OpenVMS Developer's Guide to VMSINSTAL

Order Number: AA–PWBXA–TE

May 1993

This manual describes the VMSINSTAL command procedure and gives developers information about using VMSINSTAL to design their own installation procedures. Digital recommends that all installation procedures for products that layer on the OpenVMS operating system use VMSINSTAL. The *OpenVMS Developer's Guide to VMSINSTAL* is available as an optional OpenVMS manual; it is not included in the standard OpenVMS documentation set.

Revision/Update Information: This document supersedes the *OpenVMS Developer's Guide to VMSINSTAL*, Version 5.5. It also supersedes the OpenVMS AXP Version 1.5.

Operating System and Version: OpenVMS VAX Version 6.0
OpenVMS AXP Version 1.5

Digital Equipment Corporation
Maynard, Massachusetts

May 1993

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

ZK4529

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

Preface vii

1 Introduction

1.1 Product Kits 1–2
1.1.1 Files Included in a Product Kit 1–2
1.1.2 Building a Product Kit 1–2
1.2 VMSINSTAL Functional Overview 1–5
1.3 Invoking VMSINSTAL 1–6
1.4 Choosing VMSINSTAL Options 1–8
1.4.1 Installer's Options 1–8
1.4.2 Developer's Options 1–11
1.5 VMSINSTAL History File (AXP Only) 1–12
1.6 VMSINSTAL Product Installation File (AXP Only) 1–12
1.7 List Installed Products Procedure (AXP Only) 1–12
1.8 Safety Mode 1–13
1.9 Recovery from System Failure 1–13

2 Guidelines and Conventions

2.1 Product Identification String 2–1
2.2 Save Set Identification 2–1
2.3 Volume Labeling 2–2
2.3.1 Diskette Kits 2–2
2.3.2 TU58 Cartridge Kits 2–3
2.3.3 Magnetic Tape Kits 2–3
2.4 Logical Names and Global Symbols 2–3
2.5 Error Handling 2–3
2.6 Compatibility Mode (VAX Only) 2–4
2.7 Referencing Other Products 2–4
2.8 Changing Global State 2–4
2.9 Verifying Installation 2–4
2.10 Prompting the Installer for Input 2–5
2.11 Internationalization of VMSINSTAL Messages 2–5
2.12 Release Notes 2–6

3 KITINSTAL Command Procedure

3.1 Guidelines for Writing a KITINSTAL Command Procedure 3–1
3.1.1 Installation Phase 3–1
3.1.2 IVP Phase 3–2
3.2 Using Callbacks 3–3
3.2.1 Accessing Files 3–4
3.2.2 Moving Files from the Kit's Working Directory 3–4

3.2.3 Updating Files 3–4
3.2.3.1 Updating an Existing File Version 3–4
3.2.3.2 Updating a File by Creating a New Version 3–4
3.2.4 Updating a Library 3–5
3.2.5 Deleting a File 3–5
3.2.6 Creating a Directory 3–5
3.3 Summary of KITINSTAL Design Specifications 3–6
3.4 Basic KITINSTAL Command Procedure 3–7

4 VMSINSTAL Functional Description

4.1 Overview 4–1
4.2 Functional Steps 4–2
4.2.1 Step 1 4–2
4.2.2 Step 2 4–3
4.2.3 Step 3 4–4
4.2.4 Step 4 4–4
4.2.5 Step 5 4–5
4.2.6 Step 6 4–5
4.2.7 Step 7 4–6
4.2.8 Step 8 4–6
4.2.9 Step 9 4–6
4.2.10 Step 10 4–7
4.2.11 Step 11 4–7
4.2.12 All Done 4–7
4.3 Special Steps 4–8
4.3.1 Step 12 4–8
4.3.2 Step 13 4–10

5 VMSINSTAL Callbacks

5.1 ADD_IDENTIFIER Callback 5–2
5.2 ASK Callback 5–2
5.3 CHECK_NETWORK Callback 5–5
5.4 CHECK_NET_UTILIZATION Callback 5–5
5.5 CHECK_PRODUCT_VERSION Callback 5–7
5.6 CHECK_VMS_VERSION Callback 5–8
5.7 COMPARE_IMAGE Callback 5–9
5.8 CONTROL_Y Callback 5–10
5.9 CREATE_ACCOUNT Callback 5–11
5.10 CREATE_DIRECTORY Callback 5–11
5.10.1 Creating a System Directory 5–12
5.10.2 Creating a System-Specific Directory 5–12
5.10.3 Creating a Common Directory 5–13
5.10.4 Creating a User Directory 5–13
5.11 DELETE_FILE Callback 5–14
5.12 FIND_FILE Callback 5–15
5.13 GET_IMAGE_ID Callback 5–16
5.14 GET_PASSWORD Callback 5–17
5.15 GET_SYSTEM_PARAMETER Callback 5–17
5.16 MESSAGE Callback 5–18
5.17 PATCH_IMAGE Callback (VAX Only) 5–19
5.18 PRINT_FILE Callback 5–20
5.19 PRODUCT Callback 5–21

5.20 PROVIDE_DCL_COMMAND Callback 5–21
5.21 PROVIDE_DCL_HELP Callback 5–22
5.22 PROVIDE_FILE Callback 5–23
5.23 PROVIDE_IMAGE Callback 5–24
5.24 RENAME_FILE Callback 5–27
5.25 RESTORE_SAVESET Callback 5–27
5.26 RUN_IMAGE Callback 5–28
5.27 SECURE_FILE Callback 5–29
5.28 SET Callback 5–30
5.28.1 SET ACL Option 5–30
5.28.2 SET ASK_CASE Option 5–31
5.28.3 SET_FILE Option (AXP Only) 5–31
5.28.4 SET IVP Option 5–32
5.28.5 SET POSTINSTALL Option 5–33
5.28.6 SET PRODUCT_NAME Option (AXP Only) 5–33
5.28.7 SET PURGE Option 5–34
5.28.8 SET REBOOT Option 5–35
5.28.9 SET SAFETY Option 5–35
5.28.10 SET_SEMANTICS Option (AXP Only) 5–36
5.28.11 SET SHUTDOWN Option 5–36
5.28.12 SET STARTUP Option 5–37
5.29 SUMSLP_TEXT Callback 5–38
5.30 TELL_QA Callback 5–40
5.31 UNWIND Callback 5–40
5.32 UPDATE_ACCOUNT Callback 5–40
5.33 UPDATE_FILE Callback 5–41
5.34 UPDATE_IDENTIFIER Callback 5–42
5.35 UPDATE_LIBRARY Callback 5–43

A Symbols and Logical Names

B Sample OpenVMS Installation Procedure

C Product-Specific Callback Conventions

D How to Use the VMI$VMS_VERSION Symbol

E Product Registration

Index

Examples

3–1 Basic KITINSTAL.COM 3–7
B–1 Sample DIGITAL OpenVMS KITINSTAL.COM B–2
C–1 Product_Specific Callback Procedure C–2
D–1 Template for Using the VMI$VMS_VERSION Symbol D–1

Preface

Intended Audience

This manual is intended for developers who design installation procedures for optional products that layer on the OpenVMS operating system.

Document Structure

Information in this manual is organized as follows:

- Chapter 1 presents an overview of the VMSINSTAL command procedure, including the command line for invoking it, and a description of the available installation options.
- Chapter 2 discusses the various guidelines and conventions used to design installation procedures that are compatible with VMSINSTAL.
- Chapter 3 describes the basic steps of the installation procedure. Because KITINSTAL has a generic structure, the specifics might differ slightly from one product to another.
- Chapter 4 describes the eleven functional steps and the two special steps of the VMSINSTAL logical sequence.
- Chapter 5 describes the VMSINSTAL callbacks, which are specialized subroutines that execute the various installation tasks.
- Appendix A lists and briefly describes the symbols and logical names used by VMSINSTAL.
- Appendix B provides an example of a KITINSTAL.COM installation procedure for an actual Digital product. This command procedure serves as a sample for writing product-specific command procedures.
- Appendix C provides an example of a product-specific callback. Installation designers can develop their own product-specific callbacks, as long as they follow the explicit guidelines.
- Appendix D provides a template for using VMI$VMS_VERSION, a symbolic value used to verify that the product being installed is compatible with the target system.
- Appendix E explains how to register a product.

Associated Documents

For information on specific system installation and upgrade procedures, see the installation and operations guide for your processor.

For information about writing OpenVMS command procedures, see the *OpenVMS User's Manual*.

Conventions

In this manual, every use of OpenVMS AXP means the OpenVMS AXP operating system, every use of OpenVMS VAX means the OpenVMS VAX operating system, and every use of OpenVMS means both the OpenVMS AXP operating system and the OpenVMS VAX operating system.

The following conventions are used to identify information specific to OpenVMS AXP or to OpenVMS VAX:

AXP	The AXP icon denotes the beginning of information specific to OpenVMS AXP.
VAX	The VAX icon denotes the beginning of information specific to OpenVMS VAX.
♦	The diamond symbol denotes the end of a section of information specific to OpenVMS AXP or to OpenVMS VAX.

The following conventions are also used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
GOLD *x*	A sequence such as GOLD *x* indicates that you must first press and release the key defined GOLD, then press and release another key. GOLD key sequences can also have a slash (/), dash (-), or underscore (_) as a delimiter in EVE commands.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.

