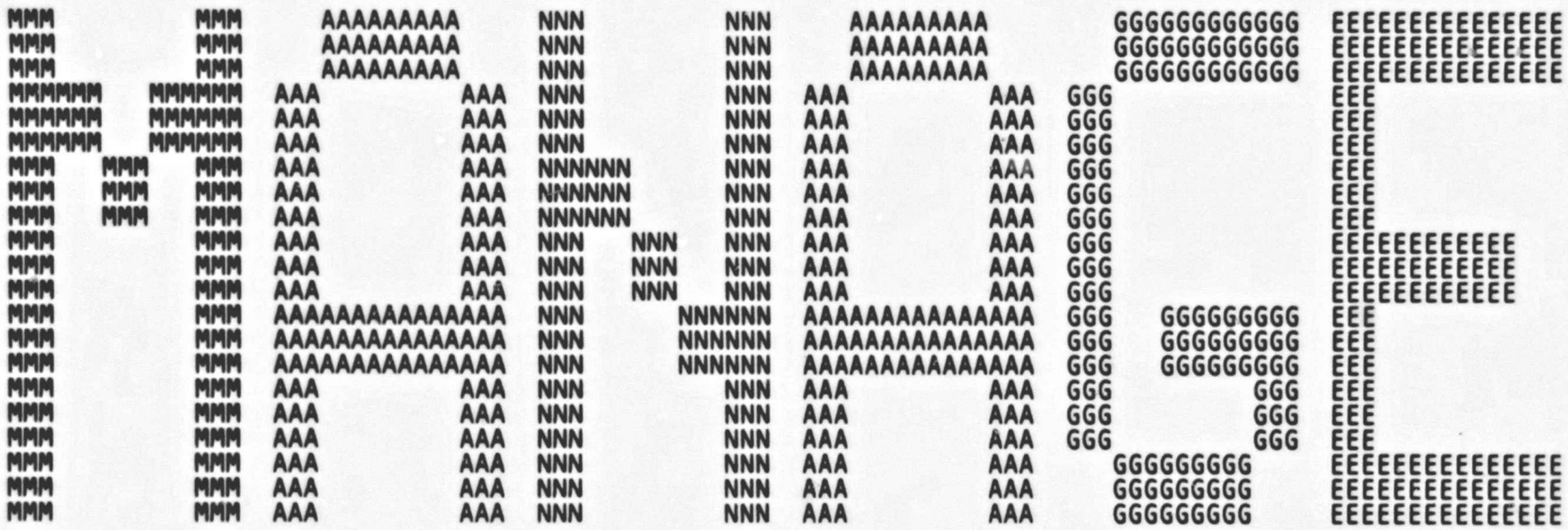
MMM MMM AAAAAAAAA NNN NNN AAAAAAAAA GGGGGGGGGGG EEEEEEEEEEEEEEE
MMM MMM AAAAAAAAA NNN NNN AAAAAAAAA GGGGGGGGGGG EEEEEEEEEEEEEEE
MMM MMM AAAAAAAAA NNN NNN AAAAAAAAA GGGGGGGGGGG EEEEEEEEEEEEEEE
MMMMMM MMMMMM AAA AAA NNN NNN AAA AAA GGG EEE
MMMMMM MMMMMM AAA AAA NNN NNN AAA AAA GGG EEE
MMMMMM MMMMMM AAA AAA NNN NNN AAA AAA GGG EEE
MMM MMM MMM AAA AAA NNNNNN NNN AAA AAA GGG EEE
MMM MMM MMM AAA AAA NNNNNN NNN AAA AAA GGG EEE
MMM MMM MMM AAA AAA NNNNNN NNN AAA AAA GGG EEE
MMM MMM AAA AAA NNN NNN NNN AAA AAA GGG EEEEEEEEEEE
MMM MMM AAA AAA NNN NNN NNN AAA AAA GGG EEEEEEEEEEE
MMM MMM AAA AAA NNN NNN NNN AAA AAA GGG EEEEEEEEEEE
MMM MMM AAAAAAAAAAAAAAA NNN NNNNNN AAAAAAAAAAAAAAA GGG GGGGGGGGG EEE
MMM MMM AAAAAAAAAAAAAAA NNN NNNNNN AAAAAAAAAAAAAAA GGG GGGGGGGGG EEE
MMM MMM AAAAAAAAAAAAAAA NNN NNNNNN AAAAAAAAAAAAAAA GGG GGGGGGGGG EEE
MMM MMM AAA AAA NNN NNN AAA AAA GGG GGG EEE
MMM MMM AAA AAA NNN NNN AAA AAA GGG GGG EEE
MMM MMM AAA AAA NNN NNN AAA AAA GGG GGG EEE
MMM MMM AAA AAA NNN NNN AAA AAA GGGGGGGGG EEEEEEEEEEEEEEE
MMM MMM AAA AAA NNN NNN AAA AAA GGGGGGGGG EEEEEEEEEEEEEEE
MMM MMM AAA AAA NNN NNN AAA AAA GGGGGGGGG EEEEEEEEEEEEEEE

```
VV      VV  MM      MM   SSSSSSSS   IIIIII   NN      NN   SSSSSSSS  TTTTTTTTTT    AAAAAA    LL
VV      VV  MM      MM   SSSSSSSS   IIIIII   NN      NN   SSSSSSSS  TTTTTTTTTT    AAAAAA    LL
VV      VV  MMMM  MMMM  SS             II     NN      NN  SS            TT       AA    AA   LL
VV      VV  MMMM  MMMM  SS             II     NN      NN  SS            TT       AA    AA   LL
VV      VV  MM  MM  MM  SS             II     NNNN    NN  SS            TT       AA    AA   LL
VV      VV  MM  MM  MM  SS             II     NNNN    NN  SS            TT       AA    AA   LL
VV      VV  MM      MM    SSSSSS       II     NN  NN  NN    SSSSSS      TT       AA    AA   LL
VV      VV  MM      MM    SSSSSS       II     NN  NN  NN    SSSSSS      TT       AA    AA   LL
VV      VV  MM      MM          SS     II     NN    NNNN          SS    TT       AAAAAAAAAA   LL
VV      VV  MM      MM          SS     II     NN    NNNN          SS    TT       AAAAAAAAAA   LL
  VV  VV    MM      MM          SS     II     NN      NN          SS    TT       AA    AA   LL                ....
  VV  VV    MM      MM          SS     II     NN      NN          SS    TT       AA    AA   LL                ....
    VV      MM      MM   SSSSSSSS   IIIIII   NN      NN   SSSSSSSS     TT       AA    AA   LLLLLLLLLL        ....
    VV      MM      MM   SSSSSSSS   IIIIII   NN      NN   SSSSSSSS     TT       AA    AA   LLLLLLLLLL        ....

MM      MM  EEEEEEEEEE  MM      MM
MM      MM  EEEEEEEEEE  MM      MM
MMMM  MMMM  EE          MMMM  MMMM
MMMM  MMMM  EE          MMMM  MMMM
MM  MM  MM  EE          MM  MM  MM
MM  MM  MM  EE          MM  MM  MM
MM      MM  EEEEEEEE    MM      MM
MM      MM  EEEEEEEE    MM      MM
MM      MM  EE          MM      MM
MM      MM  EE          MM      MM
MM      MM  EE          MM      MM
MM      MM  EE          MM      MM
MM      MM  EEEEEEEEEE  MM      MM
MM      MM  EEEEEEEEEE  MM      MM
```

DEVELOPER'S GUID
DEVELOPER'S GUIDE

to the

VAX/VMS VMSINSTAL PROCEDUR
VAX/VMS VMSINSTAL PROCEDURE

VMS Development

16 September 1984

1 INTRODUCTION

VMSINSTAL is a procedure which supports the installation of software products on an existing VAX/VMS system. It is used to install both VMS products (updates and upgrades) and optional software products. It completely subsumes the functionality of VMSUPDATE and the current VMSINSTAL. Our current plans are to remove VMSUPDATE and the old VMSINSTAL in a future major release.

This document describes the features of VMS V4.0 VMSINSTAL, and lays out a set of guidelines for layered products to use when designing their installation procedures. Therefore, some of the features described herein do not apply to the VMS V3.x versions of VMSINSTAL. The use of VMSINSTAL and adherence to the guidelines will accomplish the following goals.

- Product kits are composed of BACKUP savesets, which allow files larger than a single console volume to be included in the kit. Furthermore, BACKUP can recover from media errors in some cases.
- Products may be installed from a wide variety of media, including console media, tapes, disks, and the network.
- Multiple products can be included on the same distribution volume. This is useful for software specialists and may become a distribution scheme in the future.
- Consistency among software product installations will be increased.
- Software product installation procedures should be more immune to changes in VMS from one version to the next.

1.1 Software Product Conventions

In order to produce a software product that can be smoothly integrated into the VMS environment, and that can coexist with other products, you must follow certain conventions when designing and building the product. These conventions are described in detail in various documents available from the Spit Brook SQM group. All such documents can be obtained by logging into the VMSINFO account on AURORA (password VMSINFO).

1.2 Identifying Products

Each VAX software product is identified by a facility code. This is a sequence of up to six alphameric characters. The facility code must be registered with the central VMS product registrar, who can be contacted by sending mail to AURORA::Facility_Registrar.

As versions and updates are released, they are assigned unique version/update numbers. These are 3-digit integers in the form vvu, where vv is the major version number and u is the update. For example, VMS V3.0 receives the number 030, while the first update will be 031.

1.3 A Quick User's Guide

The system manager installs software products by logging into the SYSTEM account and invoking VMSINSTAL. No other commands are required to establish the environment. The VMSINSTAL procedure is invoked as follows:

```
$ @SYS$UPDATE:VMSINSTAL [product,...] [device]
```

The first parameter specifies the product(s) that the user wishes to install. VMSINSTAL prompts if this parameter is omitted. A product can have one of three forms.

- facvvu - The particular version/update of the specified product is installed.
- fac - All versions and updates of the specified product are installed in order.
- * - Every product on the distribution volume is installed by installing all versions and updates in order.

Products are installed in alphabetical order.

The second parameter identifies the device where the distribution volumes are to be mounted. VMSINSTAL prompts if this parameter is omitted.

The procedure also accepts various options which are described in Chapter 5.

2 PACKAGING SOFTWARE PRODUCTS

A VMS software product is distributed to the customer in a form called a "kit". A kit consists of one or more BACKUP savesets, each with a standardized name. VMSINSTAL allows these savesets to reside on any disk or tape media, including, of course, the console media. Each saveset is named as follows:

```
facvvu.s        where fac is the facility code
                      vv  is the major version number
                      u   is the update number
                      s   is a sequence letter
```

The facility code and version/update number have already been described. The sequence letter provides for 26 savesets per kit. The following paragraphs describe the kits in greater detail.

