.tmp
$!   vmi$no_output
$!   patch/output=vmi$kw:/journal=nl: vmi$new
$!   purge 'f$elem(0,'',f$strnlnm("VMISNEW"))
$!   set prot=s=rwed vmi$kw:vmi.tmp;
$!   delete/nolog vmi$kw:vmi.tmp;
$!pi29:    vmi$find vmi$old 'p3 'p4 s'o vmi$old
$!           if vmi$old .eqs. "" then vmi$callback MOVE_FILE 'p2 'p4''p3 'p5,k'o
$!           if vmi$old .eqs. "S" then vmi$callback MOVE_FILE 'p2 vmi$old 'p5,r'o
$!           exit $status
```

```
$!      RENAME_FILE ddnt new-nt
$!
$RENAMFILE:
$    vmi$find vmi$ren 'p2 ""' s,e
$    if .not. $status then exit $status
$    if vmi$safety then goto rf20
$    rename 'f$trnlnm("VMISREN")*' 'p3
$    exit vmi$_success
$!
$rf20: write vmi$defer_file p1," ",f$trnlnm("VMISREN")," ",p3
$    exit vmi$_success
```

```

$! RESTORE_SAVESET letter [options]
$! N - saveset begins on next volume
$S
$SRESTORE_SAVESET:
$S set noon
$S if f$type(vmi$backup_openin) .nes. "" then goto rs5
$S vmi$no_output
$S vmi$no_error
$S backup 'vmi$kwrd:vmi$no_saveset/save vmi$kwrd:
$S vmi$backup_openin == $status
$S
$Srs5:
$S ssn = p2
$S if f$length(p2) .eq. 1 then goto rs7
$S if f$length(p2) .eq. 0 then goto rs95
$S v = f$integer(p2)
$S if v .le. 26 then goto rs100
$S ssn = f$fao("VMI_!4ZL",v)
$S
$Srs7:
$S vmi$msg i restore "Restoring product saveset ''p2...""
$S rewind = "/NOREWIND"
$S f = "VMISKWD:VMIBCKERR.TMP"
$S
$S If the distribution volume was mounted, and the caller specified
$S that this saveset begins on a new volume, then mount the volume.
$S We may also come back up here if we find that the required saveset
$S isn't on the current volume and we want to try the next.
$S
$Srs10:
$S if .not. vmi$remount .or. f$loc('N',p3) .eq. f$len(p3) then goto rs20
$S dismount/unload 'vmi$device'
$S say 'Please mount the next distribution volume on ''vmi$device''.'
$S vmi$callback PLACE_VOLUME "(If no more volumes, answer NO.)"
$S if .not. $status then goto rs90
$S mount/foreign/noassist/nowrite 'vmi$device'
$S
$S Try restoring the saveset. This may involve rewinding a tape in
$S case the savesets were out of order. If success or warning, we're
$S done. If any error other than BACKUP-?-OPENIN, that's a failure.
$S
$Srs20:
$S if f$search(f) .nes. "" then delete 'f';*
$S define /user sys$error 'f'
$S vmi$no_output
$S backup 'rewind 'vmi$place''vmi$product.'ssn/save set vmi$kwrd:
$S if $severity .or. $severity .eq. 0 then exit vmi$_success
$S if $status .ne. vmi$backup_openin then goto rs90
$S
$S If the saveset was remote or on tape, then we can't find it.
$S If it was on a disk that we mounted, try the next volume.
$S
$S if vmi$remote then goto rs90
$S if f$getdvi(vmi$place,'DEVCLASS') .eq. 2 then goto rs35
$S if .not. vmi$remount then goto rs90
$S goto rs10
$S
$Srs35:
$S if rewind,.egs. "/REWIND" .or. p2 .nes. "A" then goto rs90
$S rewind = "/REWIND"
$S vmi$msg i rewind "The tape will be rewound to try again."
$S goto rs20
$S
$S We come here in the event that the saveset cannot be found.
$S
$Srs90: vmi$msg e nosaveset "Saveset ''p2 cannot be restored."
$S w = f$search(f)
$S if w .nes. "" then type 'f'

```

```
$ if w .nes. "" then delete 'f';*
$ exit vmi$_failure
$rs95: vmi$msg e'nulsaveset "Null saveset name specified to RESTORE_SAVESET."
$ exit vmi$_failure
$rs100: vmi$msg e'illsaveset "Illegal saveset name ''p2 specified to RESTORE_SAVESET"
$ exit vmi$_failure
```

```
$! SECURE_FILE full-spec [owner-uic] [protection]
$$SECURE_FILE:
$  vmi$find vmi$ 'p2 """ w,s,e
$  if .not. $status then exit $status
$  if p3 .nes. """ then set file vmi$/owner='p3
$  if p4 .nes. """ then set protection=('p4) vmi$
$  exit vmi$_success
```

```

$!      SET IVP    {YES | NO | ASK} [options]
$!      PURGE   {YES | NO | ASK} [options]
$!                                H - display question help first
$!      REBOOT   {YES | NO}
$!      SAFETY   {YES peak : CONDITIONAL peak : NO}
$!      STARTUP name-type

$SET:
$      goto SET_`f$ext(0,3,p2)

$SET_IVP:
$      if p3 .eqs. "ASK" then vmi$callback ASK vmi$ivp "Do you want to run the IVP after the installation" -
$          yes b,'p4 ""'vmi$callback HELP_IVP"
$      if p3 .nes. "ASK" then vmi$ivp == p3 .eqs. "YES"
$      exit vmi$_success

$SET_PUR:
$      if p3 .eqs. "ASK" then vmi$callback ASK vmi$purge "Do you want to purge files replaced by this installation" -
$          yes b,'p4 ""'vmi$callback HELP_PURGE"
$      if p3 .nes. "ASK" then vmi$purge == p3 .eqs. "YES"
$      exit vmi$_success

$SET_REB:
$      vmi$reboot == p3 .eqs. "YES" .and. .not. vmi$alternate_root
$      exit vmi$_success

$SET_SAF:
$      goto ss_`p3

$ss_YES:
$      vmi$safety == true
$      if (p4 + p4/10) :le: vmi$free_blocks then goto ss10
$      vmi$msg e peakutil "This product requires `p4 blocks during installation."
$      exit vmi$_failure

$ss_CONDITIONAL:
$      vmi$safety == (p4 + p4/10) .le. vmi$free_blocks
$      goto ss10

$ss_NO:
$      vmi$safety == false

$!      Update the state in the marker file so we can tell which safety
$!      mode we're operating in.

$ss10: vmi$update_marker
$      read vmi$marker_file m
$      m[111,2]:= 'f$ext(vmi$safety,1,'US')      ! State S = safety, U = unsafe
$      write/update vmi$marker_file m
$      close vmi$marker_file
$      exit vmi$_success

$SET_STA:
$      vmi$startup == '@vmi$root:[sysmgr]' + p3
$      exit vmi$_success

```

```

$! SUMSLP_TEXT logical command-name-type -
$!   [file-spec      FILE old-checksum [new-checksum] [options]]
$!   lib-spec,member type                                K - keep old versions
$! SUMSLP_TEXT:
$ vmi$find vmi$com 'p3 vmi$kwd: w,e
$ if .not. $status then exit $status
$ if p4 .nes. "" then goto st15
$ open/read vmi$temp_file vmi$com
$ read vmi$temp_file l
$ close vmi$temp_file
$ l = f$edit(l-=:,"COMPRESS,TRIM,UPCASE")
$ p4 = f$elem(0,":")
$ p5 = f$elem(1,":")
$ p6 = f$elem(2,":")
$ p7 = f$elem(3,":")
$ p8 = f$elem(4,":")
sst15: if p5 .nes. "FILE" then goto st30
$ vmi$find vmi$txt 'p4 "' w,s,e vmi$txt
$ if .not. $status then exit $status
$ checksum vmi$txt
$ if checksum$checksum .ne. 'p6 then goto st90
$ d = f$parse("vmi$txt","","DEVICE","SYNTAX_ONLY") + f$parse("vmi$txt","","DIRECTORY","SYNTAX_ONLY")
$ n = f$parse("vmi$txt","","NAME","SYNTAX_ONLY") + f$parse("vmi$txt","","TYPE","SYNTAX_ONLY")
$ edit/sum/output=vmi$kwd:'n vmi$txt/update=vmi$com
$ if vmi$txt .eqs. 'S' then vmi$callback PROVIDE_FILE 'p2 'n 'd 'p8
$ if vmi$txt .eqs. 'S' then exit $status
$ exit vmi$_success
sst30: l = f$ext(0,f$loc("",,p4),p4)
$ vmi$find vmi$lib 'l "",s,e
$ if .not. $status then exit $status
$ m = p4 - l - ""
$ library/'p5/extract='m/output=vmi$kwd:'m.vmi vmi$lib
$ checksum vmi$kwd:'m.vmi
$ if checksum$checksum .ne. 'p6 then delete/nolog vmi$kwd:'m.vmi;*
$ if checksum$checksum .ne. 'p6 then goto st90
$ edit/sum/output=vmi$kwd:'m.vmi vmi$kwd:'m.vmi/update=vmi$com
$ vmi$callback UPDATE_LIBRARY 'p2 vmi$lib 'p5 "/replace" vmi$kwd:'m.vmi
$ s = $status
$ delete/nolog vmi$kwd:'m.vmi;*
$ exit s
sst90: if p7 .nes. "" then if checksum$checksum .ne. 'p7 then goto st92
$ vmi$msg i updated "File ''p4 is already updated."
$ exit vmi$_success
sst92: vmi$msg e tamper "File ''p4 has been tampered with."
$ exit vmi$_failure

```