[ ]	In format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification, or in the syntax of a substring specification in an assignment statement.)
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in Bookreader versions of the manual.
italic text	Italic text emphasizes important information, indicates variables, and indicates complete titles of manuals. Italic text also represents information that can vary in system messages (for example, Internal error *number*), command lines (for example, /PRODUCER=*name*), and command parameters in text.
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
numbers	All numbers in text are assumed to be decimal, unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

1
Introduction

The VMSINSTAL command procedure is the primary tool used by Digital to install OpenVMS updates and optional (layered) software products. This manual provides the installation designer with information about using VMSINSTAL to develop procedures for installing products that layer on the OpenVMS operating system.

VMSINSTAL lets the OpenVMS developer create installation procedures that conform to uniform standards and are compatible with other Digital products. Digital recommends that others who design layered products observe these same standards.

By using VMSINSTAL and following the guidelines provided in this manual, you can achieve the following installation procedure design goals:

- Build product kits as BACKUP save sets by using the Software Product Kit Building command procedure (SPKITBLD.COM). As a result, you can include larger files in the kit. In addition, the use of **checksum** blocks increases the chances of recovering from media errors that occur during the installation.
- Install products from a wide variety of media, including console media, magnetic tapes, disks, and both local and remote disk directories (installing products from remote directories requires DECnet software, an optional OpenVMS product).
- Minimize the effects of changes in OpenVMS from one version to the next on software product installation procedures.
- Include multiple products on a single distribution volume.
- Develop software product installations that are more uniform and consistent.

This chapter discusses the functional components and concepts required for using VMSINSTAL. The chapters that follow describe VMSINSTAL in more detail, including guidelines and conventions for its use, a description of the KITINSTAL command procedure, an outline of VMSINSTAL functional steps, and a reference section describing the VMSINSTAL **callbacks**.

The term **callback** is used throughout this manual to refer to a recursive invocation of a specialized subroutine within VMSINSTAL. For example, VMSINSTAL provides callbacks that move files, create directories, and delete files. Your KITINSTAL.COM procedure invokes a specific callback within VMSINSTAL. When VMSINSTAL has completed execution of the callback, it returns control back to the KITINSTAL.COM procedure. (See Appendix B.)

1.1 Product Kits

Software products that layer on OpenVMS are assembled into product kits that are physically distributed as one or more save sets (files created by the Backup utility).

The file name of each save set must be identical to the product name and be assigned a unique file type that reflects the order in which the save sets are installed. The first 26 product save-set files are assigned file types using alphabetic characters (for example, TEST011.A). See Chapter 2 for detailed information on product identification conventions.

1.1.1 Files Included in a Product Kit

A product kit includes some or all of the following files:

- KITINSTAL command procedure—The kit must include a command procedure that installs your product. The command procedure must be named KITINSTAL.COM, and it must be included in the kit's primary save set (assigned file type A). See Chapter 3 for an outline of how to write a KITINSTAL command procedure for your product.
- Release notes file—If online release notes are provided with your product and you select the N option, the release notes file must be included in save set A. (See Section 1.4.1 for more information on the N option.)
- Command language description (CLD) files—These files contain the command language descriptions that allow a user to invoke your product from the Digital Command Language (DCL) level.
- Product image files—These are the images that execute when a user invokes your product from the DCL level.
- Source help files—These files describe how to invoke your product and how to assign parameters and qualifiers most effectively.
- Help library files—These files contain the various modules of help text for your product. (See the *OpenVMS Command Definition, Librarian, and Message Utilities Manual* for more information on creating help files.)
- Other product files—Any other files required to use your product.
- Installation Verification Procedure (IVP)—This is a procedure that you design to check whether the product was installed properly. The procedure should display or write the results of the verification.

1.1.2 Building a Product Kit

You build your product kit by using the Software Product Kit Building command procedure (SPKITBLD.COM), located in the SYS$UPDATE directory. When you invoke SPKITBLD, you have the option of including required parameters on the command line, or you can have the command procedure prompt you for them.

If you include the required parameters on the command line, use the following format to invoke SPKITBLD:

```
@SYS$UPDATE:SPKITBLD product-name device product-files data-file
```

product-name

The product-name parameter (P1) is used to label the save sets. See Chapter 2 for details on the conventions for naming your product. This parameter must be null if you are specifying a data file.

device

The device parameter (P2) identifies the device that contains the media on which the product save sets are to be built.

product-files

The product-files parameter (P3) identifies the files that are being converted into save sets. Typically, you put the files for a particular save set in a dedicated directory and then use a wildcard input specifier for the BACKUP command. This parameter must be null if you are specifying a data file.

data-file

The data-file parameter (P4) is an optional parameter that identifies a data file that SPKITBLD can use to obtain necessary information. The following symbols can be defined in the data file:

1. SPKITBLD$KITNAME—The name of the kit that SPKITBLD is to build. Kit names have the following format:

   facvvu

   where:

fac	is the facility name
vv	is the major version number
u	is the update number

2. SPKITBLD$REWIND_TAPE—A Boolean flag that tells SPKITBLD whether save sets already exist on the media. The default is Yes.
3. SPKITBLD$SAVESET_n—One or more symbols that represent the files to be placed in the respective save sets. The variable *n* starts with the alphabet from A to Z and then proceeds from VMI_0027 through VMI_9999 (for example, the first symbol is SPKITBLD$SAVESET_A, the second is SPKITBLD$SAVESET_B, the twenty seventh symbol is SPKITBLD$SAVESET_VMI_0027, and so on.)
4. SPKITBLD$NEW_MEDIA_FOR_SAVE SET_n—A Boolean flag that tells SPKITBLD that save set *n* should begin on a new piece of media. This symbol applies only to small disks; kits being built on large disks or tapes must reside on the same piece of media. The default is No.
5. SPKITBLD$AUTOINIT_TAPE—A Boolean flag that tells SPKITBLD to reinitialize a tape that does not have the correct volume label. The default is Yes.
6. SPKITBLD$SKIP_FIRST_TAPE_READY_PROMPT—A Boolean flag that tells the SPKITBLD procedure to skip the first prompt, which asks if the tape is ready. The default is No.
7. SPKITBLD$SKIP_FIRST_DISK_READY_PROMPT—A Boolean flag that tells the SPKITBLD procedure to skip the first prompt, which asks if the disk is ready. The default is No.

The following are examples of data files :

```
! This data file builds a Pascal kit
!
SPKITBLD$KITNAME = "PASCAL038"
SPKITBLD$SAVESET_A = "PASCAL038.RELEASE_NOTES,KITINSTAL.COM"
SPKITBLD$SAVESET_B = "PASCAL.EXE,PASCAL.CLD"

! This data file builds a FORTRAN kit on the same media with Pascal
!
SPKITBLD$KITNAME = "FORT050"
SPKITBLD$REWIND_MEDIA = No
SPKITBLD$SKIP_FIRST_TAPE_READY_PROMPT = Yes  !Just use existing tape
SPKITBLD$SAVESET_A = "[.FORT050A]*.*"
SPKITBLD$SAVESET_B = "[.FORT050B]*.*"
SPKITBLD$NEW_MEDIA_FOR_SAVE SET_C = Yes       !Needs new media
SPKITBLD$SAVESET_C = "[.FORT050C]*.*"
```

When you invoke SPKITBLD, it prompts you for any additional data it needs and, if applicable, it prompts you to physically manipulate media.

Following is an example of how to invoke SPKITBLD by including required parameters on the command line:

```
$ @SYS$UPDATE:SPKITBLD TEST010 DUA0 DUB0:[SCRATCH]FRED.DAT,FRED.EXE,FRED.COM
```

Note that this example includes a list of files to be converted into save sets. The files in a list must be separated by commas, with no intervening spaces.

If you choose to have the SPKITBLD procedure prompt you for the required information, perform the following steps:

1. Invoke SPKITBLD as follows:

   ```
   $ @SYS$UPDATE:SPKITBLD
   ```

   A version message is returned.

2. The following prompt appears:

   ```
   * Do you want to use a kit building data file? [N]
   ```

   If you want to use a kit building data file, perform the following steps:

   a. Enter *Y* in response to the prompt.

   b. Specify the name of the data file in response to the following prompt:

      ```
      * Enter name of data file to use:
      ```

   c. Proceed to step 3.

   If you do not want to use a kit building data file, perform the following steps:

   a. Enter *N* or press Return in response to the prompt. The following message appears:

      ```
      Kits have names in the format "facvvu", where

          fac is the facility name
          vv is the major version number
          u is the update number
      ```

   b. Specify the name of the kit in response to the prompt:

      ```
      * Enter name of kit you want to build (facvvu):
      ```

3. Specify the destination device in response to the prompt:

   ```
   * On which device is the kit to be built?
   ```

4. The following message appears:

   ```
   * Are you ready (Y/N)?
   ```

   Enter *Y* when you are ready to proceed. The device is then mounted and allocated. The following message appears:

   ```
   Save set A must contain the KITINSTAL.COM procedure.
   ```

5. If you specified a data file for SPKITBLD to use, proceed to step 7. Otherwise, the following prompt appears:

   ```
   * Which files:
   ```

   Respond to this prompt by specifying the files to be included in the save set, and press Return.