- Floppy - Each volume label for a floppy kit should be in the format facnn, where fac is the facility code and nn is a sequence number. The kit must be created directly on the master distribution volumes using BACKUP to create sequential disk savesets. The following qualifiers are required: /INTERCHANGE, /VERIFY, /BLOCK_SIZE=9000, /GROUP_SIZE=25. The latter two result in optimal use of the blocks on the floppies. Don't forget to initialize double-density floppies for single-density use, or the SDC will have problems.

- TU58 - Just like floppies.

- Magnetic Tape - The magnetic tape volume label should be in the format fac, where fac is the facility code. The kit must be created directly on the master volume using BACKUP to create savesets. The following qualifiers are required: /INTERCHANGE, /VERIFY. The savesets must be placed on the tape in order. If desired, any number of kits may be placed on one magnetic tape.

- Files-11 Disk - A kit may be created directly into an arbitrary directory of a Files-11 disk. This is not currently a distribution method, but may become one in the future. The following qualifiers are required: /INTERCHANGE, /VERIFY. If desired, any number of kits may be placed in a directory.

The conventions described for volume labels are strong suggestions, although VMSINSTAL requires no particular labelling scheme. The owner UIC of files in the savesets is irrelevent. The protection of files must be as specified in section 3.3.1.

2.1 The SPKITBLD Procedure

The rather messy rules described above are embodied in a DCL procedure named SPKITBLD.COM. This procedure may be used to build your kits, or you may steal code from it for your own procedures. The latest version of SPKITBLD will be kept in the same public directory as this document.

SPKITBLD is invoked as follows:

```
$ @SPKITBLD [facvvu] [device] [a-files]
```

The first parameter is the name of the kit to be built. VMSINSTAL prompts if this parameter is omitted.

The second parameter is the device on which the kit is to be built. If you want to build the kit in a Files-11 directory, then the parameter can include a directory specification. Volume initialization is performed when appropriate. VMSINSTAL prompts if this parameter is omitted.

The third parameter is a list of specifications of the files to be placed in the primary kit saveset A. Stickiness prevails and wildcards are allowed. VMSINSTAL prompts if this parameter is omitted.

SPKITBLD always prompts for information about additional savesets.

3 THE INSTALLATION PROCEDURE

The VMSINSTAL procedure, a bundled and supported VMS facility, provides the logic to install software products on an existing VMS system. It accomplishes this installation by interacting in a standardized fashion with the software product kit. This interaction falls into three major categories.

1. VMSINSTAL assumes that the kit has been created as described in Chapter 2.

2. VMSINSTAL assumes that the principal saveset is the one with sequence letter A. This saveset must begin on volume 1 of the kit, and it must contain a DCL procedure named KITINSTAL.COM.

3. KITINSTAL.COM drives the installation of the product by requesting services from VMSINSTAL. These services are requested by recursively invoking VMSINSTAL with the appropriate parameters. Hence they are referred to as "callbacks".

3.1 Overview Of VMSINSTAL Logic

VMSINSTAL performs the following steps when invoked.

1. Initialize and set up the standard environment.

2. Prompt for the device parameter if not supplied by the user, and verify this parameter.

3. Ask the user which products are to be installed from the next distribution volume set. Also allow the user to exit. Mount the first volume of the set and make a list of products to be installed.

4. Establish the final file environment for the installation. This includes the use of tailoring to create the maximum free space on the system disk (if this is a small disk system).

5. Prepare to install the next product on the list. If there are no more products, go back to step 3.

6. Set up the environment necessary to install the product. This includes the creation of a working directory for the kit's installation procedure.

7. Restore all of saveset A into the kit's working directory.

8. Invoke the KITINSTAL procedure restored from saveset A. The procedure will utilize callbacks to effect the installation. Some callbacks may not be performed immediately, but deferred until after the procedure has completed.

9. Performs all callbacks which were deferred in step 8.

10. Invoke the product's Installation Verification Procedure (IVP) if available.

11. Clean up and delete the kit's working directory. Loop back to step 5 for the next product.

12. This is a special step that is only performed if the system crashed during an installation. It is invoked directly from STARTUP.COM to clean up after the crash. See Appendix B for more detail.

13. This is a special step that is performed when copying kit savesets into a disk directory for later installation. See Chapter 5 for more detail.

The following sections expand upon the above information.

3.2 The Kit's Installation Procedure

The primary installation procedure for a software product is named KITINSTAL.COM. Subprocedures may also be supplied if needed. The primary procedure must accept a request code as its first parameter. When the procedure is invoked by VMSINSTAL to install the product, it does so in the following manner.

```
$ @KITINSTAL VMI$_INSTALL kit-debug
```

Therefore, every kit's procedure must handle the request code VMI$_INSTALL, which means to perform an installation of the product. The request code is also used under other circumstances, as described throughout this document. The procedure must not blindly perform a GOTO to the request code, treating it as a label. This is because we may add request codes in the future, which will cause an error. The procedure must check for specific requests and ignore all others.

The kit-debug parameter is a boolean value which specifies whether or not the kit debug option was requested when VMSINSTAL was invoked. Please see Chapter 5 for more detail.

The installation procedure must exit back to VMSINSTAL with a status. If the installation was successful, exit with the status VMI$_SUCCESS. If it failed in an unrecoverable way, exit with VMI$_FAILURE. If an unknown request code is received, exit with VMI$_UNSUPPORTED.

3.3 The Installation Environment

When VMSINSTAL executes the kit's installation procedure, it provides a strictly-defined environment in which the procedure must operate.

3.3.1 Defaults

When the kit's installation procedure is invoked, the following process defaults will be in effect. The procedure must not change the defaults except via callbacks.

- All components of the message prefix will be displayed.
- All privileges will be enabled, except for BYPASS.
- The UIC will be [1,4].
- The default file protection will be system:RWED, owner:RWED, group:RWED, world:RE. This may be changed in the future. Note that files restored from a product saveset will keep their original protection, which should be the same as the default. The process default only pertains to files created during the installation.
- The default device and directory will be MISSING:[MISSING]; therefore all file references must be explicit.

3.3.2 Naming Conventions

Any logical names or global symbols which VMSINSTAL defines will begin with the prefix VMI$. If the kit's procedure needs logical names or global symbols, their names must begin with the facility code and a dollar sign (e.g., FORT$). The kit's procedure may define local symbols with any name.

The kit's procedure should be very careful to avoid abbreviations in no
DCL commands. Verbs must not be abbreviated because special symbols may be defined for them by VMSINSTAL. Qualifiers should also be spelled out, but abbreviation to four letters or more is allowed if absolutely necessary.

If the kit needs installation subprocedures, they may be assigned any name. Any number of additional levels of procedure invocation are allowed (up to the limits of DCL), but callbacks may only be performed from KITINSTAL.COM and its direct callees.

3.3.3 Logical Names

The following logical names are defined when the kit's procedure is invoked.

- VMI$KWD - The kit's working directory created by VMSINSTAL. All references to the working directory must be made via this logical name.
- VMI$ROOT - The top-level system directory for the target
  Al
  system on which the product is to be installed. All references to system directories must be of the for references to system directories must be of the form VMI$ROOT:[SYSxxx]. You cannot use the SYS$ logical names VMI$ROOT:[SYSxxx]. You cannot use the SYS$ logical names, nor can you refer directly to the system device nor can you refer directly to the system device.
- VMI$SPECIFIC - The system-specific top-level system directory for the target system on which the product is to be installed. In those cases where no common system directory structure is in use, VMI$SPECIFIC will be identical to VMI$ROOT. If a common system directory exists, VMI$SPECIFIC will still point to the system-specific top-level system directory while VMI$ROOT will point to the common top-level system directory.

3.3.4 Global Symbols

The following global symbols are defined when the kit's procedure is invoked.

- TRUE and FALSE - Symbolic constants which can be used to assign boolean values to symbols.
- VMI$ALTERNATE_ROOT - A boolean symbol which specifies whether or not the product is being installed to an alternate root. You should try to write an installation procedure which doesn't need to know.
- VMI$CALLBACK - This is the symbol that is used to perform callbacks to VMSINSTAL.

- VMI$COMMON_ROOT - A boolean symbol which specifies whether or not the product is being installed to a common system root.

- VMI$CONSOLE - A boolean symbol which specifies whether or not the distribution device is a console device.

- VMI$DEVICE - The distribution device as an ASCII string.

- VMI$_FAILURE - The failure status returned by callbacks and by the kit's procedure. It has a severity of warning.

- VMI$PLACE - The location of the distribution volume as an ASCII string.

- VMI$PRODUCT - The product identification in the standard format facvvu.

- VMI$REMOTE - A boolean symbol which specifies whether or not the distribution volume is on a remote node.

- VMI$_SUCCESS - The success status returned by callbacks and by the kit's procedure.

- VMI$_UNSUPPORTED - The status returned by the kit's installation procedure if it is passed an unknown request code.

- VMI$VMS_VERSION - The version of VMS which is currently running. If a released version, this string will have the form "RELEASED,vvu". If a update field test version, this string will have the form "UPDATE FT,vvu". Finally, if an upgrade field test baselevel, this string will have the form "UPGRADE FT,xxxx", where xxxx is the full baselevel identification (not necessarily four characters). You cannot tell from the baselevel identification which version is being field tested. (See the appendices for examples on how to decode VMI$VMS_VERSION.)

3.4 Additional Conventions

This section describes some additional conventions that the kit's installation procedure must follow. These are miscellaneous conventions and are described in no particular order.

3.4.1 Error Handling

The kit's installation procedures must have an ON WARNING statement to handle errors. These errors may occur in the kit's procedures themselves, or they may be returned by a callback (VMI$_FAILURE is a warning status). In the best of all possible worlds, this ON WARNING statement can just exit, propagating the status up a level:

```
$       ON WARNING THEN EXIT $STATUS
```

If the procedure must do some cleanup, however, then the error handling might look as follows:

```
$       ON WARNING THEN GOTO ERROR
  .
  .
  .
$ERROR:
$       S = $STATUS
$       Clean up.  You must close any files you opened.
$       Make sure to handle any errors that occur in the
$       clean up.
$       EXIT S
```

3.4.2 Asking Questions

Ask all questions of the user at the beginning of the installation procedure, so that afterwards the user can go out for a quickie. If the installation is being done from a fast device, such as a real disk or tape, the user won't have to wait around during the entire installation.

3.4.3 SET Command

Use of the SET command is limited to the following cases.

- SET VERIFY if in kit debug mode.
- SET ON and SET NOON
- SET FILE to alter files in the kit's working directory only.

3.4.4 SHOW Command

The kit's installation procedures must never rely on the format of output from the SHOW command, or any other utility.

3.4.5 Compatibility Mode

The kit's installation procedures must not make use of compatibility mode under the assumption that it is bundled with VMS. Compatibility mode may be unbundled in the future.