```
$!      TELL_QA message
$!
$TELL_QA:
$  if vmi$ga then vmi$msg i qanote "NOTE TO QUALITY ASSURANCE PERSON:" ""'p2"""
$  exit vmi$_success
```

```
$! UPDATE_FILE logical full-spec
$S
$SUPDATEFILE:
$S   vmi$find 'p2 'p3 ""' w,s,e vmi$fil
$S   if .not. $status then exit $status
$S   if .not. vmi$safety .or. vmi$fil .eqs. "W" then exit vmi$_success
$S   backup ;p2 vmi$kwds:*.*/;owner=original
$S   vmi$callback MOVEFILE 'p2 'p2
$S   exit $status
```

```
$!      UPDATE_LIBRARY logical full-lib-spec type qualifiers [wild-full-spec]
```

```
$UPDATE_LIBRARY:
```

```
$  vmi$find 'p2 'p3 ""' s,e
```

```
$  if .not. $status then exit $status
```

```
$!
```

```
$  Update the state in the marker file so we can tell which library
```

```
$  we are updating right now.
```

```
$!
```

```
$  vmi$update_marker
```

```
$  read vmi$marker_file m
```

```
$  m[112,1]:= L
```

```
$  m[113,64]:= 'f$trnlnm(p2)
```

```
$  write/update vmi$marker_file m
```

```
$  close vmi$marker_file
```

```
$!
```

```
$  library/'p4'''p5 'p2 'p6
```

```
$  s = $status
```

```
$!
```

```
$  Update the state in the marker file so we can tell that we finished
```

```
$  updating the library.
```

```
$!
```

```
$  vmi$update_marker
```

```
$  read vmi$marker_file m
```

```
$  m[112,1]:=
```

```
$  write/update vmi$marker_file m
```

```
$  close vmi$marker_file
```

```
$  exit s
```

\$! The following callbacks display help for questions we ask.
\$
\$HELP_BACKUP:
\$ type sys\$input:
VMSINSTAL attempts to ensure that a power failure or system crash will not
corrupt your system disk. However, for absolute safety we recommend that
you back it up before installing new products. Please see the documentation
for more information on crashes during installation.
\$ exit
\$
\$HELP_CONSOLE_750:
\$ type sys\$input:
It is not necessary that a console volume be mounted in order to boot a
VAX-11/750. However, it is recommended that you keep one mounted anyway.
\$ exit
\$
\$HELP_CONTINUE:
\$ type sys\$input:
The above conditions may cause VMSINSTAL to function improperly. In
particular, problems may occur with file ownership if you are not logged
into the SYSTEM account. Furthermore, the existence of other processes
may lead to interactions with VMSINSTAL that cannot be anticipated.
If you continue at this point, you do so at your own risk.
\$ exit
\$
\$HELP_DEVICE:
\$ type sys\$input:
Please enter the device on which the distribution volume is to be mounted.
o If the kit is on console media, you can use the console device
or any other diskette or TU58 drive.

o If the kit is on magnetic tape, you can use any tape drive.

o If the kit is in a disk directory, you specify the disk
and directory in the standard format. If you have DECnet,
the directory can be on a remote node.
\$ exit
\$
\$HELP_GET:
\$ type sys\$input:
Please enter the device and directory into which you want the product
savesets copied. Once the savesets reside in this directory, you may
install the product directly from there, saving time over installation
from the console media.
\$ exit
\$
\$HELP_IVP:
\$ type sys\$input:
Most products provide an Installation Verification Procedure (IVP)
which verifies the completeness and accuracy of the installation.
You may wish to run the IVP immediately after installation.
\$ exit
\$
\$HELP_LIBRARY:
\$ type sys\$input:
In order to tailor the system disk, the library disk must be mounted.
Please enter the device specification of the drive on which the library
disk will be mounted.
\$ exit
\$
\$HELP_PLACE:
\$ type sys\$input:
If there is anything preventing you from mounting a volume on the device,

^{H 12}
such as lack of such a volume, then simply answer YES to this question.
§ exit

H 12

SHELP_PRODUCT:

S type sys\$input:

Please enter the names of the products you wish to process.

- o If the distribution is on console media, then there is only one product. Just enter an asterisk (*).
 - o If the distribution is on disk or tape, there can be multiple products. Enter a list of product names. An item in the list can be the facility code for the product (e.g., COBOL), in which case all versions and updates are processed in order. It can also be the identification of a specific version of a product. This is entered as the facility code followed immediately by the version and update numbers in the form vvu (e.g., COBOL020, VMS032).
 - o If the distribution is on disk or tape, you can process all products by entering an asterisk (*).

SHELP PURGE:

S type sys\$input:

During this installation, new files will be provided to replace existing versions. You may purge these older versions to save disk space, or keep them if you feel they may be of use. Purging is recommended.

S éxit

```

$! MOUNT_LIBRARY_DISK
$ 
$MOUNT_LIBRARY_DISK:
$   if f$trnlnm("LIB$SYSDEVICE") .nes. "" .and. f$trnlnm("LIB$SYSROOT") .nes. "" then goto mld10
$ 
$Mld2: vmi$callback ASK vmi$drv "On which drive will the library disk be mounted" -
$   s "" vmi$callback HELP_LIBRARY"
$   if f$loc("::" vmi$drv) .eq. f$len(vmi$drv) then vmi$drv == vmi$drv + ":"
$   vmi$drv == f$pars(vmi$drv, "DEVICE", "SYNTAX_ONLY")
$   if f$getdvi(vmi$drv, "EXISTS") then -
$     if f$getdvi(vmi$drv, "DEVCLASS") .eq. 1 then goto mld5
$   vmi$msg e badlibdev "Please specify an explicit disk drive."
$   goto mld2
$Mld5: if f$getdvi(vmi$drv, "MNT") then dismount/nounload 'vmi$drv'
$   vmi$callback PLACE VOLUME "Please mount the library disk on ''vmi$drv.''
$   if .not. $status then exit $status
$   d = vmi$drv
$   goto mld20
$ 
$Mld10: d = f$trnlnm("LIB$SYSDEVICE")
$   if f$getdvi(d, "MNT") then vmi$tailor dismount
$   if .not. $status then exit $status
$Mld20: vmi$tailor mount/write 'd
$   if .not. $status then exit $status
$   if vmi$alternate root then exit vmi$_success
$   vmi$find imagelib vmi$root:[syslib]imagerib.olb,"" s,e
$   vmi$find starlet vmi$root:[syslib]starlet.olb,"" s,e
$   d = f$trnlnm("IMAGELIB") - f$pars("IMAGELIB", "TYPE", "SYNTAX_ONLY") -
$     - f$pars("IMAGELIB", "VERSION", "SYNTAX_ONLY")
$   define imagelib 'd'
$   d = f$trnlnm("STARLET") - f$pars("STARLET", "TYPE", "SYNTAX_ONLY") -
$     - f$pars("STARLET", "VERSION", "SYNTAX_ONLY")
$   define starlet 'd'
$   exit vmi$_success
$ 
```

```
$! OPEN_REPORT_FILE logical file-spec heading
$OPEN_REPORT_FILE:
  open7write 'p2 'p3
  write 'p2 f$fao('!_!_!AS - !AS',p4,vmi$pretty)
  write 'p2 """
  write 'p2 """
  write 'p2 f$fao("It is !AS at !AS." -
    f$cvtime("ABSOLUTE","DATE"),f$ext(12.5,f$time()))
  n = f$strnlm("SYSSNODE") - "" - ":";,
  if n .eqs. "" then n = "no name"
  write 'p2 f$fao("Node: !AS, CPU Type: !UL, SID Register: !XL", -
    n,f$getsyi("CPU"),f$getsyi("SID"))
  write 'p2 f$fao("VMS Version: !AS",f$getsyi("VERSION"))
  write 'p2 f$fao("Target Disk: !AS (!UL blocks, !UL free), Label: !AS", -
    f$getdvi("VMISROOT","FULLDEVNAM") -
    f$getdvi("VMISROOT","MAXBLOCK"),f$getdvi("VMISROOT","FREEBLOCKS"), -
    f$getdvi("VMISROOT","VOLNAM"))
  write 'p2 f$fao("System Root: !AS",f$strnlm("VMISROOT"))
  if vmi$tailoring .eq. 2 then write 'p2 "Tailoring is in effect."
  write 'p2 """
  write 'p2 """
  exit vmi$success
```