   The following message appears:

   ```
   Resetting protection on all files to (S:RWED,O:RWED,G:RWED,and W:RE)
   ```

6. After the save set has been created, the following prompt appears:

   ```
   * Is there another save set to create (Y/N)[N]?
   ```

   If another save set is to be created, specify *Y*. Then, specify *Y* or *N* in response to the following prompt:

   ```
   * Does it need to start on a new volume (Y/N)[N]?
   ```

   If no other save set is to be created, press Return.

7. After all save sets have been created, the following message is returned to indicate a successful product kit build:

   ```
   Kit Build Completed Successfully
   ```

1.2 VMSINSTAL Functional Overview

The following list provides an overview of the functions performed by VMSINSTAL when it installs a layered product:

1. VMSINSTAL initializes the software environment, defines logical names and symbols, makes validity checks, and gives you an opportunity to back up your current system disk. All user-defined global and local symbols are deleted at this point.
2. VMSINSTAL then prompts for the distribution device, if you do not include it in the command line, and verifies that it is a valid device.
3. It then prompts for the product to be installed, if you do not include it in the command line, mounts the volume, and checks for the presence of the correct save sets.
4. Next, the environment needed to install the product is set up, including a work directory dedicated to the storage of the product files during the installation. The directory name reflects the product name and is logically referred to as VMI$KWD. VMI$KWD is either a subdirectory of the system-specific update directory, VMI$SPECIFIC:[SYSUPD], or a subdirectory of the device and directory specified by the alternate working device (AWD) option. The VMI$KWD subdirectory is deleted when the installation is completed.

5. The primary save set (save_set_name.A) is restored to the work directory VMI$KWD.
6. Next, the KITINSTAL command procedure, restored from the primary save set, is invoked. KITINSTAL uses callbacks to VMSINSTAL to do the installation.
7. When the installation is completed, the product's Installation Verification Procedure (IVP) is invoked, if applicable.
8. After doing any necessary housekeeping, VMSINSTAL prompts for the next product to install, if applicable.
9. If you specified an asterisk (*) when VMSINSTAL prompted you for the product, all products found on the distribution device are installed. VMSINSTAL then prompts for additional product kits.

1.3 Invoking VMSINSTAL

Invoke VMSINSTAL using the following command line format:

```
@SYS$UPDATE:VMSINSTAL product_list source:  [options] [option-list] [alternate_dir] [qualifiers]
```

product_list
Use this parameter (P1) to enter a list of the products to be installed. Separate multiple entries in P1 with commas. If you use a wildcard character (*) as the product_list parameter, all versions and updates of all products from the distribution source are installed in alphabetical order.

If you do not specify a product using this parameter, VMSINSTAL prompts for it.

This parameter is required.

source
Use this parameter (P2) to identify the source of the software product as one of the following:

- A device that holds the distribution medium.
- A storage directory where the product save set has been transferred from the distribution media for later installation (this occurs when you specify the **get save set** option described in Section 1.4.1).
- A disk directory on another node, if DECnet software is installed on both nodes.

If you do not specify the source, VMSINSTAL prompts for it.

You can use a logical name to specify the source.

This parameter is required.

options
Use this parameter (P3) to enter the keyword OPTIONS when VMSINSTAL options are to be listed in the next parameter (P4). The installation aborts if P4 contains an option list and the options keyword is omitted. Using this keyword minimizes that possibility.

option-list

Use this parameter (P4) to select VMSINSTAL options by entering the appropriate option letter. See Section 1.4 for more information on available options.

If you list more than one option, the letters representing the options must be separated by commas. Do not include spaces between letters; spaces are interpreted as parameter delimiters.

The following example of a command line for invoking VMSINSTAL specifies the alternate working device (AWD), file log (L), and release notes (N) options.

```
$ @SYS$UPDATE:VMSINSTAL TEST042 DUA0:[KITS] OPTIONS -
_$ AWD=DUA2:[INSTALL],L,N
```

This parameter is optional.

alternate_dir

By default, VMSINSTAL assumes that the product is to be installed in the common directory of the system on which you are running VMSINSTAL.

In the following instances, you can use this parameter (P5) to further qualify the destination for the product software:

1. If you select the **alternate root** option (R), the product is restored to a common directory in a system disk other than that on which the target system is running. In this case, P5 must identify the alternate system root using the following form:

   ddcu:[SYSn.]

   where:

ddcu:	is the destination disk device.
SYSn.	is the top-level directory of the alternate system root.

   VMSINSTAL will accept a previously defined logical name for the alternate root.

2. If you select the **get save set** option (G) to copy the product kit save sets into a storage directory for later installation, P5 must specify the directory in the following format:

   [node::]ddcu:[directory]

   where:

node::	is a node remote from the target system. (You must have DECnet software installed to use this option.) If no node is specified, VMSINSTAL assumes that the product save sets are to be stored on the local node.
ddcu:	is the destination disk device.
directory	is usually a directory dedicated to product save sets on the specified disk.

This parameter is optional.

qualifiers

Where applicable, use this parameter in conjunction with the **get save set** (G) option to further qualify the BACKUP command that copies the save sets to the destination directory. (See Section 1.4.1 for more information on the get save set option.)

The following example of a command line for invoking VMSINSTAL includes option G and BACKUP qualifiers:

```
$ @SYS$UPDATE:VMSINSTAL TEST042 DUA0:[KITS] OPTIONS -
_$ G DUB0:[KITS] "/VERIFY/LOG/CONFIRM"
```

The following example of a command line for invoking VMSINSTAL uses a wildcard character (*) as the product-list parameter to install all products from the distribution source:

```
$ @SYS$UPDATE:VMSINSTAL * DUA0:[KITS] OPTIONS N,L
```

The following example specifies a product name, TEST, as the product_list to install all versions of a product from the distribution source:

```
$ @SYS$UPDATE:VMSINSTAL TEST DUA0:[KITS] OPTIONS N,L
```

This parameter is optional.

VMSINSTAL only prompts the installer for options when *both* of the following conditions are true:

- Options are not specified on the command line.
- VMSINSTAL must also prompt for either the **product** or **source device** parameters.

1.4 Choosing VMSINSTAL Options

The VMSINSTAL command procedure permits the use of twelve options, six that apply to the installer and six that are used for testing the installation procedures during development and acceptance testing. When invoking VMSINSTAL, specify options by entering the appropriate letters in the option-list parameter of the command line.

1.4.1 Installer's Options

Following is a description of the six VMSINSTAL installer's options. These options can also be used when testing an installation procedure.

- **A**—The **auto-answer** option makes it easier to reinstall a product by providing responses to VMSINSTAL questions and prompts during the reinstallation. The auto-answer option is used most often for reinstalling products after a system is upgraded.

  If you specify the auto-answer option when initially installing the product, VMSINSTAL creates an answer file in the SYS$UPDATE directory. The name of the answer file is in the following format:

  product_name.ANS

  where:

product_name	is a file name that reflects the product-list parameter in the call to VMSINSTAL.

  The answer file records responses to questions and prompts from VMSINSTAL during the installation. When you subsequently reinstall the product and specify the auto-answer option (typically after a system upgrade), VMSINSTAL reads the answer file instead of prompting the installer for responses.

You should be familiar with the questions asked during a particular product installation before using the auto-answer option. Many products ask different questions depending on the environment. If the questions change format, or if questions are added or skipped when you are using answer files, the installation terminates.

To create a new answer file when reinstalling a product, you must first delete or rename the existing answer file.

- **AWD**—The **alternate working device** option lets you specify an alternate working device for the temporary working directory (defined as the logical VMI$KWD). This option enables you to perform an installation with fewer free blocks on the VMI$ROOT device than is otherwise required.

  If you do not specify this option, VMSINSTAL creates the temporary working directory in the following location:

  ```
  SYS$SPECIFIC:[SYSUPD.facvvu]
  ```

  The variable *facvvu* represents the product identification string.

  Specify this option using the following format:

  ```
  AWD=dev:[dir]
  ```

  dev:
  Specifies the alternate working device.

  dir
  Specifies the directory under which the *facvvu* subdirectory will be created. Specifying a directory is optional. If you do not specify a directory, VMSINSTAL creates the working directory on the specified device with the following directory specification: [000000.facvvu]. If you specify a directory, VMSINSTAL creates the working directory as a subdirectory to the directory that you specify (for example, [WORK.facvvu]). The directory that you specify must already exist. If it does not exist, VMSINSTAL will not create it.

  For example, to create the working directory [INSTALL.facvvu] on the alternate device DUA2, specify the following:

  ```
  AWD=DUA2:[INSTALL]
  ```

- **G**—The **get save set** option is used to copy the product kit save sets into a disk directory or other storage device (such as a magnetic tape) for later installation. When option G is selected, all kit save sets are copied, but no installation is performed. When save sets are copied using this option, the directory structure originally assigned to files within the save set are not maintained. Because the directory structure is lost, this option cannot be used to copy OpenVMS operating system kits. (See Section 4.3.2 for more information on this option.)

- **L**—The **file log** option is used to log all file activity to the controlling terminal during installation. File activity is defined as any action that alters the disposition of a file, such as creating a new file, updating a library, or deleting a file.

- **N**—The **release notes** option is used to display or print the release notes file supplied by the layered product.

  The release notes file is named by the person who builds the product kit. It is given the file-name fac_vvu.release_notes, where *fac_vvu* represents the product facility code, version and update numbers. The release note file must

be included in the primary save set, that is the save set containing the file extension .A (for example, TEST042.A).

See Section 2.12 for more information on product release notes. During the installation of a product, VMSINSTAL moves release notes files to the SYS$HELP directory. See Chapter 2 for more information about product naming conventions.