3.4.6 Referencing Other Products

Installation procedures may reference and invoke other optional software products. There are essentially four levels of such referencing.

1. The procedures do not need to reference any other optional products.

2. The procedures need to alter their flow based upon the existence of another product, but do not reference the product. Use the FIND_FILE callback to determine the existence of the product.

3. The procedures need to reference another product's files, but do not invoke the product. Use the FIND_FILE callback to determine the existence of the product and set up a logical name to reference a file.

4. The procedures need to invoke another product. Use the FIND_FILE callback to determine the existence of the product. Invoke the product in the standard fashion. NOTE that the invocation of another product probably involves implicit references to the other product's files. VMSINSTAL assumes these files are in the system root or on a user disk.

3.4.7 Global State

If the installation procedure changes the global state of the system, except insofar as callbacks are concerned, it must restore the state before terminating. Some examples are as follows.

- If the installation procedure needs to INSTALL an image in order to complete the installation, it must deINSTALL the image before exiting. This pertains only to images needed during the installation itself. INSTALLs that must be done for product use should be done in the product-specific startup command procedure (see the SET STARTUP callback).

3.5 The IVP

Each product should provide an Installation Verification Procedure which verifies the completeness and accuracy of installation. If the kit's installation procedure declares that such a procedure exists, then VMSINSTAL will perform it after the installation is completed. VMSINSTAL does this by invoking KITINSTAL as folows.

```
$ @VMI$KWD:KITINSTAL VMI$_IVP
```

Thus, a kit's installation procedure must handle the request code VMI$_IVP if it declares an IVP. VMSINSTAL tries to set up a realistic environment for the IVP before it runs. The following points are relevent.

- All files will be in their final resting place.
- If the installation procedure declared a product-specific startup procedure with the SET STARTUP callback, VMSINSTAL will invoke it before the IVP.
- The default directory will be set to the kit's working directory, thus simulating a user's default directory in real life.
- The IVP cannot use callbacks.

KITINSTAL.COM must exit back to VMSINSTAL with a status. Exit with VMI$_SUCCESS if the IVP is successful, or with VMI$_FAILURE if the IVP is unsuccessful. Note that VMSINSTAL does not attempt to "undo" an installation if the IVP fails. It is up to the installation procedure to ensure that the product is installed in its entirety and to further ensure that the IVP will not fail. This may sound sarcastic, but adherence to such a philosophy will produce the best installation procedure.

4 THE CALLBACKS

This chapter describes the callbacks provided by VMSINSTAL. The syntax of each callback is presented in the standard pseudo-BNF format. Some of the metalinguistic symbols require further explanation.

- KEYWORDS - Keywords are specified in upper case. They must not be abbreviated in any way.
- logical - A logical name to be defined by the callback. These logical names must begin with the facility prefix fac$. By convention, use the logical name consisting simply of the prefix if it will never be referenced after being specified in the callback.
- symbol - A global symbol to be equated by the callback. Global symbols must begin with the facility prefix fac$. Again, use just the prefix if the symbol is never referenced.
- dd - A disk and directory specification. References to the kit's working directory are made using the logical name VMI$KWD. References to system directories are made using the logical name VMI$ROOT and an explicit directory, such as VMI$ROOT:[SYSLIB]. Other references must specify an explicit device logical name (not device number) and a directory. A logical name may be used for the entire specification. At the time of reference, the disk must be mounted and the directory must exist.
- nt - A file name and type. The type is always required, but no version may be specified. Neither logical names nor wildcarding are allowed.
- ddnt - A complete file specification, consisting of a disk, directory, name, and type. A logical name may be used, but wildcarding is not allowed.
- options - All options are specified as a comma-separated list of single-character codes. Embedded spaces are not allowed.

All callbacks return an exit status. VMI$_SUCCESS is returned if the callback could be completed without error. VMI$_FAILURE, a warning, is returned if anything went wrong.

4.1 ASK

This callback is used to ask the user a question and obtain a valid answer. It also supports the question mark help feature of VMSINSTAL.

```
$ VMI$CALLBACK ASK symbol prompt [default] [options] [help]
```

Parameters

symbol - The name of a global symbol to be equated to the answer. The answer will be blank compressed, blank trimmed, uncommented, and upcased.

prompt - A quoted string containing the question to be asked. Do not include any trailing punctuation or whitespace; this is formatted automatically.

default - The default answer. If not specified, the user must enter an answer. Specify the default for boolean questions as "YES" or "NO".

options - A comma-separated list of single-character options.

- o B - Boolean. The user must answer YES or NO. The integer 1 or 0 is returned as the answer.
- o D - Double space. A line is skipped before asking the question.
- o H - Help first. The help information is displayed before the question is asked. This is useful for very complicated questions.
- o I - Integer. The user must answer with an integer.
- o N - Null answer. A null answer is allowed. This only makes sense when a string is requested and no default is provided.
- o R - Ring bell. The terminal bell is rung before the question is asked.
- o S - String. The answer can be any character string. This is the default data type.
- o Z - CTRL/Z returned. If the user enters CTRL/Z, it is returned as a string of the form "^Z". Without this option, CTRL/Z is ignored.

help - Help information. This can be a quoted string containing the help, or a quoted procedure invocation. If a procedure invocation, it must begin with an at sign (@) and may invoke any procedure in the kit. Typically it would invoke KITINSTAL.COM and pass it a request code, as follows:

"@VMI$KWD:KITINSTAL HELP_WHATEVER"

Because the request code does not begin with VMI$, it can't conflict with a request code passed by VMSINSTAL.

4.2 CHECK_NET_UTILIZATION

This callback is used to check that there are enough free blocks on the system disk to successfully complete the installation. The net
mus
block usage for a product must be obtained using the statistics option (see Chapter 5).

$ VMI$CALLBACK CHECK_NET_UTILIZATION symbol blocks

Parameters

symbol - The name of a global symbol, set to true if the required free blocks are available, false otherwise.

blocks - The net number of blocks used by the product, that is, the number of blocks ultimately used when the installation is complete.

4.3 CHECK_VMS_VERSION

This callback is obsolete, but is retained in VMSINSTAL for compatability with V3. Use the VMI$VMS_VERSION global symbol for these checks. (See the appendices for an example on how to decode VMI$VMS_VERSION)

This callback allows you to check the version of the running VMS system, in the event that your product has some dependency on the version. A test is performed to determine if the specified version requirements are met.

$ VMI$CALLBACK CHECK_VMS_VERSION symbol [version] [baselevel]

Parameters

symbol - The name of a global symbol which is set to true if the test is passed, false if not.

version - This parameter is used when a released version of VMS is currently running. It is in the format vvu, and the test is passed if the specified version of VMS, or a later version, is running. If the parameter is omitted, all VMS versions cause the test to fail.

baselevel - This parameter is used when a field test baselevel of VMS is currently running. If in the format xxx, where xxx is the baselevel identification excluding the leading X or Y, then the test is passed if the specified baselevel of VMS is running. If in the format xxx-yyy, then the test is passed if a baselevel of VMS in the specified range is running. If the parameter is omitted, all field test baselevels cause the test to pass.

4.4 CONTROL_Y

This callback must be invoked when the user enters CTRL/Y. The first command in every installation procedure must either be as specified below, or it must go to a label which performs cleanup and then the CONTROL_Y callback.

```
$ ON CONTROL_Y THEN VMI$CALLBACK CONTROL_Y
```

Notes

The callback returns a fatal status, which will result in the execution of your ON WARNING statement. If you do not include ever this line in every command procedure, and the user enters CTRL/Y at the wrong time, all hell breaks loose.

4.5 CREATE_ACCOUNT

This callback is used to create a new account in SYSUAF.DAT. It should be used sparingly, if ever.

```
$ VMI$CALLBACK CREATE_ACCOUNT username qualifiers
```

Parameters

username - The username to be associated with the account.

qualifiers - A sequence of qualifiers as accepted by the ADD command of the AUTHORIZE utility. The qualifiers must be enclosed in quotation marks.

Notes

In most instances, software products need not create new accounts. As an example, it is usually unnecessary to create a system management account for your product, because the system manager can perform all management from the standard SYSTEM account. Your product must not assume that it knows the UIC for a new account, but must ask the user.

4.6 CREATE_DIRECTORY

This callback is used to create a directory on the system disk or some other user disk.

```
$ VMI$CALLBACK CREATE_DIRECTORY {SYSTEM hierarchy} [qualifiers]
                                {USER   dd        }
```

Parameters

SYSTEM heirarchy - This combination of parameters is used when you want to create a directory underneath the system root. The directory name must not begin with "SYS". For example, SYSTEM SYSHLP.FOO will create the directory VMI$ROOT:[SYSHLP.FOO].

USER dd - This combination is used to create an arbitrary directory on a user disk. For example, USER WRKD$:[SNORK] will create the top-level SNORK directory on the user work disk.

qualifiers - A sequence of qualifiers for the CREATE/DIRECTORY command. You can specify one or more of the following: /OWNER_UIC, /PROTECTION, /VERSION_LIMIT. The qualifiers must be enclosed in quotation marks.

Notes

Do not create directories on a user disk unless absolutely necessary. Most layered products are associated with a particular system root and thus should appear in that root.

4.7 DELETE_FILE

This callback is used to delete an obsolete file created by a previous installation. Remember, BYPASS privilege is not enabled.

```
$ VMI$CALLBACK DELETE_FILE ddnt
```

Parameters

ddnt - The complete specification of the file to be deleted.

4.8 FIND_FILE

This callback is used by VMSINSTAL whenever it must locate a file specified in another callback. You must also use FIND_FILE itself when you need to reference a system file for some purpose not supported by a callback. Files in the kit's working directory may be referenced directly.

```
$ VMI$CALLBACK FIND_FILE logical ddnt [default] locate [symbol]
```

Parameters

logical - A logical name which will point at the file when the callback returns. All references to the file from that point on must be made via the logical name.

ddnt - A full or partial specification of the file to be located.

default - A full or partial file specification to be used as the default when parsing the previous parameter.

locate - A comma-separated list of single-character codes that specifies how the file is to be located. The items in the list must be chosen from the following, and must appear in the order shown.

W - kit's Working directory. Check the kit's working directory for a file with the matching name and type.

S - as Specified. Check the directory specified by the ddnt and default parameters. If the file is not found there, and this is a small disk system, check the corresponding directory on the library disk, if any.

E - Error. If the file has not yet been found, produce an error message and exit with st&tus VMI$_FAILURE.

O - system-specific root. Use the system-specific root, if installing to a common root, when searching for a matching name and type.

symbol - This optional global symbol will be equated to reflect the results of the callback, as follows.

SYMBOL VALUE	MEANING
"W"	W was included in the locate list, and the file was found in the kit's working directory.
"S"	S was included in the locate list, and the file was found in the specified directory.
"E"	E was included in the locate list, and an error was reported.
""	None of the above.