```
$! PLACE_VOLUME prompt
$PLACE_VOLUME:
$  say p2
$  vmi$callback ASK vmi$ "Are you ready" "" b
$  if vmi$ then exit vmi$_success
$  vmi$callback ASK vmi$ "Is it impossible to fulfill the request" -
$    "" b "" vmi$callback HELP_PLACE"
$  if vmi$ then exit vmi$_failure
$  goto PLACE_VOLUME
```

```
$!      REMOUNT_CONSOLE  
$  
$REMOUNT_CONSOLE:  
$  d = "CSA1:"  
$  if f$getdvi(d,"ALL") then deallocate 'd  
$  n = f$getsyi("CPU")  
$  if n .[e. 8 then goto rc_n  
$  vmi$msg w noconsole "Unknown CPU type '"f$getsyi("CPU").'" -  
$          "Console volume cannot be remounted."  
$  exit vmi$_success  
$  
$rc_5:  
$rc_1: vmi$callback PLACE VOLUME "Please mount the console volume on CSA1:."  
$  if .not. $status then exit $status  
$  goto rc20  
$  
$rc_2: vmi$callback ASK vmi$ "Do you want to mount the console volume" -  
$          no b ""'vmi$callback HELP_CONSOLE_750"  
$  if .not. vmi$ then exit vmi$ success  
$  vmi$callback PLACE VOLUME "Please mount it on CSA1:."  
$  if .not. $status then exit $status  
$  goto rc20  
$  
$rc_3: if f$getdvi("CSA2:","MNT") then exit vmi$_success  
$  d = "CSA2:"  
$  goto rc20  
$  
$rc20: mount/foreign/system/protection=s:rwl/p/nowrite/noassist 'd console  
$  
$rc_4:  
$rc_6:  
$rc_7:  
$rc_8:  
$  exit vmi$_success
```

```

$-----+
$ This is the mainline for the installation procedure.
$ Step 1 is overall initialization.

$STEP_1:
$ Set up a few symbols for some fundamental options. Save the
$ default device/directory so we don't lose it when we deassign
$ all the logical names.

vni$version = 'V4.0'                                !VMSINSTAL version
vni$booting = f$loc('B',p4) .ne. f$len(p4)
vni$debug = f$loc('D',p4) .ne. f$len(p4)
vni$saved_dir = f$environment('DEFAULT')

Get rid of the user's personal commands so they don't screw us up.

delete = "delete"
if .not. vni$booting .and. .not. vni$debug then delete/symbol/global/all

Display a nice greeting message.

say = "write sys$output"
say f$fao("!/_!/_!AS!/_!/_!It is !AS at !AS.!/_!AS!", -
          "VAX/VMS Software Product Installation Procedure ''vni$version'', -"
          f$ctime('ABSOLUTE','DATE'),f$ext(12,5,f$time()), -
          "Enter a question mark (?) at any time for help.")

Disable CTRL/Y until the environment is consistent again.

set nocontrol=y

Set up the standard environment in which we operate.

vni$saved_msg = f$environment('MESSAGE')
set message/facility/severity/identification/text
if .not. vni$debug then vni$saved_privs = f$setprv("all,nobypass")
vni$saved_uic = f$user()
if .not. vni$debug then set uic [1,4]
vni$saved_prot = f$environment('PROTECTION')
set protection=(s:rwed,o:rwed,g:rwed,w:re)/default
set default missing:[missing]

Open a file for reading from the terminal.

open/read vni$terminal_file sys$command

Set up a bunch of static symbols, both internal and for the use
of the product installation procedures.

define = "define/nolog"
false = 0
true = 1
vni$callback = "@"+ f$elem(0,";",f$environment('PROCEDURE'))
vni$failure = %x10f50000
vni$find = vni$callback + " FIND_FILE"
vni$installing = true
vni$msg = vni$callback + " MESSAGE vmsinstal"
vni$no_error = "define/user sys$error nl:"
vni$no_output = "define/user sys$output nl:"
vni$qa = f$loc('Q',p4) .ne. f$len(p4)
vni$qa_fail == false

```

```

$ vmi$success = %x10f50001
$ vmi$failor = '@sys$update:vmstailor'
$ vmi$unsupported = %x10f50008
$ vmi$update marker = "open/read/write vmi$marker_file sys$update:vmimarker.dat"
$ w = f$edit(f$getsyi('VERSION'), 'TRIM,UPCASE')
$ w2 = f$ext(1,999,w)
$ if f$ext(0,1,w) .eqs. "V" .and. f$loc("...",w) .ne. f$len(w) then -
$   vmi$vms_version = 'RELEASED,' + -
$     f$fao('!ZL!AS',f$int(f$elem(0,"..",w2)),f$elem(1,"..",w2))
$ if f$ext(0,1,w) .nes. "V" .and. f$loc("...",w) .ne. f$len(w) then -
$   vmi$vms_version = 'UPDATE FT,' + -
$     f$fao('!ZL!AS',f$int(f$elem(0,"..",w2)),f$elem(1,"..",w2))
$ if f$loc(":",w) .eq. f$len(w) then -
$   vmi$vms_version = 'UPGRADE FT,' + w
$ 
```

Enable CTRL/Y.

From now on, errors or severe errors should cause us to quit.

```

$ on control_y then goto CONTROL_Y
$ set control=(t,y)
$ on error then goto error_done
$ 
```

Make sure any options were specified correctly.

```

$ if p3 .nes. "" then if p3 .nes. "OPTIONS" then vmi$msg f$inoptions -
$   "To specify options, parameter 3 must be OPTIONS."
$ 
```

If we are called with the B option, then STARTUP.COM thinks we need
to recover from a crash during installation.

```

$ if vmi$booting then goto STEP_12
$ 
```

Set up a symbol to say whether or not we are installing to an
alternate root. Define a logical name for the root.

```

$ vmi$alternate_root = f$loc('R',p4) .ne. f$len(p4)
$ if vmi$alternate_root then vmi$root = p5
$ if .not. vmi$alternate_root then vmi$root = f$strnlm("SYSSSPECIFIC")
$ vmi$specific = vmi$root
$ vmi$common_root = f$search("vmi$root[000000]syscommon.dir") .nes. ""
$ if vmi$common_root then vmi$root = vmi$root - "J" + "syscommon.J"
$ define/nolog/translation=(terminal,concealed) vmi$root 'vmi$root
$ define/nolog/translation=(terminal,concealed) vmi$specific 'vmi$specific
$ 
```

If the user has requested the file log option, then equate a bunch
of DCL verbs so they will log.

```

$ if f$loc('L',p4) .eq. f$len(p4) then goto 1d
$ append = "append/log"
$ backup = "backup/log"
$ copy = "copy/log"
$ create = "create/log"
$ delete = "delete/log"
$ librarian = "library/log"
$ library = "library/log"
$ purge = "purge/log"
$ rename = "rename/log"
$ 
```

1d:

If VMIORIG.TLR is still around, then we have detailored the system
disk but not retailored it. Make the user do something.