If release notes are available and the N option is specified, VMSINSTAL prompts the installer with the following questions (the default answers are indicated in brackets):

```
   Release notes included with this kit are always copied to SYS$HELP.

   Additional Release Notes Options:

       1. Display release notes
       2. Print release notes
       3. Both 1 and 2
       4. None of the above

* Select option [2]:

* Queue name [SYS$PRINT]:

* Do you want to continue the installation [N]:
```

The first prompt (* Select option:) allows the installer to (1) display the release notes on a terminal, (2) print the release notes, (3) print and display the notes, or (4) choose none of these options.

The second prompt (* Queue name:) is displayed only if option 2 or 3 is selected. If you enter the name of a print queue, the system returns a message indicating that the release notes have been successfully queued to the printer. If a print queue is not specified, the output is directed to SYS$PRINT by default.

The third prompt (* Do you want to continue the installation:) allows you to either continue or terminate the installation. The default is to terminate the installation at this point.

If no release notes are supplied with the product, VMSINSTAL returns two error messages, followed by the prompt to continue or terminate the installation, as follows:

```
%VMSINSTAL-W-NOFILE, New file *.RELEASE_NOTES does not exist.
%VMSINSTAL-W-NORELNOTE, Unable to locate release notes.

* Do you want to continue the installation [N]:
```

You can continue the installation by entering *Y* in response to this prompt, regardless of whether or not release notes are available.

- **R**—The **alternate root** option is used to install the product on a system disk other than that of the running system. This option makes it possible to test a new product without disturbing the running system.

  The OpenVMS system in the alternate root must be complete and at the same version level as the running system. Also, all files and software products referenced by the product installation must be present in the alternate root.

  All files are installed under the common directory of the alternate root.

1.4.2 Developer's Options

Following is a description of the six VMSINSTAL options that are not normally used by the installer. These options are usually used for test purposes.

- **C**—The **callback trace** option traces all callbacks to VMSINSTAL during the installation and lists them in the file [SYSUPD]*facvvu*.CBT. See Chapter 2 for an explanation of the product identification facility code (facvvu).
- **I**—The **inhibit initial prompts** option is used to suppress the following initial VMSINSTAL prompts:

  ```
  * Do you want to continue anyway [NO]?
  * Are you satisfied with the backup of your system disk [YES]?
  * Where will the distribution volumes be mounted:
  ```

- **K**—The **kit debug** option directs VMSINSTAL to pass a Boolean value that may be used as a debugging tool by KITINSTAL.

  The kit debug option disables verification (SET NOVERIFY) while VMSINSTAL processes callbacks. Verification is reactivated after callbacks are processed.

- **Q**—The **quality assurance mode** (QA mode) option specifies that the installation is being done in test mode. It enables various functions that are peculiar to the test environment and ignores other functions that are not required in the test environment. For example, TELL_QA callbacks are performed and logged to the controlling terminal.
- **RSP**—The **restore save set and pause** option is a debugging option that causes the installation procedure to pause after restoring each save set. At the pause you can either press Return at the prompt to resume, or log in to another terminal if you want to create a subprocess.

  This option lets you test a kit without rebuilding the kit after each minor fix. For example, during the pause, you might edit a file that was previously restored, or replace one version of a file with another.

  You can also pause the installation procedure after a particular save set is restored by specifying the option in the following format; *s* indicates the save set to be restored before pausing.

  ```
  RSP=s
  ```

  For example, to pause after the installation procedure restores save set C specify the following option:

  ```
  RSP=C
  ```

  If you do not specify a save set with the RSP option, the installation procedure pauses after each save set is restored.

- **S**—The **statistics** option 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 OpenVMS software and enough free blocks for an installation in **safety mode**. See the following section for more information about safety mode.

1.5 VMSINSTAL History File (AXP Only)

AXP

On AXP systems, a VMSINSTAL history file, SYS$UPDATE:VMSINSTAL.HISTORY, is updated when VMSINSTAL terminates. The new file entry indicates the product for which installation was attempted, and the status of the installation. To view this history file, use the TYPE or PRINT commands.♦

1.6 VMSINSTAL Product Installation File (AXP Only)

AXP

On AXP systems, a log file named SYS$UPDATE:facvvu.VMI_DATA is created following a successful product installation. This file contains the following information:

- Product that was installed
- Who installed the product
- Names of files that were added, deleted, or modified during installation

To view this log file, use the TYPE or PRINT commands.♦

1.7 List Installed Products Procedure (AXP Only)

AXP

On AXP systems, the procedure SYS$UPDATE:INSTALLED_PRDS.COM lets you check what products have been installed. This procedure lists the product name and version, date of installation, and who installed the product. The procedure has an optional parameter to specify a restricted search of installed products.

Use the following format to invoke this procedure:

@SYS$UPDATE:INSTALLED_PRDS [product-mnemonic]

where:

product_mneumonic	is an optional string that specifies one or more product save set names to search for. This string can contain a save set name (product name and version), a product name only, or wildcard characters. Installation data is displayed only for product log files that are found on the system.

For example:

```
$ @SYS$UPDATE:INSTALLED_PRDS RDBVMS030
```

This procedure lists information only if a data file for RDBVMS V3.0 is found.

```
$ @SYS$UPDATE:INSTALLED_PRDS RDBVMS
```

This procedure lists information for all versions of RDBVMS for which a data file exists.

```
$ @SYS$UPDATE:INSTALLED_PRDS R*
```

This procedure lists information for all versions of products whose name begins with the letter R.

The following is a sample listing:

```
$ @SYS$UPDATE:INSTALLED_PRDS RDBVMS

        Node: AZSUN

        Installed products:  2-APR-1992 13:13

    Product: AXP RDB/VMS
    Mnemonic: RDBVMS    Version: 3.0
    Installed: 12-SEP-1992
    Installed by: SOMMER on node AZSUN♦
```

1.8 Safety Mode

On both AXP and VAX systems, VMSINSTAL is designed to minimize the risk of installation failure by executing the installation in two phases. A safety mode feature allows the installer to defer critical operations until the second phase of the installation.

During the initial installation phase, operations that may severely impact your system if a failure occurs are deferred until the second phase. These critical operations include moving new files into the system directories, updating existing files, and patching system images. Note, however, that system libraries are updated immediately because they may be required by the installation process itself.

The primary tool for deferring critical operations is called the **defer file**. This is a command file (VMIDEFER.COM) that VMSINSTAL creates in the work directory to record deferred commands.

If the installation is in safety mode, VMSINSTAL checks to see whether or not an installation command is critical. If it is, VMSINSTAL records the command in VMIDEFER.COM (logically referred to as VMI$DEFER_FILE); otherwise, the command is executed immediately.

When the installation progresses to the second phase, each of the deferred commands is executed in the order in which it was originally issued.

You set the required level of safety for the installation by specifying the SAFETY option with the SET callback. By default, VMSINSTAL automatically implements safety mode operations "unconditionally," regardless of whether you use the SET SAFETY option. However, unconditional safety mode requires a higher peak disk space. If you use the SET SAFETY option, you can specify the keyword CONDITIONAL to implement safety mode only if there is sufficient disk space to support it.

Your installation procedure should use the SET SAFETY option to help the installers of your product select a feasible level of safety for their particular situation. See Section 5.28.9 for more information on this option.

1.9 Recovery from System Failure

In addition to providing the safety feature implemented through deferred operations, VMSINSTAL creates a **marker file** in the system update directory to aid in recovery from system failures that occur during the installation. The marker file is named VMIMARKERpid.DAT, where pid is the process identification, and logically referred to as VMI$MARKER_FILE. This file periodically records state information as the installation progresses. By opening

the marker file for brief periods, the possibility that this file will be impacted by a system failure is minimized.

When the system startup procedure detects a marker file as it attempts to recover from a system failure, it calls VMSINSTAL with the **booting** option (B). This option (reserved for Digital use only) indicates that an installation was in progress when the failure occurred. VMSINSTAL responds by branching to a subroutine that determines what action is needed. Depending on the information in the marker file, VMSINSTAL performs one of the following actions:

- If the failure occurred before KITINSTAL was invoked, the installer is told to start the installation again.
- If the failure occurred while KITINSTAL was installing the product, the following conditions are evaluated:
  - If a library was being updated at the time of the failure, the installer is told to restore the library from the backup copy of the system and to start the installation again.
  - If the installation was in unsafe mode at the time of the failure, the installer is told to restore the entire system from the backup copy and to start again.
  - For all other conditions, the installer is told to start the installation again.
- If KITINSTAL has executed the deferred callbacks, the installation should complete satisfactorily.
- If KITINSTAL has completed the installation, the installer is told that the installation was completed satisfactorily.

2
Guidelines and Conventions

This chapter describes guidelines and conventions for developing products to be installed with VMSINSTAL. All VMSINSTAL users should closely adhere to the product naming conventions presented in the sections that follow.

2.1 Product Identification String

Each software product must be identified by a product identification string. The product identification string consists of the facility code *(fac)*, the version number *(vv)*, and the update number *(u)*. The facility code can have up to 36 alphanumeric characters.

The following example gives the format of the product identification string:

facvvu

where:

fac is the facility code.

vv is the current version number.

u is the current update number.

For example, you would identify Version 1.1 of a product named TEST as TEST011.