Notes

If you ever try to reference a system file directly, without going through some callback, your installation procedure will be prone to breakage.

4.9 GENERATE_SDL_DEFINITIONS

This callback is used to generate SDL definitions. The generated files will be located in the kit's working directory with the language specific file extensions.

```
$ VMI$CALLBACK GENERATE_SDL_DEFINITIONS module_file language -
                                        [qualifiers] [options]
```

Parameters

module_file - The name of the module to be extracted from STARLETSD.TLB for processing by SDL. If the module name is preceded by an "@", then module_file is assumed to be a file, found in VMI$KWD: (i.e. shipped on the kit). This file should

contain one module name per line. VMI$KWD:.DAT is used as the default file specification when searching for the module name file.

language - The name of any supported SDL output language. If more than one language output is desired, separate them with a comma, and enclose the list with quotation marks.

qualifiers - This is an optional list of one or more valid SDL qualifiers which will be applied to the SDL command. This allows special processing to be done on the SDL output at the discretion of the layered product. The qualifiers must be enclosed in quotation marks.

options - This is an optional list of single character options, separated by commas. There are none currently defined and is reserved for future expansion.

4.10 GET_SYSTEM_PARAMETER

This callback is used to obtain the current value of a system parameter.

```
$ VMI$CALLBACK GET_SYSTEM_PARAMETER symbol name
```

Parameters

symbol - The name of a global symbol to be equated to the value of the parameter.

name - The full name of the parameter to be obtained.

Notes

There is no way to set the value of a system parameter. You may make recommendations to the system manager, but you cannot set the parameters.

4.11 MESSAGE

This callback is used to display a message in the standard VMS format.

```
$ VMI$CALLBACK MESSAGE severity id text ...
```

Parameters

severity - The severity of the message. Use the standard codes S, I, W, and E. You may not generate a fatal error.

id - The mnemonic identification of the message. This allows cross-referencing in your installation guide.

text - The actual text of the message(s). You can specify up to three message lines, the first of which will be prefixed with a percent sign (%) and the remainder with a hyphen (-).

Notes

It is not necessary to use this callback when displaying large blocks of explanatory text. You should use it when displaying important messages about the status of the installation.

4.12 PATCH_IMAGE

This callback is used to patch an existing native-mode image.

```
$ VMI$CALLBACK PATCH_IMAGE logical patch-nt [image-ddnt] -
                                            [options]
```

Parameters

logical - A logical name which will point at the patched image when the callback returns. All references to the image from that point on must be made via the logical name.

patch-nt - The name and type of the file containing patch commands. All desired patch commands, and only patch commands, must be present in this file. The file must reside in the kit's working directory, and may be deleted after the callback returns in order to save disk space.

image-ddnt - The full specification of the image to be patched.

options - a comma-separated list of options.

- o J - Journal. Create or update a patch journal in the same directory as the image.

- o K - Keep. Do not purge old versions of the image.
- o A - Absolute. Use absolute mode when patching image (PATCH/ABSOLUTE...)

Notes

If the image is not specified in the callback, it is assumed to be specified on the first line of the patch command file. This line consists of an exclamation point and the full image specification. Furthermore, if an option list is present after the specification, it is merged in with the option list specified in the callback.

4.13 PRINT_FILE

This callback is used to queue a file to SYS$PRINT for printing.

```
$ VMI$CALLBACK PRINT_FILE ddnt [copies]
```

Parameters

ddnt - The full specification of the file to be printed.

copies - The number of copies to be printed. The default is one.

4.14 PRODUCT

This callback provides a simple facility for adding product-specific callbacks to VMSINSTAL's callback repertoire. When a set of products form a logical grouping (e.g., VAX Information Architecture, RSX/VAX), there may very well be some additional callbacks which the products in the group would like to share. In this case, the base product in the group (e.g., CDD, RSX) can provide a command procedure containing additional callback logic. The PRODUCT callback is the window through which installation procedures obtain the services of that procedure.

```
$ VMI$CALLBACK PRODUCT procedure:callback parameter ...
```

Parameters

procedure - The name of the procedure which provides the required callback. It is assumed to reside in SYS$UPDATE with a file type of COM.

callback - The name of the desired callback.

parameter - The remainder of the parameters are simply passed on to the product-specific callback.

Notes

The status returned by the PRODUCT callback is the status returned by the product-specific callback. Conventions for coding a product-specific callback procedure are outlined in Appendix C.

4.15 PROVIDE_DCL_COMMAND

This callback is used to add a DCL command to the DCL command tables. If the command already exists, it is replaced.

```
$ VMI$CALLBACK PROVIDE_DCL_COMMAND nt
```

Parameters

nt - The file name and type of the CLD file. It must reside in the kit's working directory, and may be deleted after the callback returns in order to save disk space.

Notes

The command is also added to the active process command tables. You may add more than one command by using the callback multiple times.

4.16 PROVIDE_DCL_HELP

This callback is used to insert help into the DCL help library. If the help already exists, it is replaced.

    $ VMI$CALLBACK PROVIDE_DCL_HELP nt

Parameters

nt - The file name and type of the help text file. It must reside in the kit's working directory, and may be deleted after the callback returns in order to save disk space.

Notes

You may provide more than one help entry by using this callback multiple times. Only a top-level help entry may be inserted.

4.17 PROVIDE_FILE

This callback is used to provide a complete new file as part of the software product. If the file already exists, a new version is created.

    $ VMI$CALLBACK PROVIDE_FILE logical nt dd [options]

Parameters

logical - A logical name which will point at the file when callback returns. All references to the file from that point on must be made via the logical name.

nt - The file name and type of the file being provided. It must reside in the kit's working directory.

dd - The target disk and directory for the file.

options - A comma-separated list of options.

- K - Keep. Do not purge old versions of the file.
- L - Library. Put the file on the library disk if this is a tailored system.
- O - System-specific. Move the file to the system-specific root if installing to a common root.

Notes

no
Do not use this callback to provide an image. See the PROVIDE_IMAGE callback.

4.18 PROVIDE_IMAGE

This callback is used to provide a complete new image as part of the software product. If the image already exists, a new version is created.

```
$ VMI$CALLBACK PROVIDE_IMAGE logical nt dd [options] [eco-list]
```

Parameters

logical - A logical name which will point at the image when callback returns. All references to the image from that point on must be made via the logical name.

nt - The file name and type of the image being provided. It must reside in the kit's working directory.

dd - The target disk and directory for the image.

options - A comma-separated list of options.

- E - ECO list. The following parameter specifies a list of ECO levels which are to be set in the image.
- I - IMAGELIB. Add a sharable image to IMAGELIB.OLB so that it will be automatically searched by the linker.
- K - Keep. Do not purge old versions of the image.
- L - Library. Move the image to the library disk if this is a tailored system.
- O - System-specific. Move the image to the system-specific root if installing to a common root.

eco-list - A comma-separated list of ECO level numbers which are to be set in the image. This parameter is used only if the E option is present.

Notes

If the image was INSTALLed before the installation began, it will be reINSTALLed afterwards. Compatibility-mode images must also be provided with this callback.

4.19 RENAME_FILE

This callback is used to rename a file which was created by a previous installation. The file name and type can be changed, but the file cannot be moved from one directory to another.

```
$VMI$CALLBACK RENAME_FILE ddnt new-nt
```

Parameters

ddnt - The complete specification of the file to be renamed.

new-nt - The new file name and type for the file.

4.20 RESTORE_SAVESET

This callback is used by VMSINSTAL to restore the primary kit saveset. You may also use it to restore savesets other than the primary one.

```
$ VMI$CALLBACK RESTORE_SAVESET saveset [options]
```

Parameters

saveset - The single-letter identification of the saveset to be restored. The entire contents of the saveset are restored into the kit's working directory. If a file to be restored already exists in the directory, an error results.

options - A comma-separated list of options.

- N - Next volume. The saveset begins on the next volume of the distribution volume set. VMSINSTAL will figure that out if you don't specify this option, but with a few funny messages appearing on the console.

Notes

Not all savesets need be restored, but they must be restored in alphabetical order. Restored files will have owner UIC [1,4] and whatever protection they had when saved. This protection should be as specified in section 3.3.1; a future enhancement to VMSINSTAL may force this to be true. You must alter the owner UIC and protection after files are restored.

4.21 SECURE_FILE

This callback is used to alter the security information associated with files. It should only be used by a product for which security is a major feature, such as a hospital application. In most cases, the default system security is fine.

```
$ VMI$CALLBACK SECURE_FILE ddnt [owner-uic] [protection]
```

Parameters

ddnt - The full specification of the file whose security is to be altered.

owner-uic - The new UIC for the files, in the standard format [g,m].

protection - The new protection for the files, in the standard format. Do not enclose the protection string in parentheses.

Notes

You have to be very careful if your installation manipulates files which have special security. Any callback that generates a new version of the file, such as PATCH_IMAGE, will cause the default security to be assigned. You must explicitly set the security back to your special values. The security on VMS system files must never be altered.

4.22 SET IVP

This callback is used to determine whether or not the product's IVP is run after the installation.

```
$ VMI$CALLBACK SET IVP {YES} [options]
                       {NO }
                       {ASK}
```

Parameters

YES - Run the IVP.

NO - There is no IVP supplied with this product. This is the default.

ASK - There is an IVP, but ask the user if it should be run.

options - A comma-separated list of single-character options.

- o H - Help first. When in ASK mode, this option specifies that the help information for the question be displayed before the question is asked.

4.23 SET PURGE

This callback is used to specify whether or not files replaced by this installation are to be purged during or after the installation.

```
$ VMI$CALLBACK SET PURGE {YES} [options]
                         {NO }
                         {ASK}
```

Parameters

YES - Purge all files during or after the installation.

NO - Do not purge file during or after the installation. This is the default.

ASK - Ask the user.

options - A comma-separated list of single-character options.

- o H - Help first. When in ASK mode, this option specifies that the help information for the question be displayed before the question is asked.

Notes

If a file was provided with callbacks that specified the keep (K) option, it is never purged.

4.24 SET REBOOT

This callback is used to specify that a system reboot is necessary after the installation. This callback is reserved for VMS updates and upgrades; layered products should never reboot.

```
$ VMI$CALLBACK SET REBOOT {YES}
                          {NO }
```

Parameters

YES - Reboot after the installation.

NO - No reboot is necessary. This is the default.

4.25 SET SAFETY

This callback is used to specify the safety level of the installation.

```
$ VMI$CALLBACK SET SAFETY {YES         peak}
                          {CONDITIONAL peak}
                          {NO                }
```

Parameters

YES peak - System safety should be optimized at the expense of disk blocks. This option is the default, and produces the highest degree of recovery capability should the system crash no during the installation. Running in safety mode does not guarantee the success of installations performed while other processes are active. The peak parameter specifies the peak block utilization during a safety mode installation. This number mus must be obtained using the statistics option (see Chapter 5).