```

$ if f$search("sys$update:vmiorig.tlr") .nes. "" then -
$ 
```

vmi\$msg f vmiorig 'VMIORIG.TLR still exists; please see your documentation.'

If the user has requested the get saveset option, then skip on ahead to Step 2.

if f\$loc("G",p4) .ne. f\$len(p4) then goto STEP_2

Now we are going to check various environment items to ensure that we really can perform the installation. This includes account, privileges, quotas, the state of the network, and the status of other processes on the system.

a = f\$edit(f\$getjpi("", "USERNAME"), "TRIM") .nes. "SYSTEM"
 if a then vmi\$msg w notsystem "You are not logged in to the SYSTEM account."
 if f\$privilege("setprv") then goto 1e
 vmi\$msg w nosetprv "You are not running on an account with SETPRV privilege."

1e:
 a = true
 if f\$getjpi("", "ASTLM") .ge. 24 .and. -
 f\$getjpi("", "BIOLM") .ge. 18 .and. -
 f\$getjpi("", "BYTLM") .ge. 18000 .and. -
 f\$getjpi("", "DIOLM") .ge. 18 .and. -
 f\$getjpi("", "ENQLM") .ge. 30 .and. -
 f\$getjpi("", "FILLM") .ge. 20 then goto 1f

vmi\$msg w lowquota "One or more account quotas may be too low."

a = true
 1f:
 if f\$trnlnm("SYSSNODE") .eqs. "" then goto 1g
 vmi\$msg w decnet "Your DECnet network is up and running."

1g:
 a = true
 f = false
 m = f\$getjpi("", "PID")
 c = ""

1h:
 p = f\$pid(c)
 if p .eqs. "" then goto 1i
 if p .eqs. m .or. -
 f\$getjpi(p, "GRP") .le. 1 .or. -
 (f\$getjpi(p, "MODE") .eqs. "INTERACTIVE" .and. f\$getjpi(p, "TERMINAL") .eqs. "") then goto 1h
 if .not. f then vmi\$msg w active "The following processes are still active:"
 if .not. f then f = true
 say "", f\$getjpi(p, "PRCNAM")
 a = true
 goto 1h

1i:
 If any of the environment items were bad, ask the user about it.
 (Unless in Q/A mode, in which case they were warned...)

if vmi\$qa then goto STEP_2

if a then vmi\$callback ASK vmi\$ "Do you want to continue anyway" -
 no b ""'vmi\$callback HELP_CONTINUE'
 if a then if .not. vmi\$ then goto all_done

Verify that the user has backed up the system disk.

vmi\$callback ASK vmi\$ "Are you satisfied with the backup of your system disk" -
 yes b ""'vmi\$callback HELP_BACKUP'
 if .not. vmi\$ then goto all_done
 goto STEP_2

```

$! Step 2 obtains parameter 2 if it was not specified, and checks it
$!
$STEP_2:
$    vmi$command_p1 = ""
$!
$2a:   vmi$ == p2
$    if p2 .eqs. "" then vmi$callback ASK vmi$ "Where will the distribution volumes be mounted" -
$        s "" vmi$callback HELP_DEVICE"
$    vmi$place = vmi$ 
$    p2 =
$    if (f$loc("[",vni$place) .eq. f$len(vni$place)) .and.-
$        (f$ext(f$len(vni$place)-1,1,vni$place) .nes. ":"") then -
$        vni$place = vni$place + ":"
$    vni$remote = f$parse(vni$place,,, "NODE", "SYNTAX_ONLY"), nes. ""
$    vni$device = f$parse(vni$place,,, "DEVICE", "SYNTAX_ONLY")
$    vni$console = f$ext(0,3,vni$device) .eqs. "CSA"
$    if vni$remote .or. vni$console then goto 2d
$    if f$getdvi(vni$device, "EXISTS") then -
$        if f$getdvi(vni$device, "DEVCLASS") .eq. 1 .or. -
$            f$getdvi(vni$device, "DEVCLASS") .eq. 2 then goto 2d
$    vni$msg e baddisdev "Device "vni$device must be a disk or tape."
$    goto 2a
$2d:
$!
$! If we will be using the console device, connect it, dismount the
$! console volume, and allocate it so no one can get at it.
$!
$    vni$remount = vni$console
$    if .not. vni$console then goto 2g
$    sysgen = "$sysgen"
$    sysgen connect console
$    if .not. f$getdvi(vni$device, "EXISTS") then vni$msg e badconde - 
$        "Console device "vni$device does not exist."
$    if .not. f$getdvi(vni$device, "EXISTS") then goto 2a
$    if f$getdvi(vni$device, "MNT") then dismount/unload 'vni$device
$    allocate 'vni$device
$2g:
$!
$! Now that we've checked the device and it's definitely configured,
$! we can prepare our final distribution spec.
$!
$    if .not. vni$remote then vni$place = f$parse(vni$place,,, "DEVICE", "SYNTAX_ONLY") + -
$        f$parse(vni$place, "[000000]", "DIRECTORY", "SYNTAX_ONLY")
$    if vni$remote then vni$place = f$parse(vni$place,,, "NODE", "SYNTAX_ONLY") + -
$        f$parse(vni$place,,, "DEVICE", "SYNTAX_ONLY") + f$parse(vni$place,,, "DIRECTORY", "SYNTAX_ONLY")
$    vni$first_next = "first"
$    goto STEP_3

```

\$! Step 3 determines which products are to be processed and the order
in which to process them.

\$STEP_3:

\$! Find out which products the user wishes to process.

```
say ""
vmi$list == p1
if vmi$command_p1 .nes. "" then goto all_done
vmi$command_p1 = p1
if p1 .nes. "" then goto 3b
say "Enter the products to be processed from the ",vmi$first_next,-
    " distribution volume set."
vmi$callback ASK vmi$list 'Products' "" s,z """vmi$callback HELP_PRODUCT"""
if vmi$list .eqs. ""^Z"" .or. vmi$list .eqs. 'EXIT' then goto all_done
3b: p1 =
vmi$first_next = "next"
```

If necessary, have the user mount the first distribution volume.
If a disk volume is mounted Files-11, it is assumed to be the right one.

```
if vmi$remote then goto 3f
if f$getdvi(vmi$device,'DEVCLASS') .eq. 1 .and. -
    f$getdvi(vmi$device,'MNT') .and. -
    .not. f$getdvi(vmi$device,'FOR') then goto 3f
if f$getdvl(vmi$device,'MNT') then dismount/nounload 'vmi$device'
if vmi$qa then goto 3c
vmi$callback PLACE_VOLUME 'Please mount the first volume of the set on "'vmi$device.'"
if .not. $status then goto STEP_2
```

mount/override=id/noassist/nowrite 'vmi\$device'
vmi\$remount = true

Ensure that the distribution spec that the user entered can be
used to search the volume.

```
w = f$parse(vmi$place) .eqs. ""
if w then vmi$msg e baddir "Directory ''vmi$place does not exist."
if w then goto STEP_2
```

Now we will determine which products are to be processed by matching
the contents of the distribution volume to the user's list.

```
if vmi$list .nes. "*" then vmi$list == "," + vmi$list + ","
vmi$list_count = 0
3l:   s = f$search(vmi$place+"*.a;")
      if s .eqs. "" then goto 3m
      p = f$parse(s,,,"NAME","SYNTAX_ONLY")
      if vmi$list .nes. "*" .and. -
          f$loc("*.+pt",vni$list) .eq. f$len(vni$list) .and. -
          f$loc("*.+f$ext(0,f$len(p)-3,p)+",vni$list) .eq. f$len(vni$list) then goto 3l
      vni$list_count = vni$list_count + 1
      vni$list*vni$list_count = p
      goto 3l
```

3m:
if vni\$list_count .eq. 0 then vni\$msg e noprods -
 'None of the specified products were found.'
if vni\$list_count .eq. 0 then goto STEP_3

Now we will sort the product names in alphabetical order. This is a
straight selection sort taken from Knuth volume 3, section 5.2.3.

```

$      j = vmi$list_count
$3s1a:   if j.lt. 2 then goto 3s1z
$          i = j
$          n = j - 1
$3s2a:   if vmi$list'n .gts. vmi$list'i then i = n
$          n = n - 1
$          if n.ge. 1 then goto 3s2a
$          t = vmi$list'j
$          vmi$list'j = vmi$list'i
$          vmi$list'i = t
$          j = j - 1
$          goto 3s1a
$3s1z:
$!
$!      Now we will list the products we intend to process.
$!
$!      say f$fao("!/The following products will be processed:!/")
$!      i = 1
$3t:   p = vmi$list'i
$        v = f$ext(f$len(p)-3,3,p)
$        say f$fao(" !AS V!UL.!AS",p-v,f$int(f$ext(0,2,v)),f$ext(2,1,v))
$        i = i + 1
$        if i .le. vmi$list_count then goto 3t
$!
$!      Dismount the distribution volume and remount it foreign for BACKUP.
$!      Don't bother if we didn't mount it in the first place.
$!
$!      if vmi$remount then dismount/nounload 'vmi$device
$!      if vmi$remount then mount/foreign/noassist/nowrite/nomessage 'vmi$device
$!
$!      If the user specified the get saveset option, then we skip on ahead
$!      to a special step which does the work. Otherwise we continue on
$!      to install the products.
$!
$!      vmi$list_done = 0
$!      if f$loc?"G",p4) .ne. f$len(p4) then goto STEP_13
$!      goto STEP_4

```

```
$! Step 4 establishes the file environment for the installation.  
$! This is only done once.  
$STEP_4:  
$  
$ if f$type(vmi$tailoring) .nes. "" then goto STEP_5  
$  
$! If this is a tailored system, we want to detailor the system disk.  
$! Don't bother if we are installing to an alternate root.  
$  
$ vmi$tailoring = f$getsyi("TAILORED") .and. .not. vmi$alternate_root  
$ if vmi$tailoring then vmi$callback DETAILOR  
$ if vmi$tailoring then vmi$tailoring = 2  
$  
$ goto STEP_5
```