As new versions and updates of your product are released, they must be assigned a new product identification string that reflects the current version and update. For example, the next update after CHECKTRAN Version 1.1 would be assigned the product identification string CHECKTRAN012.

Do not give your product the name of an existing product or component on the target system. You can check the names of existing system and user images by entering the following DCL command:

```
$ DIRECTORY SYS$SYSTEM:*.EXE
```

To avoid potential conflicts with other layered products, you must register the facility code for each product with the Digital OpenVMS Product Registrar. For more information about product registration, see Appendix E.

2.2 Save Set Identification

Optional OpenVMS software products are distributed in product **kits**. A kit is made up of one or more save sets created by the Backup utility, each with a name that reflects the product facility code.

The file name of each save set must be identical to the product identification string, with the addition of a unique file type (the sequence identifier) that reflects the installation order of the product.

The first 26 product save-set files are assigned file types using alphabetic characters. That is, you assign file type A to the save set you want installed first, file type B to the second, and so forth.

If your product requires more than 26 save sets, the additional save sets must begin with VMI_0027, VMI_0028, and so forth ranging from VMI_0027 through VMI_9999. The sequence identifier (file type) specifies the restoration order for your product save sets and can handle up to 9999 save sets per kit.

Following is a summary of the save-set naming format:

facvvu.s

where:

fac is the facility code.

vv is the current version number.

u is the current update number.

s is the sequence identifier.

The primary save set (save set A) must include the KITINSTAL command procedure. If release notes are supplied with the product kit and you use option N, the release notes file must also be included in save set A.

2.3 Volume Labeling

Product save sets can be distributed on disk or tape media, including console media. It is strongly recommended that you adhere to the conventions for volume labeling given under the kit descriptions that follow. Note that the owner user identification code (UIC) of files in the save sets is irrelevant. File protection must be set as follows:

- System—read, write, execute, delete
- Owner—read, write, execute, delete
- Group—read, write, execute, delete
- World—read, execute

2.3.1 Diskette Kits

Each volume label for a diskette kit should be in the format *facs*, where the variable *fac* represents the facility code, and the variable *s* represents the save-set sequence identifier. The kit must be created directly on master distribution volumes using BACKUP to create sequential disk save sets. The following qualifiers are required:

```
/INTERCHANGE
/VERIFY
/BLOCK_SIZE=9000
/GROUP_SIZE=25
```

The specified block and group size result in optimal use of the blocks on the diskettes. Be sure to initialize double-density diskettes for single-density use so that they can be used on VAX11-780 RX01 console diskette drives.

2.3.2 TU58 Cartridge Kits

TU58 kits follow the same rules and conventions as diskette kits.

2.3.3 Magnetic Tape Kits

Magnetic tape volume labels must consist of no more than six characters, in the format *fac*, where the variable *fac* represents the facility code. To create save sets, the kit must be created directly on the master volume, using BACKUP with the /INTERCHANGE and /VERIFY qualifiers. Save sets must be placed on the tape in sequential order. There is no restriction to the number of kits that can be placed on one magnetic tape. Save set names cannot exceed 17 characters, including the period and file type fields.

2.4 Logical Names and Global Symbols

All logical names and global symbols assigned by VMSINSTAL begin with the prefix *VMI$*. KITINSTAL.COM logical names and global symbols must be prefixed with your product facility code followed by an underscore (_). For example, a FORTRAN product might use the prefix FORT_. Local symbols can use any name.

Avoid using abbreviations in the KITINSTAL procedure's DCL commands. Do not abbreviate verbs because special symbols may be defined for them by VMSINSTAL. Qualifiers should also be spelled out, but you can truncate them to a minimum of four characters where it is absolutely necessary.

2.5 Error Handling

The KITINSTAL.COM file must include an ON WARNING statement to handle errors that might be detected within the kit's procedures or that might be returned by a callback. (For example, VMI$_FAILURE is a warning status.) If no housekeeping is needed, the ON WARNING statement can exit, propagating the status up to the next level:

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

If some housekeeping is needed, the error routine could be structured as follows:

```
$  ON WARNING THEN GOTO ERROR
     .
     .
     .
$ERROR:
$    S = $STATUS
$    .            !
$    .            ! Use this routine to do your housekeeping,
$    .            ! to close any files that are open, and
$    .            ! to handle errors that might occur during
$    .            ! housekeeping.
$    .            !
$    EXIT S
```

2.6 Compatibility Mode (VAX Only)

VAX

On VAX systems, the KITINSTAL procedure must sometimes check for compatibility mode. Compatibility mode was distributed as a separate layered product on VAX systems beginning with VMS Version 4.0. See the following section for more information about referring to other products in your KITINSTAL procedure.

To check for compatibility mode, add the following line to your KITINSTAL.COM file:

```
$ VMI$CALLBACK FIND_FILE rsx vmi$root:[sysexe]rsx.exe "" se
```

If this file is present, you can assume that compatibility mode can be used on the system.♦

2.7 Referencing Other Products

On AXP and VAX systems, your installation procedure may need to refer to other products in one of the following ways:

- If the installation procedure must alter its logical flow because of the existence of another product but does not need to refer to that product, you can use the FIND_FILE callback to verify the existence of the product. See Section 5.12 for more information on this callback.
- If the procedure refers to another product's files but does not invoke the product, you can use the FIND_FILE callback to verify the product's existence and to set up a logical name for accessing the file.
- If the procedure must invoke another product, you can use the FIND_FILE callback to determine the existence of the product, then invoke the product with the appropriate command. Because this usually involves implicit references to the other product's files, VMSINSTAL assumes that these files are in the system root or on a user disk.
- If a product requires a minimum version of another product to be installed on the system, use the CHECK_PRODUCT_VERSION callback. See Section 5.5 for more information on this callback.

2.8 Changing Global State

If your installation procedure changes the global state of the system (except in reference to callbacks), it must restore that state before terminating. For example, if the installation procedure needs to invoke the Install utility to install an image that is needed for completing the installation, it must deinstall the image before exiting. Any use of the Install utility following installation of the product should be provided by a product-specific startup command procedure. (For more information, refer to the the SET STARTUP option description in Section 5.28.12.)

2.9 Verifying Installation

Your product kit should include an Installation Verification Procedure (IVP), which is a procedure for checking the completeness and accuracy of the installation.

The IVP may consist of many separate files but it should be invoked by one command procedure that can be run independently of the installation procedure.

The IVP command procedure should adhere to the following naming convention:

facility-name_IVP.COM

For example, the IVP command procedure for the product CHECKTRAN would be named CHECKTRAN_IVP.COM.

The IVP may be invoked from KITINSTAL.COM but the installer must be able to invoke it any time after the installation. Therefore, the IVP must remain available after the installation. Digital suggests that the IVP be placed in the SYS$TEST directory.

The kit's installation procedure prompts the installer on whether to execute the IVP. If the installer wants to execute the IVP, then VMSINSTAL will run it when the installation is complete by invoking the kit installation procedure with the VMI$IVP request code.

VMSINSTAL tries to set up a realistic work environment before running the IVP by doing the following:

- It assumes that KITINSTAL has put all product files in place.
- If the installation procedure specified a product-specific startup procedure with the SET STARTUP callback, VMSINSTAL invokes it immediately before running the IVP.
- VMSINSTAL sets the default directory to the kit's working directory, thus simulating a real user's environment.
- The IVP should not use callbacks or the VMI$_SUCCESS or VMI$_FAILURE symbols.

At the completion of the IVP, KITINSTAL.COM exits back to VMSINSTAL with VMI$_SUCCESS if the IVP is successful, or with VMI$_FAILURE if the IVP was unsuccessful. VMSINSTAL does not attempt to undo an installation if the IVP fails.

For specific guidelines for creating an IVP see Section 3.1.2.

2.10 Prompting the Installer for Input

When you design an installation procedure, keep in mind that the procedure should prompt the installer for most input at the beginning of the installation. This minimizes installer interaction when the installation is in progress.

2.11 Internationalization of VMSINSTAL Messages

All messages generated by VMSINSTAL are defined as symbols in the file SYS$MESSAGE:VMSINSTAL_LANGUAGE.COM. Customers developing applications for non–English speaking markets might want to change these symbol values to make installations understandable for the intended market.

AXP

On AXP systems, you can specify a language file that is appropriate for your needs. To specify a special language file, define the logical VMSINSTAL_LANGUAGE to point to that specific language file. For example:

```
$ DEFINE VMSINSTAL_LANGUAGE SYS$MESSAGE:VMSINSTAL_LANGUAGE_GERMAN.COM
```

This file must be compatible with the provided VMSINSTAL_LANGUAGE file. Message numbers must not be reassigned to different functions. Only the text portion of the messages should be modified.

If the specified language cannot be found or is not valid, VMSINSTAL uses the default language file.♦

2.12 Release Notes

Release notes provide information on changes made to a product since the previous release of that product. Digital recommends that all product kits include both a printed and online version of release notes. This section provides guidelines for including online release notes.

The Release Notes (N) option of VMSINSTAL enables the installer to display or print online release notes before beginning an installation. To use this option, the installer must invoke VMSINSTAL. However, once VMSINSTAL has been invoked and the release notes are printed or displayed, the installer has the option to continue or cancel the installation. Product files do not have to be restored from the kit at that time. See Section 1.4.1 for more information on the N option.)

The instructions for displaying and printing release notes should be identical for all products. Provide these instructions in both the product's installation guide and the cover letter.