CONDITIONAL peak - If the specified peak blocks are available, then optimize safety; otherwise do not. This is the option that should be specified if at all possible.

NO - System safety should be ignored in order to save disk space.

4.26 SET STARTUP

This callback is used to specify your product-specific startup procedure so that it can be invoked after the installation and before the IVP is run. This callback does not put the installation procedure in SYS$MANAGER; use a PROVIDE_FILE callback for this purpose. This callback does not tell the system manager about the procedure.

```
$ VMI$CALLBACK SET STARTUP nt
```

Parameters

nt - The file name and type of your product-specific installation procedure. It is assumed to ultimately reside in the SYS$MANAGER directory.

Notes

By isolating all product-specific startup activities in one procedure, you minimize the changes to SYSTARTUP.COM, make the system manager's job easier, and allow a simulation of startup before the IVP is run.

4.27 SUMSLP_TEXT

This callback is used to edit an existing text file or library member with the SUMSLP editor.

```
$ VMI$CALLBACK SUMSLP_TEXT logical command-nt -
        [{ddnt        FILE} old-checksum [new-checksum] [options]]
         {ddnt,member type}
```

Parameters

logical - A logical name which will point at the edited file or library when the callback returns. All references to the file or library from that point on must be made via the logical name.

command-nt - The name and type of the file containing SUMSLP edit commands. All desired commands, and only commands, must be present in this file. The file must reside in the kit's working directory, and may be deleted after the callback returns in order to save disk space.

ddnt FILE - The full specification of the text file to be edited. If you want to edit a library member, use the following format.

ddnt,member type - The full specification of the library containing the member to be edited. Immediately following is a comma and the name of the member. The type parameter specifies the type of the library: HELP, MACRO, TEXT.

old-checksum - The checksum of the file or member prior to editing. This is used by VMSINSTAL to ensure that you are editing what you think you are editing. This checksum is calculated using the Checksum utility, which will be shipped with the new VMSINSTAL. If the checksum fails, VMSINSTAL reports an error and returns unsuccessfully.

new-checksum - The checksum of the file or member after editing. This is used by VMSINSTAL if the old checksum does not match the file or member, so that it can produce any informatory message rather than an error message.

options - A comma-separated list of options.

o K - Keep. Do not purge old versions of the file.

Notes

If the file or library member is not specified in the callback, it is assumed to be specified on the first line of the SUMSLP command file. This line consists of the characters dash, semicolon, exclamation point (-;!) followed by parameters 4, 5, 6, 7 and 8 as described above.

4.28 TELL_QA

There is a Quality Assurance (QA) group at Spit Brook which installs layered product kits on various baselevels of VMS. We hope to catch problems at an early stage, so that the fixes to VMS and/or the layered products can be made in a timely fashion.

The QA group will check many aspects of your product installation. If they find that you are doing something in a nonstandard fashion, they will complain to you. If this nonstandard operation is essential, there is no point in wasting everyone's time with the complaint. This callback is used to bring these essential deviations to the QA folks' attention. When the installation is done in QA mode (see Chapter 5), information specified by TELL_QA is displayed on the terminal. When

run normally, such as at a customer site, the information is not displayed.

```
$ VMI$CALLBACK TELL_QA message
```

Parameters

message - A quoted string to be displayed when installing in QA mode.

Notes

If you have a special problem and consult with a VMS developer, you may be told to include this callback in front of the resulting installation code.

4.29 UPDATE_ACCOUNT

This callback is used to update an account in SYSUAF.DAT. It should be used sparingly, if ever.

```
$ VMI$CALLBACK UPDATE_ACCOUNT username qualifiers
```

Parameters

username - The username that identifies the account.

qualifiers - A sequence of qualifiers as accepted by the MODIFY command of the AUTHORIZE utility. The qualifiers must be enclosed in quotation marks.

Notes

You may only update an account that was originally created by your product. Please also see the notes under CREATE_ACCOUNT.

4.30 UPDATE_FILE

This callback is used to update an existing file. It is only used when updating the file in place, that is, performing an update that modifies an existing copy of the file. First issue the UPDATE_FILE callback, and then reference and modify the file via the logical name.

```
$ VMI$CALLBACK UPDATE_FILE logical ddnt
```

Parameters

logical - A logical name which will point at the file when callback returns. All references to the file from that point on must be made via the logical name.

ddnt - The complete specification of the existing file to be updated. You may not create a new version of the file.

Notes

This callback is not used to update libraries (see UPDATE_LIBRARY). If you need to create a new version of an existing file, and none of the other callbacks are sufficient for your purposes, then create the new version in the working directory. Once the new version is ready, use the PROVIDE_FILE callback to replace the old version.

4.31 UPDATE_LIBRARY

This callback is used to update an existing library.

```
$ VMI$CALLBACK UPDATE_LIBRARY logical lib-ddnt type -
                              qualifiers [file-ddnt]
```

Parameters

logical - A logical name which will point at the library when callback returns. All references to the library from that point on must be made via the logical name.

lib-ddnt - The complete specification of the existing library to be updated.

type - The type of the library: HELP, MACRO, OBJECT, SHARE, TEXT.

qualifiers - A sequence of qualifiers for the librarian. The qualifiers must be enclosed in quotation marks.

file-ddnt - The full specification of the input/output file for the librarian, if required by the qualifiers. This specification may include wildcards. The file may be deleted after the callback returns in order to save disk space.

Notes

It is suggested that you use the /REPLACE operation rather than /INSERT. Be careful not to rely on defaults in the qualifiers.

5 VMSINSTAL OPTIONS

VMSINSTAL accepts various options which can simplify the creation and debugging of your installation procedure. You request these options by appending two or three parameters to the invocation of VMSINSTAL.

```
$ @VMSINSTAL product device OPTIONS list [root]
```

It requires two parameters so you don't do it accidentally. The fourth parameter is a comma-separated list of option letters, as follows.

- A - Auto-answer. This feature allows you to pre-answer the questions asked during the installation of a product and then use the answers at a later date. If the file [SYSUPD]facvvu.ANS exists, VMSINSTAL uses the answers in the file and does not bother asking you the questions. If there is no such file, VMSINSTAL asks you the questions interactively and records the answer in a new file for later use.

- B - invoked during Booting. This option is used when VMSINSTAL is invoked from STARTUP.COM after a crash during installation.

- C - Callback trace. VMSINSTAL traces all callbacks and lists them in the file [SYSUPD]facvvu.CBT.

- D - Debug mode. This option is used in the debugging of VMSINSTAL itself. Do not use it when debugging your installation procedure.

- G - Get savesets. This option is used to copy the savesets comprising a kit into a disk directory for later installation. All kit savesets are copied, but no installation is performed. Parameter 5 can specify the disk and directory into which the savesets are to be copied.

- K - Kit debug mode. VMSINSTAL passes a boolean parameter to the kit's installation procedure when requesting that it do an installation. This parameter reflects the presence or absence of the K option. The kit's installation procedure can then modify its behavior as desired.

- L - file Log. VMSINSTAL defines special symbols for commands so that all file activity is logged to the terminal. This is why command verbs cannot be abbreviated. Normally, file activity is not logged.

- Q - QA mode. This option specifies that the installation should be done in QA mode. Any TELL_QA callbacks performed by the kit's installation procedure are logged to the terminal.

- R - alternate Root. The product is installed in a system root other than that of the running system. Parameter 5 must specify the root in the format "_ddcu:[SYSn.]". This alternate root must contain a full complement of VMS software. The VMS system in the alternate root must be at the same version/update level as the running system. If the installation procedure references any other layered products, the versions of the products on the running system will be used.

- S - Statistics report. VMSINSTAL produces a statistics report in the file [SYSUPD]facvvu.ANL. This report contains a description of the hardware and software on which the installation was performed, disk usage statistics, and a list of files added, deleted, modified, and accessed by the installation. Correct disk usage statistics require that the system disk contain a full complement of VMS software and enough free blocks for an installation in safety mode. Note that the statistics pertain only to the system disk, so that the library disk on a small disk system is not included.