```
$!      Step 5 begins the installation of the next product on the list.  
$  
$STEP_5:  
$  
$!      Determine the next product to be installed.  
$  
$  vmi$list_done = vmi$list_done + 1  
$  if vmi$list_done .gt. vmi$list_count then goto STEP_3  
$  vmi$product = vmi$list'vmi$list$done  
$  
$!      Create the marker file that will allow us to recover from system  
$!      crashes. This file contains enough state information so that we  
$!      can determine how to finish up the installation after a crash.  
$  
$  open/write vmi$marker_file sys$update:vmimarker.dat  
$  write vmi$marker_file-f$fao('!64AS!39AS!8AS!2AS!64AS!1AS!1AS!64AS',-  
$          f$trnlm('VMI$ROOT'), -           target system root  
$          vmi$product, -                  facility and version  
$          f$str(vmi$tailoring), -        tailoring flag  
$          'B', -                          state B = before  
$          "", -                         Library being updated  
$          f$str(vmi$alternate_root),-    True if alternate root  
$          f$str(vmi$common_root),-       True if common root  
$          f$trnlm('VMI$SPECIFIC')       System specific root  
$  
$  close vmi$marker_file  
$  
$!      Tell the user what we are about to do.  
$  
$  v = f$ext(f$len(vmi$product)-3,3,vmi$product)  
$  vmi$pretty = f$fao("!AS V!UL..!AS",vmi$product-v,f$int(f$ext(0,2,v)),f$ext(2,1,v))  
$  say f$fao("!/!/! Beginning installation of !AS at !AS!/", -  
$          vmi$pretty,f$ext(12,5,f$time()))  
$  goto STEP_6
```

\$! Step 6 establishes the complete environment for the kit's installation procedure. H 13
 \$
 \$STEP_6:
 \$
 \$! If the kit's working directory exists, get rid of it.
 \$
 \$ if f\$search(''vmi\$root:[sysupd]''vmi\$product.dir''),eqs. ''' then goto 6b
 \$ w = f\$search(''vmi\$root:[sysupd.''vmi\$product]*.*;'')
 \$ if w .nes. ''' then -
 \$ set prot=s=rwed vmi\$root:[sysupd.'vmi\$product...]*.*;*
 \$ if w .nes. ... then -
 \$ delete vmi\$root:[sysupd.'vmi\$product...]*.*;*
 \$ set prot=s=rwed vmi\$root:[sysupd]'vmi\$product.dir;*
 \$ delete vmi\$root:[sysupd]'vmi\$product.dir;*
 \$6b:
 \$
 \$! Record the number of free blocks for use by the callbacks.
 \$
 \$ vmi\$free_blocks = f\$getdvi(''VMISROOT'', ''FREEBLOCKS'')
 \$
 \$! If the statistics option was requested, then start up a demon subprocess to do it. Wait for the process to record the initial state of the system.
 \$
 \$ if f\$loc(''S'',p4) .eq. f\$len(p4) then goto 6q
 \$ if f\$getjpi(''PRCCNT'') .gt. 0 then stop vmsinstal\$demon
 \$ say ''(Waiting for demon to record initial state...''
 \$ define/table=lnm\$job vmi\$demon 60
 \$ spawn/process=vmsinstal\$demon/input=nl:/output=nl:/nowait/nolog -
 \$ vmi\$callback STATISTICS DEMON
 \$ set process/priority='f\$int(f\$getjpi(''PRIB'')+1) vmsinstal\$demon
 \$ wait 00:00:02
 \$ if f\$trnlnm(''VMISDEMON'', ''LNMSJOB'') .nes. ''' then goto 6p
 \$ say "...done"
 \$6q:
 \$
 \$! If the auto-answer option was requested, then find out if there is an answer file or if we will be creating it.
 \$
 \$ if f\$loc(''A'',p4) .eq. f\$len(p4) then goto 6f
 \$ if f\$search(''vmi\$root:[sysupd]''vmi\$product.ans'') .nes. ''' then goto 6e
 \$ vmi\$auto_option = ''W''
 \$ open/write vmi\$auto_file vmi\$root:[sysupd]'vmi\$product.ans
 \$ vmi\$msg i recordans "An auto-answer file will be recorded."
 \$ goto 6f
 \$6e:
 \$ vmi\$auto_option = ''R''
 \$ open/read vmi\$auto_file vmi\$root:[sysupd]'vmi\$product.ans
 \$ vmi\$msg i useans "The auto-answer file will be used."
 \$6f:
 \$
 \$! If the callback trace option was requested, then create a file to contain the trace.
 \$
 \$ if f\$loc(''C'',p4) .ne. f\$len(p4) then -
 \$ vmi\$callback OPEN REPORT FILE vmi\$call_file -
 \$ vmi\$root:[sysupd]'vmi\$product.cbt "Callback Trace"
 \$
 \$! Create the kit's working directory.
 \$
 \$ define vmi\$kwd vmi\$root:[sysupd.'vmi\$product]
 \$ create/directory/protection=o:rwed vmi\$kwd

| 13

\$! Create a file to receive the sequence of deferred callbacks.
\$ open/write vmi\$defer_file vmi\$kwd:vmidefer.com
\$!
\$ Set up default values for items defined by the callbacks.
\$
\$ vmi\$ivp == false | Invoke product IVP
\$ vmi\$purge == true | Purge files after installation
\$ vmi\$reboot == false | Reboot after installation
\$ vmi\$safety == true | Sacrifice disk blocks for safer installation
\$ vmi\$startup == "!" | Invoke product startup command procedure
\$ goto STEP_7

\$! Step 7 restores the primary kit saveset (letter A) into the
\$! working directory. J 13

\$STEP_7:

\$ vmi\$callback RESTORE_SAVESET a
\$ if .not. \$status then goto STEP_11

\$ There better be a KITINSTAL procedure in that saveset.

\$ if f\$search("vmi\$kwd:kitinstal.com") .nes. ""' then goto STEP_8
\$ vmi\$msg e noproc "The kit's installation procedure is missing."
\$ goto STEP_11

```
$!      Step 8 invokes the kit's installation procedure.  
$  
$STEP_8:  
$  
$      Update the state in the marker file so we can tell that the  
$      installation has begun.  
$  
$      vmi$update_marker  
$      read vmi$marker_file m  
$      m[111,2]:= 'f$ext(vmi$safety,1,'US')      ! state S = safety, U = unsafe  
$      write/update vmi$marker_file m  
$      close vmi$marker_file  
$  
$      Invoke the procedure, telling it to install the product.  
$  
$      @vmi$kwd:kitinstal VMIS_INSTALL 'f$int(f$loc("K",p4) .ne. f$len(p4))  
$  
$      Check that the procedure succeeded.  If not, forget the rest of  
$      this installation.  
$  
$      if $status then goto STEP_9  
$      vmi$msg e insfail "The installation of ""vmi$pretty has failed."  
$      goto STEP_11
```

```
$!      Step 9 performs all of the deferred callbacks.  
$  
$STEP_9:  
$  
$!      If we were operating in safety mode, we will be spending some time  
$!      performing deferred callbacks. Tell the user.  
$  
$      if vmi$safety then vmi$msg i movefiles -  
$!          "files will now be moved to their target directories..."  
$  
$!      Update the state in the marker file so we can tell that the  
$!      deferred callbacks have begun.  
$  
$      vmi$update_marker  
$      read vmi$marker_file m  
$      m[111,2]:= D                      ! state D = deferred  
$      write/update vmi$marker_file m  
$      close vmi$marker_file  
$  
$!      Clear the safety flag so that all deferred callbacks are now  
$!      performed immediately.  
$  
$      vmi$safety == false  
$  
$!      Reopen the deferred callback file and read each record in it.  
$!      Perform the callbacks and check their status.  
$  
$      close vmi$defer_file  
$      open/read vmi$defer_file vmi$kwd:vmidefer.com  
$9a:   read/end_of_file=9b vmi$defer_file c  
$      vmi$callback 'c  
$      if $status then goto 9a  
$      vmi$msg e insdfail "The installation of ''vml$pretty has failed."  
$      goto STEP_11  
$9b:  
$      close vmi$defer_file  
$  
$!      Update the state in the marker file so we can tell that the  
$!      installation is essentially complete.  
$  
$      vmi$update_marker  
$      read vmi$marker_file m  
$      m[111,2]:= A                      ! state A = after  
$      write/update vmi$marker_file m  
$      close vmi$marker_file  
$      goto STEP_10
```