The online release notes file should be a machine-readable, printable file. The name of this file must adhere to the following format:

fac_vvu.RELEASE_NOTES

The variable *fac* is the product's facility name, and *vvu* is the version number. For example, TEST_042.RELEASE_NOTES is the name of the release notes file for TEST Version 4.2.

If your file name follows the recommended format, VMSINSTAL can construct the correct name of the release notes file. The file can then be restored and copied to the SYS$HELP directory without adding a callback statement to KITINSTAL to provide the name of the file. VMSINSTAL relies on this file name format when purging the release notes files, if the installer chooses this option. This recommended format also helps users to easily find release notes files in SYS$HELP, or to find all files for a particular product.

Do not include the release notes text in the product Help files. Instead, include the release notes file in the kit's primary saveset (saveset-name.A) along with the KITINSTAL procedure (and any other command procedures invoked by KITINSTAL) . This permits the user to view or print the information at the beginning of the installation.

The product's Help library may contain selected release note topics. However, the entire text of the release notes should still be supplied in a separate file. The product's Help file might contain a reference that points to this file. For example, a user might enter the command HELP CHECKTRAN RELEASE_NOTES to see the following Help text:

```
release notes for CHECKTRAN, are contained in the file:

 SYS$HELP:CHECKTRAN_042.RELEASE_NOTES

You can type or print this file to read the release note
information.
```

The product-specific Help file could also include directions for using the DIRECTORY command to find the file (and earlier versions of it.)

To eliminate the need to reload the Help library with each release, Help text on release notes may be written generically, that is, without specific reference to version number.

Note

Digital recommends that your product's documentation advise users to keep previous editions of online release notes after receiving a new release of a product.

3
KITINSTAL Command Procedure

After your primary product save set is restored to the installation working directory, VMSINSTAL invokes the KITINSTAL command procedure to direct the installation. KITINSTAL issues a series of callbacks to VMSINSTAL by using the following callback command line format:

```
VMI$CALLBACK callback [parameter-2 ...]
```

The symbol VMI$CALLBACK invokes VMSINSTAL, which in turn uses the first parameter (the name of the callback) to execute the appropriate subroutine. The remaining parameters are the arguments for the particular callback. (See Chapter 5 for detailed descriptions of the callbacks.)

3.1 Guidelines for Writing a KITINSTAL Command Procedure

When you design the KITINSTAL command procedure for your product, be sure to observe the guidelines and conventions described in this section. They apply to both the installation phase and the IVP phase of KITINSTAL.

3.1.1 Installation Phase

This section provides guidelines and conventions for the installation phase of KITINSTAL.COM. See Section 3.1.2 for guidelines governing the IVP phase.

- Naming conventions—All global symbols and logical names created by VMSINSTAL have the prefix VMI$. Your product global symbols and logical names must use a prefix consisting of the product name followed by an underscore. For example, each global symbol and logical name created by the product CHECKTRAN requires the CHECKTRAN_ prefix. Local symbols can take any name.
- Logical references—Your installation procedure must use the following logical references:
  - VMI$KWD—The logical name VMSINSTAL creates for the kit working directory. Any reference your product makes to the kit working directory must use this logical name.
  - VMI$ROOT—The logical name VMSINSTAL creates for the target system's root directory. Again, all references to this directory must use the logical name assigned by VMSINSTAL, not the usual system logical names. For example, your installation procedure must refer to the system library directory using VMI$ROOT:[SYSLIB] rather than SYS$LIBRARY.
  - VMI$SPECIFIC—The logical name VMSINSTAL creates to reference the target system's system-specific top-level directory. VMI$SPECIFIC points to the system-specific top-level system directory, whereas VMI$ROOT points to the common top-level system directory. Your installation procedure must use this reference instead of using SYS$SPECIFIC.

- Subprocedures—You can include product-specific callbacks in the installation kit and invoke them from KITINSTAL (see Appendix C for information about product-specific callbacks). However, you can invoke the VMSINSTAL callbacks only from KITINSTAL and subprocedures from one level below KITINSTAL.
- Defaults—The following defaults are in effect when KITINSTAL is invoked:
  - The UIC is [1,4].
  - The default file protection is [s:rwed,o:rwed,g:rwed,w:re].
  - The default device and directory are MISSING:[MISSING]. VMSINSTAL sets the default to this nonexistent directory to prevent products from assuming a default device and directory. To move or update any files, a full file specification must be given, otherwise VMSINSTAL will try to do the work in MISSING:[MISSING] and fail.
  - All privileges except BYPASS are in effect.
  - Messages appear in full format.
- SET commands—In the installation portion of KITINSTAL.COM (as opposed to the IVP portion), you are restricted from using the SET commands so that you do not alter the environment that VMSINSTAL has set up. Exceptions to this rule are as follows:
  - You can use the SET ON and SET NOON commands without restriction.
  - You can use the SET VERIFY command only if the installer specifies the debug option.
  - You can use the SET FILE command only with files in VMI$KWD.
- SHOW command—Your KITINSTAL command procedure must not rely on the output format of the SHOW command or the output format of any utility.

3.1.2 IVP Phase

This section provides guidelines and conventions for the IVP phase of KITINSTAL.COM.

- The IVP may be invoked by KITINSTAL.COM but the installer must be able to invoke it any time after the installation. If you follow the guidelines listed here, you will ensure that the installer can consistently invoke the IVP by using the following command line format:

  ```
  @SYS$TEST:facility-name_IVP
  ```

- Put the IVP in a separate file. This allows the IVP to run independently of the installation procedure. The IVP may consist of separate files, but it should be invoked by one controlling command procedure.
- Name the IVP according to the following format:

  ```
  facility_IVP.COM.
  ```

  For example, you might name a procedure CHECKTRAN_IVP.COM.

- The KITINSTAL.COM should tell the installer the location of the IVP.

  Digital recommends that the IVP be located in the SYS$TEST directory If other files are required to perform the installation verification, for example, a data file or executable image, these files must be placed in a subdirectory of the SYS$TEST directory. The name of the subdirectory should be the same as the facility name of the product. For example, a subdirectory named SYS$SYSDEVICE:[SYSTEST.CHECKTRAN] would hold additional files for the IVP for the product CHECKTRAN.

  Use the CREATE_DIRECTORY callback to create the subdirectory. For example, the following command creates a subdirectory for a product called CHECKTRAN:

  ```
  $ VMI$CALLBACK CREATE_DIRECTORY USER VMI$ROOT:[SYSTEST.CHECKTRAN]
  ```

- Digital recommends that the IVP define a logical name that points to the subdirectory holding supplementary IVP files. This logical name can be used throughout the IVP to refer to these files.

- The KITINSTAL.COM must ask the installer whether to run the IVP. To do so, use the SET callback as follows:

  ```
  $ VMI$CALLBACK SET IVP ASK
  ```

  If the installer wants to run the IVP, then VMSINSTAL will perform it after the installation is complete, by invoking the KITINSTAL procedure as follows:

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

- The kit installation procedure must handle the request code VMI$_IVP.

- The IVP cannot use VMSINSTAL callbacks.

- The IVP should not use the VMI$_FAILURE and VMI$_SUCCESS symbols.

- The IVP should issue a message indicating the success or failure of its execution. This message should include the product name and version. For example, an IVP might issue the following message:

  ```
  IVP for CHECKTRAN V3.0 completed successfully.
  ```

- At the completion of the IVP, the IVP must exit with a status back to the installation procedure.

- If the IVP fails, VMSINSTAL does not attempt to undo the installation. The installation procedure must ensure that the product installs correctly.

For general information on the IVP see Section 2.9.

3.2 Using Callbacks

This section describes how to execute functions from KITINSTAL using callbacks to VMSINSTAL.

3.2.1 Accessing Files

While the product is being installed, the only files that KITINSTAL can access directly are those in VMI$KWD. All other files must be accessed by using logical names supplied by various callbacks.

To gain read-only access to a file outside of VMI$KWD, first invoke the FIND_FILE callback. Then use the logical name provided by FIND_FILE to access the file. For example, the following shows how you would access a file in VMI$ROOT:[SYSTEST] from KITINSTAL:

```
$ VMI$CALLBACK FIND_FILE TSTFILE TEST.DAT VMI$ROOT:[SYSTEST] S
$ TYPE TSTFILE
```

See Section 5.12 for more information on the FIND_FILE callback.

3.2.2 Moving Files from the Kit's Working Directory

To move a file from the kit working directory (VMI$KWD) to a system directory or a user directory, use the PROVIDE_FILE callback. The following KITINSTAL command moves a file to VMI$ROOT:[SYSUPD] and assigns a logical name to the file for future reference:

```
$ VMI$CALLBACK PROVIDE_FILE CHECKTXT CHECKTRAN.TXT VMI$ROOT:[SYSUPD]
```

See Section 5.22 for more information on the PROVIDE_FILE callback.

3.2.3 Updating Files

There are two ways to update a file. You can either update an existing version or create another version.

3.2.3.1 Updating an Existing File Version

To update an existing version of a file, use the UPDATE_FILE callback to copy the file to VMI$KWD. Then use the logical name assigned by the callback to modify the file as needed. The callback provides for the subsequent return of the updated file to the source directory.

If the installation is set for safety operation, the callback provides for the return of the updated file when the installation progresses beyond safety mode. However, if the installation is not set for safety operation, you must provide an explicit callback command to return it to the source directory after you modify it. If the file is found in VMI$KWD, the file is not copied or returned to the source directory.