$!
$!
$ IF
$ EX
$!
$!
$!
$!
$!
$!
$!
$ ER
$!
$ S
$ IF
$ EX

A SAMPLE INSTALLATION PROCEDURE

The following is the installation procedure for FMS. This procedure is a very good sample for analysis by would-be installation procedure implementers.

```
$ !                       K I T I N S T A L . C O M
$ !
$ !                          COPYRIGHT (C) 1983 BY
$ !                    DIGITAL EQUIPMENT CORPORATION, MAYNARD
$ !                     MASSACHUSETTS.  ALL RIGHTS RESERVED.
$ !
$ !   THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
$ !   ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE INCLUSION
$ !   OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR ANY OTHER COPIES
$ !   THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER
$ !   PERSON.  NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
$ !
$ !   THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE AND
$ !   SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
$ !
$ !   DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
$ !   SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
$ !
$ !*****************************************************************************
$ !
$ !       This procedure installs VAXFMS V2.1 on VMS using VMSINSTAL
$ !
$ !       Setup error handling
$ !
$ ON CONTROL_Y THEN VMI$CALLBACK CONTROL_Y
$ ON WARNING THEN GOTO ERR_EXIT
$ !
$ !
$ !       Handle INSTALL, IVP and unsupported parameters passed by VMSINSTAL
$ !
$ IF P1 .EQS. "VMI$_INSTALL" THEN GOTO INSTALL
$ IF P1 .EQS. "VMI$_IVP" THEN GOTO IVP
$ EXIT VMI$_UNSUPPORTED
$ !
$ INSTALL:
$ !
$ !       Let VMSINSTAL know we have an IVP which must be executed,
$ !       and that all replaced files are to be purged.
$ !
$ VMI$CALLBACK SET IVP YES
$ VMI$CALLBACK SET PURGE YES
$ !
$ !       Check for valid VMS version
$ !       Will install to any version 3.2 or later.
$ !
```

```
$ VMI$CALLBACK CHECK_VMS_VERSION VAXFMS$VMS 032
$ IF .NOT. VAXFMS$VMS THEN VMI$CALLBACK MESSAGE E BADVMS -
        "This kit requires Version 3.2 or a subsequent version of VMS"
$ IF .NOT. VAXFMS$VMS THEN EXIT VMI$_FAILURE
$ !
$ !     Check for enough free blocks on system disk.
$ !     Need a minimum of 5000.
$ !
$ VMI$CALLBACK CHECK_NET_UTILIZATION VAXFMS$ 5000
$ IF .NOT. VAXFMS$ THEN VMI$CALLBACK MESSAGE E NOSPACE -
        "System disk does not contain enough free blocks to install FMS"
$ IF .NOT. VAXFMS$ THEN EXIT VMI$_FAILURE
$ !
$ !     Check for V2.0 FMS
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]FMSFED.EXE -
        "" S VAXFMS$V2FED
$ IF VAXFMS$V2FED .EQS. "S" THEN GOTO 20$               !V2.0 FED IS THERE
$ !
$ !     Check for V1 FMS
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]FED.EXE -
        "" S VAXFMS$V1FED
$ IF VAXFMS$V1FED .EQS. "S" THEN GOTO 10$               !V1 FED IS THERE
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]FUT.EXE -
        "" S VAXFMS$V1FUT
$ IF VAXFMS$V1FUT .EQS. "S" THEN GOTO 10$               !V1 FUT IS THERE
$ GOTO 20$
$ !
$ !     Clean up parts of Version 1
$ !
$ !     Remove the old FDV object modules from STARLET.OLB.
$ !
$ 10$:
$ VMI$CALLBACK UPDATE_LIBRARY VAXFMS$STARLET VMI$ROOT:[SYSLIB]STARLET.OLB -
        OBJECT "/DELETE=(FDV,FDVMSG,FDVDAT,FDVERR,FDVTIO,FDVXFR,HLL,HLLDFN)"
$ !
$ !
$ !     Save FMS V1 FDVSHARE.OPT and FDVSHR.EXE by renaming it to *.OLD
$ !
$ VMI$CALLBACK  FIND_FILE VAXFMS$ VMI$ROOT:[SYSLIB]FDVSHARE.OPT -
        "" S VAXFMS$SHARESTAT
$ IF VAXFMS$SHARESTAT .NES. "S" THEN GOTO 15$
$ BACKUP VMI$ROOT:[SYSLIB]FDVSHARE.OPT VMI$KWD:FDVSHARE.OLD/OWNER=ORIGINAL
$ VMI$CALLBACK DELETE_FILE VMI$ROOT:[SYSLIB]FDVSHARE.OPT
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVSHARE.OLD VMI$ROOT:[SYSLIB]
$ !
$ 15$:
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSLIB]FDVSHR.EXE -
        "" S VAXFMS$SHRSTAT
$ IF VAXFMS$SHRSTAT .NES. "S" THEN GOTO 20$
$ BACKUP VMI$ROOT:[SYSLIB]FDVSHR.EXE VMI$KWD:FDVSHR.OLD/OWNER=ORIGINAL
```

```
$ VMI$CALLBACK DELETE_FILE VMI$ROOT:[SYSLIB]FDVSHR.EXE
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVSHR.OLD VMI$ROOT:[SYSLIB]
$ !
$ !
$ 20$:
$ !
$ !     Restore saveset B and go.
$ !
$ VMI$CALLBACK RESTORE_SAVESET B
$ !
$ !     Link and supply to system the FMS and FDV Message files
$ !
$ LINK /SHAREABLE=VMI$KWD:FMSMSG         VMI$KWD:FMSMSG
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FMSMSG.EXE VMI$ROOT:[SYSMSG]
$ LINK /SHAREABLE=VMI$KWD:FDVMSG         VMI$KWD:FDVMSG
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FDVMSG.EXE VMI$ROOT:[SYSMSG] I
$ !
$ !
$ !     Link Form Driver and FIO as shared Library.
$ !
$ !  Build the image.  NOTE THAT THE GSMATCH NUMBER MUST BE UPDATED
$ !     FOR EACH RELEASE AND POINT RELEASE!
$ !
$ LINK /SHAREABLE = VMI$KWD:FDVSHR -
                /NOMAP -
                VMI$KWD:TIOFMSMSG.OPT/OPTIONS, -
                SYS$INPUT:/OPTIONS
CLUSTER = FDV_TRANSFER,,,VMI$KWD:FDVLIB/INCLUDE=FDV$XFR
CLUSTER = FDV_FIO,,,VMI$KWD:FIOLIB/LIB
CLUSTER = FDV_TIO,,,VMI$KWD:FDVLIB/INCLUDE = ( FDV$TIOTSK, -
                                               FDV$TIOGET, -
                                               FDV$TIOSI2, -
                                               FDV$TIORDV, -
                                               FDV$TIOCOM )
CLUSTER = FDV_CODE,,,VMI$KWD:FDVLIB/LIBRARY
GSMATCH = LEQUAL,2,101
$ !
$ !  Insert 1 module to STARLET
$ !
$ VMI$CALLBACK UPDATE_LIBRARY VAXFMS$STARLET VMI$ROOT:[SYSLIB]STARLET.OLB -
        OBJECT "/REPLACE" VMI$KWD:FDVPLITRM.OBJ
$ !
$ !  Put V2 FDVSHR in the System Library and install in IMAGELIB
$ !
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FDVSHR.EXE VMI$ROOT:[SYSLIB] I
$ !
$ !
$ !     Form Upgrade Utility
$ !
$ LINK /EXECUTABLE=VMI$KWD:FMSFUU -
        VMI$KWD:FUULIB/LIBRARY/INCLUDE=FUU$TOP,-
        VMI$KWD:FV1,-
```