```
$! Step 10 invokes the product startup procedure and the IVP,  
$! if available and requested.  
$  
$STEP_10:  
$  
$! Before invoking anything, set the default directory to the working  
$! directory, because in a realistic situation it would point  
$! somewhere. Also get rid of the VMISCALLBACK symbol so callbacks  
$! can't be used inadvertently.  
$  
$! set default vmi$kwrd  
$! a = "delete /symbol"           !In case L option selected  
$! a vmi$callback  
$  
$! If the installation procedure requested that a startup procedure  
$! be invoked, do that now.  
$  
$! vmi$startup  
$  
$! Invoke the IVP, if available, and remember its return status.  
$  
$! if vmi$ivp then @vmi$kwrd:kitinstal VMIS_IVP  
$! s = $status  
$! if .not. vmi$ivp then s = 1  
$  
$! Restore the standard environment.  
$  
$! set default missing:[missing]  
$! vmi$callback = '@'+ f$elem(0,";",f$environment("PROCEDURE"))  
$  
$! Tell the user what happened.  
$  
$! if s then say f$fao("!! Installation of !AS completed at !AS!!", -  
$!           vmi$pretty,f$ext(12,5,f$time()))  
$! if .not. s then vmi$msg e ivpfail "The IVP for ''vmi$pretty has failed."  
$! goto STEP_11
```

```
$!      Step 11 cleans up and loops back for the next product.
```

```
$SSTEP_11:
```

```
$!      Delete the kit's working directory.
```

```
$ close /nolog vmi$defer_file
$ set prot=s=rwed vmi$kwd:*.*/*,vni$root:[sysupd]'vni$product.dir;*
$ delete vni$kwd:*.*/*
$ delete vni$root:[sysupd]'vni$product.dir;*
```

```
$!      If we are collecting statistics, tell the demon we're done and
$!      wait for it to finish up.
```

```
$ if f$loc("S",p4) .eq. f$len(p4) then goto 11e
$ define/table=lnm$job vni$demon GO
$ say "(Waiting for demon to generate report...)"
$11d:    wait 00:00:02
$        if f$trnlnm("VMI$DEMON","LNMS$JOB") .nes. "" then goto 11d
$        say "...done"
```

```
$11e:
$!      Close any working files that might be open. Delete the marker file
$!      so no one thinks we're still doing an installation.
```

```
$ close /nolog vni$auto_file
$ close /nolog vni$call_file
$ close /nolog vni$marker_file
$ set prot=s=rwed sys$update:vmimarker.dat;*
$ delete sys$update:vmimarker.dat;*
```

```
$!      If we are to reboot, tell the user and finish up the installation.
```

```
$ if .not. vni$reboot then goto STEP_5
$ vni$msg i reboot "This product requires that the system be rebooted."
$ if vni$list_done .lt. vni$list_count then vni$msg w prodskip -
$     "Products that have not been installed will be skipped."
$ goto all_done
```

Step 12 is a special step that performs recovery if the system crashes during an installation. We are called from STARTUP.COM if a marker file exists during booting.

STEP_12:

Open a file for reading from the terminal.

open/read vmi\$terminal_file sys\$output

If there is a marker file, then get the information from it and set up some symbols.

```
open/read/error=all_done vmi$marker_file sys$update:vmimarker.dat;
read/error=all_done vmi$marker_file m
close vmi$marker_file
define/translation=(terminal,concealed) vmi$root 'f$ext(0,64,m)
vmi$product = f$edit(f$ext(64,39,m),"TRIM")
vmi$tailoring = f$int(f$ext(103,f,m))
vmi$alternate_root = f$int(f$ext(177,1,m))
vmi$common_root = f$int(f$ext(178,1,m))
vmi$specific = f$edit(f$ext(179,64,m),"TRIM")
define vmi$kwd vmi$root:[sysupd.'vmi$product]
vmi$purge == false
vmi$safety == false
```

Tell the user what is happening. Then case on the state we were in when the crash occurred.

```
v = f$ext(f$len(vmi$product)-3,3,vmi$product)
vmi$pretty = f$fao("!AS V!UL.!AS",vmi$product-v,f$int(f$ext(0,2,v)),f$ext(2,1,v))
vmi$msg, i recover """vmi$pretty was being installed when the system crashed."""
goto 12'f$ext(111,2,m)
```

\$12B: type sys\$input:

Nothing on your system disk had been changed before the crash. Simply begin the installation again.

goto 12z

\$12S: type sys\$input:

Although files on your system disk may have been changed, the system should be in a usable state. Simply begin the installation again.

goto 12z

\$12SL: type sys\$input:

The following library was being updated when the system crashed. Other than that, the system should be in a usable state. Restore the library from backup and then begin the installation again.

```
say " ",f$ext(113,64,m)
goto 12z
```

\$12U:

\$12UL: type sys\$input:

The installation was being performed in such a way as to minimize disk usage. One or more files may now be in an unusable state. Please restore your system disk from backup and then begin the installation again.

goto 12z

```
$12D: type sys$input:
```

VMSINSTAL will attempt to complete the installation, because it was almost done before the crash. Please ignore any error messages listed below. After booting is completed, you MUST boot the system again.

```
$  
$      if vmi$tailoring .eq. 2 then vmi$callback MOUNT_LIBRARY_DISK  
$      if .not. $status then goto 12z  
$      open/read/error=12z vmi$defer_file vmi$kwd:vmidefer.com  
$12Da:    read/error=12Dd vmi$défer_file c  
$          vmi$callback 'c  
$          goto 12Da  
$12Dd:  close vmi$defer_file  
$      goto 12z  
$
```

```
$12A: type sys$input:
```

The installation was completed satisfactorily.

```
$      goto 12z  
$
```

```
$!     Now we can just complete VMSINSTAL in the normal fashion.
```

```
$12z: set prot=s=rwed vmi$kwd:*.*/*,vmi$root:[sysupd]'vmi$product.dir;*  
$      delete vmi$kwd:*.*/*  
$      delete vmi$root:[sysupd]'vmi$product.dir;*  
$      type sys$input:
```

Please read your documentation for a complete description of installation crash recovery. There may be additional things that you need to do manually, such as purging the system disk or restoring the tailored environment.

```
$      goto all_done
```

\$:
\$: Step 13 is a special step that handles the get saveset option. The
\$: user simply wants to copy all the savesets for the specified
\$: products from the distribution volume into a disk directory set
\$: aside for the purpose.
\$:

\$:STEP_13:
\$:

\$: Determine the directory in which the savesets are to be copied, and
\$: ensure that the disk is mounted and ready.
\$:

```
$:say """
$:vmi$p6 = p6
$p6 =
$13a: vmi$ == p5
if p5 .eqs. "", then vmi$callback ASK vmi$ "Into which directory are the savesets to be copied" -
      "s """,vmi$callback HELP_GET"
get_dir = vmi$
$p5 .eqs.
if .not. f$getdvi(get_dir,'EXISTS') then goto 13b
get_for = 0
get_class = f$getdvi(get_dir,'DEVCLASS')
get_dev = f$getdvi(get_dir,'FULLDEVNAM')
scratch_dev = get_dev
if (get_class .eq. 1) .and. f$getdvi(get_dev,'MNT') -
    .and. .not. f$getdvi(get_dev,'FOR') then goto 13d
get_dir = get_dev + f$parses(get_dir,['00000000'],,"DIRECTORY","SYNTAX ONLY")
if f$getdvi(get_dev,'MNT') .and. f$getdvi(get_dev,'FOR') then goto T3a1
if f$getdvi(get_dev,'MNT') then dismount /nownload 'get_dev'
v = "with BACKUP options ''vmi$p6 applied."
if vmi$p6 .eqs. "" then v = ""
v1 = "mounted foreign."
if (get_class .eq. 1) then v1 = "initialized and mounted."
vmi$msg i devmnt "Device ''get_dev will be ''v1''' 'v
v =
if get_class .eq. 1 then v = "disk,pack"
if get_class .eq. 2 then v = "tape"
if v .eqs. "" then v = "media"
vmi$callback place_volume "Place a blank ''v in ''get dev."
if (get_class .eq. 1) then initialize 'get_dev vaxvmskit
v = "/foreign"
if (get_class .eq. 1) then v = "/over=id"
mount /noassist /write 'v' 'get_dev
```

\$13a1:
get_for = 1
goto 13e

\$13b: vmi\$msg e devnotexist "Device ''get_dir does not exist."

\$13d: goto 13a
if f\$parses(get_dir) ,nes. "" then goto 13e

\$13e: vmi\$msg e badgetdir "Directory ''get_dir does not exist."
goto 13a

\$: If necessary, create a top-level work directory on the scratch disk.
This directory will be used to restore the savesets, so that new
savesets can then be created in the target directory.