Following is an example of how to access an existing file during safety mode:

```
$ VMI$CALLBACK UPDATE_FILE CHECKTRAN VMI$ROOT:[SYSEXE]CHECKTRAN.DAT
```

3.2.3.2 Updating a File by Creating a New Version

To update a file by creating another version, you must find the file, copy it to VMI$KWD, update it, and then move the updated version back to the original directory with the PROVIDE_FILE callback. Use the SET PURGE callback or the K option to specify whether you want to keep the original version of the file after the updated version is created.

The following example updates a file in VMI$ROOT:[SYSTEST] by creating a new version:

```
! Find the file
$ VMI$CALLBACK FIND_FILE OLD OLDCHECK.DAT VMI$ROOT:[SYSTEST] S
! Use the logical name to copy it to vmi$kwd
$ COPY OLD VMI$KWD:OLDCHECK.DAT
! Update the file
$ OPEN OLDCHECK VMI$KWD:OLDCHECK.DAT
$ OPEN NEWCHECK VMI$KWD:NEWCHECK.DAT/WRITE
       .
       .
       .
$ CLOSE OLDCHECK
$ CLOSE NEWCHECK
! Move the file back to the source directory
$ VMI$CALLBACK PROVIDE_FILE NEW NEWCHECK.DAT VMI$ROOT:[SYSTEST]
```

Be careful not to use the same logical name in the PROVIDE_FILE callback as that used in the FIND_FILE callback.

3.2.4 Updating a Library

Use the UPDATE_LIBRARY callback to update libraries. The following example updates the library in [SYSLIB] named CHECKTRAN.TLB, by adding a text file from the product kit:

```
! Using the logical name (CHECKDOC), update the library
$ VMI$CALLBACK UPDATE_LIBRARY CHECKTLB -
_$ VMI$ROOT:[SYSLIB]CHECKTRAN.TLB TEXT "/INSERT" CHECKDOC
```

3.2.5 Deleting a File

Use the DELETE_FILE callback to delete a file. For example, if you want to delete the command language description file CHECKTRAN.CLD after you use it to update the DCL tables, use the following command:

```
$ VMI$CALLBACK DELETE_FILE VMI$ROOT:[SYSHLP]CHECKTRAN.HLP
```

Where applicable, the DELETE_FILE callback automatically removes known images created by INSTALL.

3.2.6 Creating a Directory

Use the CREATE_DIRECTORY callback to create a directory. Note the following restrictions:

- You cannot create a top-level directory on the system disk.
- You cannot create a directory or a subdirectory beginning with the string SYS.

The following example creates a new directory under the system manager directory:

```
$ VMI$CALLBACK CREATE_DIRECTORY SYSTEM SYSMGR.CHECKTRAN
```

3.3 Summary of KITINSTAL Design Specifications

As a minimum requirement, your KITINSTAL.COM must do the following:

- Handle errors and Ctrl/Y interrupts.
- Process the request code passed to it in the first parameter (P1); it should exit with VMI$_UNSUPPORTED status if it does not recognize the request code.

  Currently, VMSINSTAL may pass one of three request codes:

  - VMI$_INSTALL to initiate the installation
  - VMI$_POSTINSTALL to initiate work required after the installation phase is complete
  - VMI$_IVP to initiate the IVP.

  However, because additional request codes may be implemented by VMSINSTAL in the future, your KITINSTAL should make explicit tests on P1. For example, do not assume that if VMSINSTAL is not passing VMI$_INSTALL or VMI$_POSTINSTALL, it must be passing VMI$_IVP.

  Also, note that subprocedures can be used to invoke KITINSTAL for services by passing the appropriate string in P1. For example, the ASK callback may use KITINSTAL to get some help text from the HELP_DEVICE subroutine by passing the string "HELP_DEVICE" in P1.
- Verify that it is compatible with the version of OpenVMS running on the target system.
- Determine whether the target system has sufficient free disk space for the product and for specified safety mode operations.
- Determine whether to retain old copies of files that have been replaced.
- Where applicable, restore secondary save sets to the kit working directory in the correct order.
- Determine whether the product has an IVP and, if so, whether the installer wants the IVP to run after the installation.
- Move the product's image files to the system directories using the PROVIDE_IMAGE callback.
- Update the DCL command tables (if applicable) so that the product can be invoked from the DCL level.
- Move any product help files to [SYSHLP].
- Move the product-specific startup command file to [SYS$STARTUP]; it should be invoked before running the IVP, where applicable.
- Move any product data files to the appropriate system directory.
- Return the KITINSTAL exit status to VMSINSTAL when the installation terminates.
- Where applicable, run the IVP and indicate to VMSINSTAL whether the IVP was successful.

Note

Currently, VMSINSTAL support for Digital's Distributed Software License Architecture (DDSLA) and License Management Facility (LMF) is not available to third-party (non-Digital) developers of layered products.

3.4 Basic KITINSTAL Command Procedure

Example 3–1 demonstrates a basic KITINSTAL.COM for a product called CHECKTRAN.

Example 3–1 Basic KITINSTAL.COM

```
!****************************************************************
!                                                               *
!               CHECKTRAN KITINSTAL.COM                          *
!                                                               *
!****************************************************************
$ !
$ !      Take care of interrupts
$ !
$ ON CONTROL_Y THEN VMI$CALLBACK CONTROL_Y
$ !
$ !        Process errors
$ !
$ ON WARNING THEN EXIT $STATUS
$ !
$ !      Determine course of action
$ !
$ IF P1 .EQS. "VMI$_INSTALL" THEN GOTO CHECKTRAN_INSTALL
$ IF P1 .EQS. "VMI$_POSTINSTALL" THEN GOTO CHECKTRAN_POSTINSTALL
$ IF P1 .EQS. "VMI$_IVP" THEN GOTO CHECKTRAN_IVP
$ EXIT VMI$_UNSUPPORTED
$ !
$ !  Set the product name (AXP Systems Only)
$ !
$ VMI$CALLBACK SET PRODUCT_NAME "CHECKTRAN"
$ !
$ !      Install the product
$ !
$CHECKTRAN_INSTALL:
$ !
$ !     Check that OpenVMS version is 5.0 or later for OpenVMS VAX,
$ !     or 1.0 for OpenVMS AXP.
$ !
$ !     For VAX systems only:
$ !
$ PRODUCT$VERSION = "5.0"
$ !
$ !     For AXP systems only:
$ !
$ PRODUCT$VERSION = "1.0"
$ VMI$CALLBACK CHECK_VMS_VERSION CHECKTRAN$VERSION 'PRODUCT$VERSION'
$ IF CHECKTRAN$VERSION THEN GOTO V_OK
```

(continued on next page)

Example 3–1 (Cont.) Basic KITINSTAL.COM

```
$ !
$ !               Indicate wrong version and exit
$ !
$ WRONG_VERSION:
$    VMI$CALLBACK MESSAGE E VERSION -
       "This kit must be installed on an existing OpenVMS''product$version'
        system."
$    EXIT VMI$_FAILURE
$ V_OK:
$ !
$ !         Check for disk space, exit if not enough
$ !

$ VMI$CALLBACK CHECK_NET_UTILIZATION CHECKTRAN_SPACE 1000 500 600
$ IF .NOT. CHECKTRAN_SPACE THEN EXIT VMI$_FAILURE

$ !
$ !         Set up for a conditional safe installation
$ !
$ VMI$CALLBACK SET SAFETY CONDITIONAL 3000
$ !
$ !         Restore secondary savesets
$ !
$ VMI$CALLBACK RESTORE_SAVESET B
$ VMI$CALLBACK RESTORE_SAVESET C N
$ !
$ !     Check for purging and for IVP
$ !
$ VMI$CALLBACK SET PURGE ASK
$ VMI$CALLBACK SET IVP ASK
$ !
$ !     Move the product image to [SYSEXE]
$ !
$ VMI$CALLBACK PROVIDE_IMAGE CHECKTRAN_IMAGE CHECKTRAN.EXE VMI$ROOT:[SYSEXE]
$ !
$ !     Make the product callable from DCL
$ !
$ VMI$CALLBACK PROVIDE_DCL_COMMAND CHECKTRAN.CLD
$ !
$ !     Move the help files
$ !
$ VMI$CALLBACK PROVIDE_DCL_HELP CHECKTRAN.HLP
$ !
$ !     Move the startup command
$ !

$ VMI$CALLBACK PROVIDE_FILE CHECKTRAN_STARTUP CHECKTRAN_STARTUP.COM -
  VMI$ROOT:[SYS$STARTUP]
$ !
$ !     Create IVP dirctory
$ !
$ VMI$CALLBACK CREATE_DIRECTORY COMMON SYSTEST.CHECKTRAN
$ !
$ !     Move IVP file
$ !
$ VMI$CALLBACK PROVIDE_FILE CHECKTRAN_ CHECKTRAN_IVP.COM VMI$ROOT:[SYSTEST]

$ !
$ !     Identify the startup command file
$ !

$ VMI$CALLBACK SET STARTUP CHECKTRAN_STARTUP.COM
```

(continued on next page)

Example 3–1 (Cont.) Basic KITINSTAL.COM

```
$ !
$ !     Move data file
$ !
$ VMI$CALLBACK PROVIDE_FILE CHECKTRAN_DATA CHECKTRAN.DAT VMI$ROOT:[SYSEXE]
$ !
$ !     Installation completed, exit
$ !
$ EXIT VMI$_SUCCESS
$ !
$ ! Post-install phase
$ !
$ CHECKTRAN_POSTINSTALL
$ !
$ !    No postinstall work
$ !
$ EXIT VMS$_SUCCESS
$ !
$ ! Verify installation
$ !
$ CHECKTRAN_IVP:
$ !
$ !     run the ivp
$ !

$ @VMI$KWD:CHECKTRAN_IVP.COM

$ !
$ !     ivp completed, indicate results
$ !
$ EXIT $STATUS
```

See Appendix B for a KITINSTAL command procedure for an actual Digital product.