The

```
        VMI$KWD:FMSPTR,-
        SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ !
$ !     Forms Application Aids
$ !
$ LINK /EXECUTABLE=VMI$KWD:FMSFAA -
        VMI$KWD:FAALIB/LIBRARY/INCLUDE=FAA$MAIN,-
        VMI$KWD:FMSPTR,-
        SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ !
$ !     Form Librarian
$ !
$ LINK /EXECUTABLE=VMI$KWD:FMSFLI -
        VMI$KWD:FLILIB/LIBRARY/INCLUDE=FLI$DRIVER,-
        VMI$KWD:FMSPTR,-
        SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ !
$ !     Form Language
$ !
$ LINK /EXECUTABLE=VMI$KWD:FMSFLG -
        VMI$KWD:FLGLIB/LIBRARY/INCLUDE=DRIVER,-
        VMI$KWD:FMSPTR,-
        SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ !
$ !     Form Tester
$ !
$ LINK /EXECUTABLE=VMI$KWD:FMSFTE -
        VMI$KWD:FTELIB/LIBRARY/INCLUDE=FTE,-
        VMI$KWD:FMSPTR,-
        SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ !
$ !     Form Editor
$ !
$ LINK /EXECUTABLE=VMI$KWD:FMSFED -
        VMI$KWD:FEDLIB/INCLUDE=FED$FED/LIBRARY,-
        VMI$KWD:FEDLIB/INCLUDE=FDV$VECTOR/LIBRARY,-
        VMI$KWD:FEDLIB/INCLUDE=FDV$MEMRES/LIBRARY,-
        VMI$KWD:FMSPTR,-
        SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ !
$ !     Add FMS and qualifiers to DCL
$ !
$ VMI$CALLBACK PROVIDE_DCL_COMMAND FMSDCL.CLD
$ !
$ !     Now update the system help library
$ !
```

```
$ VMI$CALLBACK PROVIDE_DCL_HELP FMS.HLP
$ !
$ !     Now put everything in its place
$ !
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FMSFUU.EXE VMI$ROOT:[SYSEXE]
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FMSFED.EXE VMI$ROOT:[SYSEXE]
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FMSFAA.EXE VMI$ROOT:[SYSEXE]
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FMSFTE.EXE VMI$ROOT:[SYSEXE]
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FMSFLG.EXE VMI$ROOT:[SYSEXE]
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ FMSFLI.EXE VMI$ROOT:[SYSEXE]
$ VMI$CALLBACK PROVIDE_FILE  VAXFMS$ HLL11.OBJ  VMI$ROOT:[SYSLIB]
$ !
$ !     Give System Manager an FMS Startup procedure
$ !
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ -
        FMSTARTUP.COM   VMI$ROOT:[SYSMGR] K
$ !
$ !     Tell VMSINSTALL about FMSTARTUP
$ !
$ VMI$CALLBACK SET STARTUP FMSTARTUP.COM
$ !
$ TYPE SYS$INPUT

        VAX-11 FMS utilities have built successfully.
        Continuing installation...

$ !
$ !     Provide the FMS Sample Application Programs
$ !
$ ! Create [SYSHLP.EXAMPLES.FMS] (check to see if it is there, first)
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSHLP.EXAMPLES]FMS.DIR -
        "" S VAXFMS$EXAMPSTAT
$ IF VAXFMS$EXAMPSTAT .EQS. "S" THEN GOTO 30$   !THEN IT'S THERE
$ !
$ VMI$CALLBACK CREATE_DIRECTORY SYSTEM SYSHLP.EXAMPLES.FMS
$ !
$ 30$:
$ !
$ !  Create SMPVECTOR and SMPMEMRES - Need to tell system that FMSFAA
$ !  and FDVSHR are temporarily in VMI$KWD in order to use the DCL
$ !  commands necessary.
$ !
$ DEFINE FDVSHR VMI$KWD:FDVSHR.EXE
$ DEFINE FMSFAA VMI$KWD:FMSFAA.EXE
$ !
$ FMS/VECTOR/OUTPUT=VMI$KWD:SMPVECTOR -
        VMI$KWD:SAMP.FLB
$ FMS/MEMORY/OUTPUT=VMI$KWD:SMPMEMRES -
        VMI$KWD:SAMP.FLB/FORM=(HELP_KEYS,HELP_MENU)
$ !
$ DEASSIGN FDVSHR
```

```
$ DEASSIGN FMSFAA
$ !
$ ! Compile and link SAMP in each language.
$ !
$ !              ***BASIC***
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]BASIC.EXE -
        "" S VAXFMS$BASSTAT
$ IF VAXFMS$BASSTAT .NES. "S" THEN GOTO 50$
$ SET NOON
$ BASIC/OBJECT=VMI$KWD:SAMP.OBJ VMI$KWD:SAMP.BAS
$ IF .NOT. $STATUS THEN GOTO 40$
$ SET ON
$ GOTO 50$
$ !
$ 40$:
$ SET ON
$ TYPE SYS$INPUT

        The BASIC version of the VAX-11 FMS Sample Application failed
        to compile. See the VAX-11 FMS Installation Guide and Release
        Notes for error recovery.

        The installation procedure is continuing...

$ !
$ !
$ 50$:
$ SET NOON
$ LINK/EXECUTABLE=VMI$KWD:SAMP.EXE VMI$KWD:SAMP.OBJ,-
        VMI$KWD:SMPVECTOR.OBJ,-
        VMI$KWD:SMPMEMRES.OBJ,-
        SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ IF .NOT. $STATUS THEN GOTO 70$
$ SET ON
$ !
$ 60$:
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ SAMP.EXE VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ 70$:
$ SET ON
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$     SAMP.BAS        VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$     SAMPBAS.COM     VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$     FDVDEF.BAS      VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ !              ***BLISS***
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]BLISS32.EXE -
        "" S VAXFMS$BLISTAT
$ IF VAXFMS$BLISTAT .NES. "S" THEN GOTO 90$
$ !
```

```
$ DEFINE FDVDEF VMI$KWD:FDVDEF.REQ
$ SET NOON
$ BLISS/OBJECT=VMI$KWD:SAMPBLI.OBJ  VMI$KWD:SAMPBLI.BLI
$ IF .NOT. $STATUS THEN GOTO 80$
$ DEASSIGN FDVDEF
$ LINK/EXECUTABLE=VMI$KWD:SAMPBLI.EXE VMI$KWD:SAMPBLI.OBJ, -
        VMI$KWD:SMPVECTOR.OBJ, VMI$KWD:SMPMEMRES.OBJ, SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ IF .NOT. $STATUS THEN GOTO 80$
$ SET ON
$ !
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ SAMPBLI.EXE VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ GOTO 90$
$ !
$ 80$:
$ SET ON
$ TYPE SYS$INPUT

        The BLISS version of the VAX-11 FMS Sample Application failed
        to compile or link. See the VAX-11 FMS Installation Guide and
        Release Notes for error recovery.

        The installation procedure is continuing...

$ !
$ 90$:
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPBLI.BLI VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPBLI.COM VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVDEF.REQ  VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ !             ***FORTRAN***
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]FORTRAN.EXE -
        "" S VAXFMS$FORSTAT
$ IF VAXFMS$FORSTAT .NES. "S" THEN GOTO 110$
$ !
$ SET NOON
$ LIBRARY/CREATE/TEXT VMI$KWD:SMPFORTXT -
                VMI$KWD:SMPACCOM.FOR  /MODULE=ACCOUNT_COMMON,-
                VMI$KWD:SMPREGCOM.FOR /MODULE=REGISTER_COMMON,-
                VMI$KWD:SMPSTATUS.FOR /MODULE=STATUS_AREA,-
                VMI$KWD:SMPWORK.FOR   /MODULE=WORK_AREA
$ IF .NOT. $STATUS THEN GOTO 100$
$ DEFINE SMPFORTXT VMI$KWD:SMPFORTXT.TLB
$ DEFINE FDVDEF VMI$KWD:FDVDEF.FOR
$ FORTRAN/OBJECT=VMI$KWD:SAMPFOR.OBJ    VMI$KWD:SAMPFOR.FOR
$ IF .NOT. $STATUS THEN GOTO 100$
$ DEASSIGN SMPFORTXT
$ DEASSIGN FDVDEF
$ LINK/EXECUTABLE=VMI$KWD:SAMPFOR.EXE   VMI$KWD:SAMPFOR.OBJ, -
        VMI$KWD:SMPVECTOR.OBJ, VMI$KWD:SMPMEMRES.OBJ, SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
```

```
$ IF .NOT. $STATUS THEN GOTO 100$
$ SET ON
$ !
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ SAMPFOR.EXE VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ GOTO 110$
$ !
$ 100$:
$ !
$ SET ON
$ TYPE SYS$INPUT

        The FORTRAN version of the VAX-11 FMS Sample Application failed
        to compile or link. See the VAX-11 FMS Installation Guide and
        Release Notes for error recovery.

        The installation procedure is continuing...

$ !
$ 110$:
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPFOR.FOR VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPFOR.COM VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVDEF.FOR  VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SMPACCOM.FOR VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SMPREGCOM.FOR VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SMPSTATUS.FOR VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SMPWORK.FOR VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ !               ***PASCAL***
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]PASCAL.EXE -
        "" S VAXFMS$PASSTAT
$ IF VAXFMS$PASSTAT .NES. "S" THEN GOTO 130$
$ SET NOON
$ PASCAL/ENVIRONMENT=VMI$KWD:/OBJECT=VMI$KWD:  VMI$KWD:FDVDEF.PAS
$ IF .NOT. $STATUS THEN GOTO 120$
$ DEFINE FDVDEF VMI$KWD:FDVDEF.PEN
$ PASCAL/NOENVIRONMENT/OBJECT=VMI$KWD:SAMPPAS.OBJ  VMI$KWD:SAMPPAS.PAS
$ IF .NOT. $STATUS THEN GOTO 120$
$ DEASSIGN FDVDEF
$ DEFINE FDVDEF VMI$KWD:FDVDEF.OBJ
$ LINK/EXECUTABLE=VMI$KWD:SAMPPAS.EXE VMI$KWD:SAMPPAS.OBJ, VMI$KWD:FDVDEF.OBJ, -
                  VMI$KWD:SMPVECTOR.OBJ, VMI$KWD:SMPMEMRES.OBJ, SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ DEASSIGN FDVDEF
$ IF .NOT. $STATUS THEN GOTO 120$
$ SET ON
$ !
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ SAMPPAS.EXE VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ GOTO 130$
$ !
$ 120$:
$ SET ON
```



```
EXE$

Modu
MARB
XPOR
XCLO
XFAI
XMEM
XGET
XZAP
XPMS
XOPE
XPUT
XVMS
XTER
XVAL
SAPP
SCOP
STRE
STEM
SMSG
XROO
SSCA
SVAL
SFAI
XPAR
SBIN
LIB$
LIB$
LIB$
LIB$
LIB$
STR$
STR$
STR$
SYS$
LIB$
```

```
$ TYPE SYS$INPUT

        The PASCAL version of the VAX-11 FMS Sample Application failed
        to compile or link. See the VAX-11 FMS Installation Guide and
        Release Notes for error recovery.

        The installation procedure is continuing...

$ !
$ !
$ 130$:
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPPAS.PAS VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPPAS.COM VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVDEF.PAS  VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ !               ***C***
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]VAX11C.EXE -
        "" S VAXFMS$CSTAT
$ IF VAXFMS$CSTAT .NES. "S" THEN GOTO 150$
$ !
$ SET NOON
$ DEFINE FDVDEF VMI$KWD:FDVDEF.H
$ CC/OBJECT=VMI$KWD:SAMPCC.OBJ  VMI$KWD:SAMPCC.C
$ IF .NOT. $STATUS THEN GOTO 140$
$ DEASSIGN FDVDEF
$ LINK/EXECUTABLE=VMI$KWD:SAMPCC.EXE VMI$KWD:SAMPCC.OBJ, -
        VMI$KWD:SMPVECTOR.OBJ, VMI$KWD:SMPMEMRES.OBJ, -
        VMI$ROOT:[SYSLIB]CRTLIB/LIBRARY, SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ IF .NOT. $STATUS THEN GOTO 140$
$ SET ON
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ SAMPCC.EXE VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ GOTO 150$
$ !
$ 140$:
$ SET ON
$ TYPE SYS$INPUT

        The C version of the VAX-11 FMS Sample Application failed
        to compile or link. See the VAX-11 FMS Installation Guide and
        Release Notes for error recovery.

        The installation procedure is continuing...

$ !
$ 150$:
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPCC.C    VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPCC.COM  VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVDEF.H    VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ !               ***COBOL***
```

```
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]COBOL.EXE -
        "" S VAXFMS$COBSTAT
$ IF VAXFMS$COBSTAT .NES. "S" THEN GOTO 170$
$ !
$ SET NOON
$ DEFINE FDVDEF VMI$KWD:FDVDEF.LIB
$ DEFINE SAMPCOB VMI$KWD:SAMPCOB.LIB
$ DEFINE SMPCOBUAR VMI$KWD:SMPCOBUAR.LIB
$ COBOL/OBJECT=VMI$KWD:SAMPCOB.OBJ  VMI$KWD:SAMPCOB.COB
$ IF .NOT. $STATUS THEN GOTO 160$
$ DEASSIGN FDVDEF
$ DEASSIGN SAMPCOB
$ DEASSIGN SMPCOBUAR
$ LINK/EXECUTABLE=VMI$KWD:SAMPCOB.EXE VMI$KWD:SAMPCOB.OBJ, -
        VMI$KWD:SMPVECTOR.OBJ, VMI$KWD:SMPMEMRES.OBJ, SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ IF .NOT. $STATUS THEN GOTO 160$
$ SET ON
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ SAMPCOB.EXE VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ GOTO 170$
$ !
$ 160$:
$ SET ON
$ TYPE SYS$INPUT

        The COBOL version of the VAX-11 FMS Sample Application failed
        to compile or link. See the VAX-11 FMS Installation Guide and
        Release Notes for error recovery.

        The installation procedure is continuing...

$ !
$ 170$:
$ !
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPCOB.COB VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPCOB.COM VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVDEF.LIB  VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPCOB.LIB VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SMPCOBUAR.LIB VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ !              ***PL/1***
$ !
$ VMI$CALLBACK FIND_FILE VAXFMS$ VMI$ROOT:[SYSEXE]PLIG.EXE -
        "" S VAXFMS$PLISTAT
$ IF VAXFMS$PLISTAT .NES. "S" THEN GOTO 190$
$ !
$ SET NOON
$ DEFINE FDVDEFCAL VMI$KWD:FDVDEFCAL.PLI
$ PLI/OBJECT=VMI$KWD:SAMPPLI.OBJ VMI$KWD:SAMPPLI.PLI
$ IF .NOT. $STATUS THEN GOTO 180$
$ DEASSIGN FDVDEFCAL
```

```
$ LINK/EXECUTABLE=VMI$KWD:SAMPPLI.EXE  VMI$KWD:SAMPPLI.OBJ, -
        VMI$KWD:SMPVECTOR.OBJ, VMI$KWD:SMPMEMRES.OBJ, SYS$INPUT/OPTION
VMI$KWD:FDVSHR/SHAREABLE
$ IF .NOT. $STATUS THEN GOTO 180$
$ SET ON
$ !
$ VMI$CALLBACK PROVIDE_IMAGE VAXFMS$ SAMPPLI.EXE VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ GOTO 190$
$ !
$ 180$:
$ SET ON
$ TYPE SYS$INPUT

        The PL/1 version of the VAX-11 FMS Sample Application failed
        to compile or link. See the VAX-11 FMS Installation Guide and
        Release Notes for error recovery.

        The installation procedure is continuing...

$ !
$ 190$:
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPPLI.PLI VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMPPLI.COM VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVDEFCAL.PLI VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ FDVDEFFNC.PLI VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ !  Move over form library, data file, SMPVECTOR and SMPMEMRES
$ !
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SMPVECTOR.OBJ VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SMPMEMRES.OBJ VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMP.FLB      VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ VMI$CALLBACK PROVIDE_FILE VAXFMS$ SAMP.DAT      VMI$ROOT:[SYSHLP.EXAMPLES.FMS]
$ !
$ !     Tell manager about installing the shared library
$ !
$ TYPE SYS$INPUT

  System Manager:

        Upon completion of this installation, please be sure to edit
        the system startup files as described in the VAX-11 FMS
        Installation Guide and Release Notes.

$ !
$ IF F$VERIFY() THEN SET NOVERIFY
$ EXIT VMI$_SUCCESS
$ !
$ !                   End of VAXFMS installation
$ !
$ IVP:
$ !
$ !  Set up error handling
```

```
$ !
$ ON WARNING THEN EXIT VMI$_FAILURE
$ ON CONTROL_Y THEN EXIT VMI$_FAILURE
$ !
$ TYPE SYS$INPUT

  Beginning the VAX-11 FMS Installation Verification Procedure.

$ !
$ !  Make a copy of our master library
$ !
$ FMS/LIBRARY/CREATE/NOLOG IVP.FLB FMS$EXAMPLES:SAMP.FLB
$ !
$ !  Two commands will get an 'Information' message.
$ !
$ TYPE SYS$INPUT

  Please ignore the following informational messages:

$ !
$ !  Try extracting a form from a library
$ !
$ FMS/LIBRARY/EXTRACT/NOLOG/OUTPUT=ACTDAT       IVP/FORM=ACCOUNT_DATA
$ !
$ !  Now do a directory of the library
$ !
$ FMS/DIRECTORY/OUTPUT=NL:              IVP.FLB
$ !
$ !  Now do a back translate
$ !
$ FMS/DESCRIPTION/FULL/OUTPUT=CHECK     IVP/FORM=CHECK
$ !
$ !  How about a memory resident module of forms?
$ !
$ FMS/MEMORY_RESIDENT/OUTPUT=MEMORY     ACTDAT.FRM
$ !
$ !  Did it produce a valid obj file?
$ !
$ ANALYZE/OBJECT/OUTPUT=NL:             MEMORY
$ !
$ !  How about a UAR vector module?
$ !
$ FMS/VECTOR/OUTPUT=VECTOR      ACTDAT.FRM
$ !
$ !  Did it produce a valid obj file?
$ !
$ ANALYZE/OBJECT/OUTPUT=NL:             VECTOR
$ !
$ !  Now try the /delete function
$ !
$ FMS/LIBRARY/DELETE    IVP.FLB/FORM=CHECK
$ !
```

```
$ ! Successful test
$ !
$ IF F$VERIFY() THEN SET NOVERIFY
$ EXIT VMI$_SUCCESS
$ !
$ !      End of VAXFMS IVP
$ !
$ !      End of VAX FMS KITINSTAL.COM
$ !
$ ERR_EXIT:
$ !
$ S = $STATUS
$ IF F$VERIFY() THEN SET NOVERIFY
$ EXIT S
```



```
_$2

Syml
---
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
STR
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
XPO

XPO
XPO
XPO
XPO
XPO
XPO
XPO
```