```
$:if get_for then scratch_dev = "SYSSYSDEVICE:"
scratch_dev = f$getdvi(scratch_dev,'FULLDEVNAM')
define vmi$kw 'scratch_dev'[vmiwork]
v = scratch_dev + "[000000]vmiwork.dir"
td = f$trnlnm('VMISKWD')
scratch_credir = f$search(v) .eqs. ""
if scratch_credir then vmi$msg i createdir "Creating temporary directory ''td."
```

```

$ if scratch_credir then create/directory/protection=o:rwed vmi$kwed E 14
$ w = f$search('vmi$kwd:*.**')
$ if w .nes. "" then set prot=s=rwed vmi$kwd:*.**;*
$ if w .nes. "" then delete vmi$kwd:*.**;*
$!
$! Tell the user about the funny error messages that will happen.
$ type sys$input:

Because VMSINSTAL does not know how many savesets comprise a software
product, it will simply copy as many as it can find. Do not be
concerned about error messages from BACKUP after all savesets have
been copied.
$!
$! Now we go into a main loop to restore the savesets for each product
in the list. Tell the user which product we're going to get,
and then restore and back up each saveset into the target directory.
Finally, tell the user how many savesets were copied.
$13l:
$   vmi$list_done = vmi$list_done + 1
$   if vmi$list_done .gt. vmi$list_count then goto 13r
$   vmi$product = vmi$list'vmi$list_done
$   v = f$ext(f$len(vmi$product)-3,3,vmi$product)
$   vmi$pretty = f$fao('!AS V!UL.!AS',vmi$product-v,f$int(f$ext(0,2,v)),f$ext(2,1,v))
$   say f$fao('!/. Getting savesets for !AS!',vmi$pretty)
$   w = f$search('vmi$kwd:*.**')
$   if w .nes. "" then set prot=s=rwed vmi$kwd:*.**;*
$   if w .nes. "" then delete vmi$kwd:*.**;*
$   w = f$search('')
$   saveset = 0
$13n:
$     letter = f$ext(saveset,1,'ABCDEFGHIJKLMNPQRSTUVWXYZ')
$     if saveset .ge. 26 then (letter = f$string(saveset+1)
$     vmi$callback RESTORE_SAVESET 'letter
$     if .not. $status then goto 13p
$     saveset = saveset + 1
$     on error then goto 13s
$     if f$length(letter) .gt. 1 then =
$       letter = f$fao('VMI !4ZL',f$integer(letter))
$     backup/interchange/verify vmi$kwd:*.**;* -
$       'get_dir'vmi$product.'letter/save_set 'vmi$p6
$     on error then goto error_done
$     vmi$p6 = ""
$     w = f$search('vmi$kwd:*.**')
$     if w .nes. "" then set prot=s=rwed vmi$kwd:*.**;*
$     if w .nes. "" then delete vmi$kwd:*.**;*
$     w = f$search('')
$     goto 13n
$13p:
$   say f$fao('!/. A total of !UL saveset!XS copied for !AS', -
$             saveset,vmi$pretty)
$   goto 13l
$13r:
$!
$! Delete the work directory.
$ w = f$search('vmi$kwd:*.**')
$ if w .nes. "" then set prot=s=rwed vmi$kwd:*.**;*
$ if w .nes. "" then delete vmi$kwd:*.**;*
$ w = f$search('')
$ if scratch_credir then set prot=s=rwed 'scratch dev[000000]vmiwork.dir;*
$ if scratch_credir then delete 'scratch_dev[000000]vmiwork.dir;*
$!
$! Loop back for more products to get.

```

```
$  
$  
$  
$!  
$13s: goto STEP_3  
$  
$ An error occurred in creating the saveset  
$  
$ set noon  
$ vmi$msg e.backuperr "An error has occurred in recreating the saveset." -  
$ "The contents of scratch directory ''td will be deleted."  
$ w = f$search("vmi$kwd:*.*)"  
$ if w .nes. "" then set prot=s=rwed vmi$kwd:*.*;*  
$ if w .nes. "" then delete vmi$kwd:*.*;*  
$ w = f$search("")  
$ if scratch_credir then set prot=s=rwed 'scratch dev[000000]vmiwork.dir;*  
$ if scratch_credir then delete 'scratch_dev[000000]vmiwork.dir;*  
$ if (get_class .eq. 1) then delete 'get_dir''vmi$product.'letter;  
$ goto error_done
```

1
VM
Pr
VM
It
VM
VM
Th
ou
th
de
Th
th

1.
In
in
yo
pr
do
ca
(p

```

$! We come down here when we are all done. Hopefully the installation
$! was completed successfully, but we come here in any case.
$!
$!error_done:
$!    set noon
$!    vmi$msg f unexpected "Installation terminated due to unexpected event."
$!    vmi$reboot == false
$!    vmi$sqa_fail == true
$!all_done:
$!    on control_y then goto ad20
$!    set noon
$!
$!    If the demon is still running, shut it down now
$!
$!    if f$loc("S",p4) .eq. f$len(p4) then goto ad15
$!    vmi$no_error
$!    vmi$no_output
$!    set process/priority='f$int(f$getjpi("", "PRIB")+1) vmsinstal$demon
$!    if .not. $status then goto ad15
$!    define/table=lnm$job vmi$demon GO
$!    say "(Waiting for demon to generate report...)"
$!ad11:
$!    wait 00:00:02
$!    if f$trnlnm("VMISDEMON", "LNMSJOB") .nes. "" then goto ad11
$!    say "...done"
$!ad15:
$!    Close any working files we might have open. Delete any marker file.
$!
$!    close /nolog vmi$call_file
$!    close /nolog vmi$defer_file
$!    close /nolog vmi$marker_file
$!    close /nolog vmi$product_file
$!    close /nolog vmi$temp_file
$!    w = f$search('sys$update:vmimarker.dat')
$!    if w .nes. "" then -
$!        set_prot=s=rwed sys$update:vmimarker.dat;*
$!    if w .nes. "" then -
$!        delete sys$update:vmimarker.dat;*
$!
$!    Restore the file environment. This includes retailoring
$!    a small system disk.
$!
$!    if f$trnlnm("STARLET", "LNMSPROCESS") .nes. "" then deassign starlet
$!    if f$trnlnm("IMAGELIB", "LNMSPROCESS") .nes. "" then deassign imagelib
$!    if ""'vmi$tailoring'"' .eqs. "2" .and. .not. vmi$booting then -
$!        vmi$callback RETAILOR
$!
$!    Dismount any distribution volume that might still be mounted.
$!    If we're using the console device, remount the console volume.
$!
$!    if ""'vmi$remount'"' .nes. "1" then goto ad20
$!    if f$getdvi(vmi$device, "MNT") then dismount/nounload 'vmi$device
$!    if vmi$console then vmi$callback REMOUNT_CONSOLE
$!
$!    Restore the original environment as saved when we started.
$!
$!ad20:
$!    These two files have to be closed now, and not before, because they
$!    are used by the ASK callback in REMOUNT_CONSOLE.
$!
$!    close /nolog vmi$auto_file
$!    close /nolog vmi$terminal_file
$!    set default 'vmi$saved_dir
$!    set protection=(vmi$saved_prot)/default

```

```
$ if .not. vmi$debug then set uic 'vmi$saved_uic
$ if .not. vmi$debug then w = f$setprv(vmi$saved_privs)
$ set message 'vmi$saved_msg
$!
$ If we were to reboot after the last product, do it.
$!
$ if """vmi$reboot"" .eqs. "1" .and. .not. vmi$debug then -
$   @sys$system:shutdown minimum -
$   "Reboot after 'vmi$pretty installation.' -"
$   no yes "soon" no REBOOT_CHECK
$!
$ We're all done.
$!
$ if f$trnlm("VMI$ROOT","LNMSPROCESS") .nes. "" then deassign vmi$root
$ if f$trnlm("VMI$SPECIFIC","LNMSPROCESS") .nes. "" then deassign vmi$specific
$ if f$trnlm("VMI$", "LNMSPROCESS") .nes. "" then deassign vmi$
$ if f$trnlm("VMI$KWD","LNMSPROCESS") .nes. "" then deassign vmi$kwd
$ say f$fao("!-/! VMSINSTAL procedure done at !AS!/-!", -
$       f$ext(T2,5,f$time()))
$ exit
```

2
A
a
st
di
saTh
de
foTh
al
UI
mu

```
$-----  
$ This callback performs the "detailoring" of the system disk if we  
$ have decided that this system requires it. The goal is to create  
$ as much free space on the system disk as possible. Anything that  
$ prevents us from detailoring is a fatal error.  
$-----
```

```
SDETAILOR:  
S say ""  
S vmi$msg i smalldisk "This is a small disk system."  
S type sys$input:
```

This is a small disk system and must be tailored to accomodate your new software products. The current tailoring environment will be recorded, and then system files that are not required will be temporarily removed from the system disk. After installation, the original environment will be restored.

```
S! Get the library disk mounted.  
S  
S vmi$callback MOUNT LIBRARY_DISK  
S if .not. $status then vmi$msg f nolibdisk "Library disk could not be mounted."  
S  
S Record the current tailoring environment and make sure that the  
S system and library disks agree.  
S  
S if vmi$debug then goto dt40  
S vmi$tailor record vmiorig vmidiff  
S set prot=s=rwed sys$update:vmiorig.tlr;*,vmidiff.tlr;*  
S if f$file("sys$update:vmidiff.tlr","EOF") .eq. 0 then goto dt18  
S vmi$msg e libdiff "The following system and library disk files do not match."  
S type sys$update:vmidiff.tlr  
S delete sys$update:vmiorig.tlr;*,vmidiff.tlr;*  
S vmi$msg f libdiff2 "You must resolve these differences before continuing."  
Sdt18: delete sys$update:vmidiff.tlr;*  
S  
S Tell the user how to recover if we die now. We're going to delete  
S all but the required files from the system disk.  
S  
S type sys$input:  
Sdeck
```

The current tailoring environment has been recorded in VMIORIG.TLR. If a power failure or drastic error occurs during this installation, you can restore the environment by entering the following commands after mounting the library disk. Please see your documentation for more details.

```
$ @SYS$UPDATE:VMTAILOR COPY VMIORIG  
$ DELETE SYS$UPDATE:VMIORIG.TLR;*
```

```
Seod  
S b = f$getdvi('VMISROOT','FREEBLOCKS')  
S vmi$tailor delete vmiorig  
S a = f$getdvi('VMISROOT','FREEBLOCKS')  
S say f$fao('(!UL blocks were made available by tailoring.)',a-b)  
S vmi$tailoring_blocks == a - b + 200  
Sdt40:  
S exit vmi$_success
```

```

$-----  

$ This callback performs the "retailoring" of the system after the  

$ installation is complete. The goal is to restore the user's initial  

$ tailoring environment.  

$  

$RETAILOR:  

$   say ""  

$   vmi$msg i tailoror "Your original tailoring environment will now be restored."  

$   if vmi$debug then goto rt29  

$  

$ Check that there is enough disk space to bring back the files.  

$ If not, just tell the poor bastard and forget it.  

$  

$   f = f$getdvi("VMI$ROOT","FREEBLOCKS")  

$   if f .ge. vmi$tailoring_blocks then goto rt29  

$   vmi$msg w nospace "There are not enough blocks to restore the system disk."  

$   type sys$input:  

$  

$ There are not enough free blocks left on the system disk to restore your  

$ original tailoring environment. The environment has been recorded and  

$ may be restored manually (as described above) after sufficient space has  

$ been obtained by purging and/or deleting files.  

$   say f$fao('Only !UL blocks of the required !UL are available.!/', -  

$           f, vmi$tailoring_blocks)  

$   exit vmi$_failure  

$rt29:  

$  

$   Restore the original environment.  

$  

$   d = f$trnlnm("LIB$SYSDEVICE")  

$   if vmi$debug then goto rt30  

$   if .not. f$getdvi("LIB$SYSDEVICE",'EXISTS') then goto rt40  

$   if .not. f$getdvi("LIB$SYSDEVICE",'MNT') then vmi$tailor mount 'd  

$   vmi$tailor copy vmiorig  

$   set prot=s=rwed sys$update:vmiorig.tlr;*  

$   delete sys$update:vmiorig.tlr;*  

$  

$   Remount the library disk with no write access.  

$  

$rt30:  

$   if f$getdvi(d,'MNT') then vmi$tailor dismount  

$   vmi$tailor mount 'd  

$   exit vmi$_success  

$rt40:  

$   type sys$input  

$  

$ The library disk has been dismounted, and VMSINSTAL is unable to  

$ restore your previous tailored configuration. After VMSINSTAL  

$ exits, you must manually tailor your configuration by mounting  

$ the library disk, issuing the command '@SYSSUPDATE:VMSTAILOR COPY VMIORIG',  

$ and then delete SYSSUPDATE:VMIORIG.TLR.  

$  

$   exit vmi$_success

```

3
Th
pr
sy
st
fa3.
VM

```

$-----+
$ This is the statistics demon callback. It is responsible for watching
$ the installation of a product and producing a statistics report.
$ We have been spawned with a higher priority than the installer.
$STATISTICS DEMON:
$ on error then goto sd99
$ sav_bypass = f$setprv("BYPASS")
$ directory = "directory/noheader/notrailer/exclude=vmi*.*/width=file=48/column=1/size"
$ set default vmi$root:[sysupd]

$! Take a snapshot of the system directories before we begin.

$ directory/output=vmidemon.pre vmi$root:[*...]
$ i = f$getdvi("VMISROOT",'FREEBLOCKS')
$ b = f$time()

$! Open up a file to contain the statistics report.

$ define s vmi$root:[sysupd]'vmi$product.anl
$ vmi$callback OPEN_REPORT_FILE vmi$stat_file s "System Disk Statistics Report"

$! Turn on retention so all file accesses are marked with an expiration
$ date. Then tell our parent that it can do the installation.

$ if .not. vmi$debug then set volume/retention=(-:::01,-:::01) 'f$getdvi("VMISROOT",'FULLDEVNAM')
$ deassign/table=lnm$job vmi$demon

$! Now we just sit in a loop, watching the free block count during the
$ installation. VMISDEMON will appear when the installation is done.

$ n = i
$ x = i
$sd20:
$      wait 00:00:00.50
$      f = f$getdvi("VMISROOT",'FREEBLOCKS')
$      if f .lt. n then n = f
$      if f .gt. x then x = f
$      if f$Strnlnm("VMISDEMON",'LNMSJOB') .eqs. "" then goto sd20

$! Turn off retention. Then report various block statistics.

$ if .not. vmi$debug then set volume/retention=(-:::00,-:::00) 'f$getdvi("VMISROOT",'FULLDEVNAM')
$ write vmi$stat_file f$fao("!21<Initial Free Blocks:>!7UL",i)
$ write vmi$stat_file f$fao("!21<Minimum Free Blocks:>!7UL",n)
$ write vmi$stat_file f$fao("!21<Maximum Free Blocks:>!7UL",x)
$ write vmi$stat_file f$fao("!21<Final Free Blocks:>!7UL",f)
$ write vmi$stat_file ""
$ write vmi$stat_file f$fao("!21<Peak Utilization:>!7UL (Initial-Minimum)",i-n)
$ write vmi$stat_file f$fao("!21<Net Utilization:>!7SL (Initial-Final)",i-f)
$ close vmi$stat_file
$ convert/fdl=sys$input: s s
$RECORD
$ CARRIAGE_CONTROL carriage_return
$ purge s

$! Now look at the system directories and report files added, deleted,
$ modified, and accessed.

$ append sys$input: s

```

FILES ADDED

```

$ directory/output=vmidemon.pst vmi$root:[*...]
$ difference/output=vmidemon.dif/separated=revision/match=1/nonumber -
  vmidemon.pre .pst
$ search/output=vmidemon.sch vmidemon.dif -
  "***","file ",difference,vmidemon/match=nor
$ append vmidemon.sch s
$ append sys$input: s
FILES DELETED

$ difference/output=vmidemon.dif/separated=master/match=1/nonumber -
  vmidemon.pre .pst
$ search/output=vmidemon.sch vmidemon.dif -
  "***","file ",difference,vmidemon/match=nor
$ append vmidemon.sch s
$ append sys$input: s
FILES MODIFIED

$ directory/output=vmidemon.tmp/modified/since=""'b'"' vmi$root:[*...]
$ open/read vmi$temp_file vmidemon.tmp
$ open/append vmi$stat_file s
$sd30:   read/end_of_file=sd39 vmi$temp_file f
$       if f$cvtime(f$file(f,'CDT')) .[ts. f$cvtime(b) then -
          write vmi$stat_file f
$       goto sd30
$sd39:   close vmi$temp_file
$       close vmi$stat_file
$ append sys$input: s

FILES ACCESSED (except installed images)

$ directory/output=vmidemon.tmp/expired/since=""'b'"'/before=""'f$time()"' vmi$root:[*...]
$ append vmidemon.tmp s
$!
$ Clean up and tell our parent that we're done.

$sd99:   set noon
$       if .not. vmi$debug then set volume/retention=(-:::00,-:::00) 'f$getdvi("VMI$ROOT",'FULLDEVNAM')
$       set prot vmidemon.*;*
$       delete vmidemon.*;*
$       x = f$setprv(sav_bypass)
$       deassign/table=l$nm$job vmi$demon
$       exit

```

0232 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