B CRASH RECOVERY

During an installation, VMSINSTAL attempts to record enough state information so that it can recover from a system crash. This information is recorded in the file SYS$UPDATE:VMIMARKER.DAT, which is created when installation of a product is begun, and deleted when it is finished. If, during a system boot, STARTUP.COM notices that VMIMARKER.DAT exists, it invokes VMSINSTAL with the B option. VMSINSTAL inspects the state at the time the crash occurred, and recovers according to the step in progress.

The steps in the following paragraphs refer to the overview in section 3.1.

- Steps 5, 6, 7 - The user is told to simply start again.

- Step 8 - If the installation was done in safety mode, then there are two possibilities. If a library was being updated at the time of the crash, the user is told to restore it from backup and start again. If something else was happening, the user can simply start again.

  If the installation was done in unsafe mode, then the user is told to restore the system disk from backup and start again. There is no way to tell what was happening when the system crashed.

- Step 9 - The deferred callbacks are reexecuted. This should complete the installation satisfactorily.

- Steps 10, 11 - The user is told that the installation was completed satisfactorily.

The documentation for VMSINSTAL will have a thorough description of crash recovery.

C PRODUCT-SPECIFIC CALLBACK PROCEDURES

This appendix describes the conventions for coding a product-specific callback procedure suitable for use with the VMSINSTAL PRODUCT callback. The easiest way to present the conventions is with an example. Let's say that FOOBAR, the base product for a product group, is going to provide a callback procedure. The procedure is named FOOBARINS.COM, because the file name should be the product mnemonic followed by as much of the word "INSTALL" as possible. This procedure is placed in SYS$UPDATE during the installation of FOOBAR.

When another product wants to perform a callback to the procedure, it includes the following line in its installation procedure.

```
$ VMI$CALLBACK PRODUCT FOOBARINS:INCREMENT -
        PROD$INC COUNTFILE.DAT VMI$ROOT:[SYSUPD] 2
```

and FOOBARINS receives parameters as follows:

```
P1      "INCREMENT"
P2      "PROD$INC"
P3      "COUNTFILE.DAT"
P4      "VMI$ROOT:[SYSUPD]
P5      "2"
P6-P8   ""
```

The INCREMENT callback is a phony callback for purposes of illustration. The parameter order for callbacks should follow the those of standard callbacks as closely as possible. The best idea is pick a standard callback that is similar to the one you are designing and mimic its parameter order.

The following procedure is a simple example of a product-specific callback procedure.

```
$!      First thing to do is set up a CTRL/Y handler
$!      and an error handler.  We don't want to trap
$!      warnings because they can happen legitimately.
$
$       ON CONTROL_Y THEN VMI$CALLBACK CONTROL_Y
$       ON ERROR THEN EXIT $STATUS
$
$!      Now we can case on the callback code.
$
$       GOTO 'P1
$
$!      INCREMENT logical name-type directory integer
$!
$!      This callback will increment the number stored in
$!      the file by the specified integer.  A new file
$!      will be created and put back in the original
$!      place, with the logical name defined to point at it.
```

```
$
$INCREMENT:
$
$!      Begin by finding the file with the number to be
$!      incremented.  If the find fails, then return a
$!      status to inform the caller.
$
$       VMI$CALLBACK FIND_FILE 'P2 'P3 'P4 S,E
$       IF .NOT. $STATUS THEN EXIT $STATUS
$
$!      Read the record in the file, which contains the number
$!      to be incremented.
$
$       OPEN/READ VMI$PRODUCT_FILE 'P2
$       READ VMI$PRODUCT_FILE NUMBER
$       CLOSE VMI$PRODUCT_FILE
$
$!      Create a new version of the file in the working directory,
$!      and put the incremented number in it.
$
$       OPEN/WRITE VMI$PRODUCT_FILE VMI$KWD:'P3
$       WRITE VMI$PRODUCT_FILE 'F$INT(F$INT(NUMBER) + F$INT(P5))
$       CLOSE VMI$PRODUCT_FILE
$
$!      Provide the new file, which will replace the old one.
$!      Also define the logical name to point at the final file.
$
$       VMI$CALLBACK PROVIDE_FILE 'P2 'P3 'P4
$       EXIT $STATUS
```

The following conventions must be followed.

- The procedure must establish a CTRL/Y handler that (eventually) invokes the CONTROL_Y callback.

- The procedure must establish an error handler that (eventually) exits with the status that caused the handler to be invoked. Warnings are not trapped, because they are routinely returned from other callbacks.

- The first parameter to the procedure is the callback request code. Case on this parameter with a GOTO.

- The code to implement a callback must follow all of the conventions outlined elsewhere in this manual. In particular, files must be referenced with standard callbacks. The FIND_FILE callback can be used to determine the existence and location of a file. Logical names and global symbols must begin with VMI$, the VMSINSTAL facility name, because the callback procedure is a logical extension of VMSINSTAL.

- The return status from a standard callback must be checked with an IF statement to determine success or failure. Because failure is a warning, the error handler will not be invoked.

- The callback must return with either VMI$_SUCCESS or VMI$_FAILURE status, which is propagated back to the installation procedure.

D EXAMPLE FOR DECODING VMI$VMS_VERSION

The following example shows some techniques and methods for decoding the VMI$VMS_VERSION symbol. While this example only checks for a specific version, the techniques shown here can be easily adapted to check for a range of versions.

```
$! uncomment one of the definitions of product$type
$! uncomment one of the definitions of product$version
$!      product$type = "RELEASED"       ! if only on released VMS software
$!      product$version = "040"         ! VMS 4.0
$!      product$type = "UPGRADE"        ! if only runs on major vers. fieldtest
$!      product$version = "X2M9"        ! v4ft2, for instance
$!      product$type = "UPDATE"         ! if only runs on vms .n maint. update
$!                                      !    fieldtest (X4.1 for example)
$!      product$version = "041"         ! X4.1 for instance
$
$       vmi_type = f$element(0,"","",vmi$vms_version)
$       vmi_type = f$element(0,"FT",vmi_type)           !Ignore the " FT" if there
$       vmi_version = f$element(1,"",vmi$vms_version)  !Get version string
$       goto v_'product$type'
$v_released:
$       if product$type .nes. vmi_type then goto vrl20
$! Check the major and minor versions separately
$       if f$extr(0,2,product$version) .nes. f$extr(0,2,vmi_version) then goto vrl20
$       if f$extr(2,1,product$version) .les. f$extr(2,1,vmi_version) then goto v_ok
$vrl20:
$       vmi$callback MESSAGE e version -
                "This kit must be installed on an existing VMS ''product$version system."
$       exit vmi$_failure
$v_update:
$       if (product$type .nes. vmi_type) then goto vupd20
$ ! It must be the fieldtest of the version we expect
$       if f$extr(0,3,product$version) .eqs. f$extr(0,3,vmi_version) then goto v_ok
$vupd20:
$       vmi$callback MESSAGE e fieldtest -
        "This kit can only be installed on field test version ''product$version'."
$       exit vmi$_failure
$v_upgrade:
$       if (product$type .nes. vmi_type) then goto vupg20
$ ! The last 3 characters of the version must match.  The first letter
$ ! of the version on the system must be .ge. the one we expect.
$       if f$extr(1,3,product$version) .nes. f$extr(1,3,vmi_version) then goto vupg20
$       if f$extr(0,1,product$version) .les. f$extr(0,1,vmi_version) then goto v_ok
$vupg20:
$       vmi$callback MESSAGE e fieldtest -
        "This kit can only be installed on VMS version ''product$version system."
$       exit vmi$_failure
$v_ok:
```

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

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

AUTOGEN
LIS

NETCONFIG
LIS

STARTUP
LIS

VMSINSTAL
MEM

SWAPFILES
LIS

TRANSERR
LIS

MAKEROOT
LIS

SHUTDOWN
LIS

SPKITBLD
LIS

VMSINSTAL
LIS

VMSUPDATE
LIS

0233 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

MELDR

MDL32

LALOAD
MAP

MDL32
MAP

XFLOADER
MAP

LALOADER
LIS

XFLOADER
LIS

LADAMCODE
LIS

LALOAD
LIS

MARBLI

MARBLI
MAP

LALOADER
MAP

LAAMCODE
LIS

LAMRMCODE
LIS

MARBLI
LIS

LADMOT
LIS