4

VMSINSTAL Functional Description

VMSINSTAL is structured in a sequence of 11 functional steps. These steps are labeled in the command procedure. Two additional steps apply only to special installation conditions: Step 12 attempts to recover from system failures that occur during the installation, and Step 13 implements the GET save set option. This chapter provides a functional overview of VMSINSTAL and a summary of the 13 steps.

4.1 Overview

When you invoke VMSINSTAL, it announces itself, gives the time and date, and provides instructions for getting help if you need it during the installation. It also does the following initial tasks, all of which are transparent to the installer:

- Deletes all user DCL symbols.
- Assigns symbols and defines logical names needed to do the installation. These are all prefixed by VMI$.
- Verifies that you are running the installation from the SYSTEM account and that you have sufficient privileges and quotas.
- Verifies that there are no other users on the system.
- Checks the validity of parameters that you include in the calling command.

If any of the checks fail, VMSINSTAL displays an appropriate message.

If VMSINSTAL is called by the system startup procedure, indicating an aborted previous attempt at installation, the procedure branches to a special crash recovery routine (Step 12).

At this point, VMSINSTAL gives the installer the opportunity to back up the system disk before doing the installation. Then VMSINSTAL asks for the name of the device that will hold the distribution volumes during the installation, and verifies the validity of the device.

VAX

On VAX systems, if the distribution kit is to be mounted on the console device, VMSINSTAL connects it using the System Generation utility (SYSGEN), dismounts the console volume (if necessary), and then allocates the console device for the installation. VMSINSTAL also sets a flag to remount the console volume when the installation is completed.♦

On both AXP and VAX systems, VMSINSTAL next asks for the name of the products to be installed. If a product is distributed on tape or disk, VMSINSTAL asks you to physically mount the distribution volume and then waits for an indication that you are ready to continue.

Next, VMSINSTAL asks for any options that you might want to invoke.

VMSINSTAL then installs the kit and checks to see if any other kits are to be installed. The installation follows this general sequence:

1. VMSINSTAL sets up the environment needed to install the product, including the creation of a working directory for the installation procedure.
2. VMSINSTAL restores the first save set (save set A). This save set must include the KITINSTAL command procedure described in Chapter 3. If release notes are supplied with the product kit and you want to use option N, the release notes file must also be included in save set A.
3. VMSINSTAL calls KITINSTAL to direct the installation, and KITINSTAL in turn uses callbacks to VMSINSTAL to do the installation. Note that certain callbacks can be deferred until the installation is at a point where it is safe to do the callback functions. (See Chapter 5 for detailed descriptions of the VMSINSTAL callbacks.)
4. When all callbacks are completed, VMSINSTAL invokes the product's Installation Verification Procedure (IVP) if one exists.
5. Finally, VMSINSTAL cleans up the files created for doing the installation, remounts the console device, and restores the normal work environment. If the product requires rebooting the system, VMSINSTAL shuts down and reboots it.

Two special installation scenarios are detailed at the end of this chapter. One describes how VMSINSTAL recovers from system failures that may occur during the installation (Section 4.3.1); the other pertains to copying save sets into a disk directory for later installation (Section 4.3.2).

4.2 Functional Steps

The following sections describe in detail the functional steps executed by VMSINSTAL when it is invoked.

4.2.1 Step 1

In this step, VMSINSTAL provides overall initialization of the installation program. VMSINSTAL begins by setting up two symbols, one to represent the current version of VMSINSTAL (VMI$VERSION) and one to represent the booting option (VMI$BOOTING).

Assuming the program is not in booting mode, VMSINSTAL deletes all of the user's currently-defined global symbols to avoid any possible conflicts. VMSINSTAL executes SYS$MESSAGE:VMSINSTAL_LANGUAGE.COM to define symbols that have been equated to VMSINSTAL internal error messages. (For more information about SYS$MESSAGE:VMSINSTAL_LANGUAGE.COM, see Section 2.11.) VMSINSTAL then displays a welcome message that invites the installer to invoke help by entering a question mark (?) when needed.

After temporarily disabling the Ctrl/Y function, VMSINSTAL sets up the installation environment and records the current environment in related symbols. Then VMSINSTAL opens a file with the symbol definition VMI$TERMINAL_ FILE to read input from the terminal.

Next, VMSINSTAL assigns symbols for both internal use and the product installation procedures and equates a symbol called VMI$VMS_VERSION to the OpenVMS version running on the target system. See Appendix D for more information about the VMI$VMS_VERSION symbol.

After enabling Ctrl/Y, VMSINSTAL checks for the OPTION keyword in P3 before testing for the booting option. If the booting option (B) is active (VMI$BOOTING set true), VMSINSTAL is attempting to recover from a crash that occurred during a previous installation attempt. At this point, the procedure branches to step 12 to perform appropriate crash recovery procedures.

Then, VMSINSTAL determines whether the product is being installed in an alternate system root and sets up appropriate logical names (VMI$ROOT and VMI$SPECIFIC) as follows:

- VMI$ROOT represents the *common* top-level directory for the target system and, all references to common system directories must be in the form VMI$ROOT:[SYSxxx].
- VMI$SPECIFIC represents the *system-specific* top-level directory for the target system, and all references to system-specific directories must be in the form VMI$SPECIFIC:[SYSxxx].

Note

It is recommended that you use VMI$SPECIFIC or VMI$ROOT for all references to the system disk, rather than logical names prefixed with SYS$. Failure to do this will result in your product being installable only on the system disk.

If the file log option (L) is specified in P4, the /LOG qualifier is appended to a set of appropriate DCL command verbs.

VMSINSTAL then checks various environmental items to ensure that it can perform the installation. If the installer has opted for the GET save set option (G), these checks are omitted. The product is not going to be immediately installed, but the save sets are to be copied to a save-set directory.

The checks include the installer's account, privileges and quotas, and the status of other system processes. If necessary, after making the various checks, VMSINSTAL warns the installer of any unacceptable conditions and ask if the installer wants to continue. This warning is not issued if the installation is under internal Digital QA test.

Finally, after giving the installer an opportunity to back up the current system disk, VMSINSTAL goes to step 2.

4.2.2 Step 2

In this step, VMSINSTAL determines the location of the distribution volume, which may or may not be included as the second parameter (P2) in the command line. If the parameter is omitted, VMSINSTAL prompts for it using the ASK callback. VMSINSTAL also appends the colon (:) following the device designation if it is omitted.

VAX

On VAX systems, VMSINSTAL then tests the device designation to determine whether it is configured, and to establish the proper processing path. If the distribution volume is located on the console device, VMSINSTAL connects the console device using SYSGEN, dismounts the console volume if it is mounted, and allocates the console device for the installation. If the device is not properly configured, a message is generated saying that the device must be either a disk or tape drive.♦

On AXP and VAX systems, when VMSINSTAL determines that the device is configured and connected properly it finishes parsing P2 into the symbol VMI$PLACE.

4.2.3 Step 3

This step is a loop point for each product kit. When all of the products in a kit have been processed or if a kit installation is prematurely terminated, VMSINSTAL returns here to check for more product kits.

In step 3, VMSINSTAL determines the products to be installed and the proper installation order if there are multiple products in the kit. If a product is listed in the first parameter (P1) of the call to VMSINSTAL, it is equated to VMI$LIST; if not, the ASK callback is used to prompt for it. The installer can terminate the installation at this point by entering a Ctrl/Z or the word EXIT in response to the prompt.

VMSINSTAL then determines whether it needs to prompt the installer for options. VMSINSTAL only prompts for this information when *both* of the following conditions are true:

- Options are not specified on the command line.
- VMSINSTAL must also prompt for the **product_list** or **source device** parameters.

Next, VMSINSTAL checks to see if the distribution volume is mounted and, if it is not, prompts the installer to mount it. Then VMSINSTAL checks that the distribution device specification is correct before continuing. If either of these requirements is not met, the procedure branches back to step 2.

VMSINSTAL then checks the products on the distribution volume against the list of products entered by the installer. If the product's primary save set is not found, an appropriate message is displayed, the installation of the kit is aborted, and VMSINSTAL branches back to step 3 for the next product kit.

If the primary save set is found, VMSINSTAL loops to build a product list from the distribution volume. Following this, VMSINSTAL sorts the products alphabetically to build an ordered installation product list.

Next, the volume is dismounted and remounted for BACKUP operations, the actual installation mechanism. If the installer specified the GET save set option, VMSINSTAL branches to step 13; if not, it proceeds to step 4.

4.2.4 Step 4

In this step, VMSINSTAL sets up the environment for restoring the kit's save sets.

VMSINSTAL determines if the RSP (restore save set and pause) option was specified, and if this option specified an individual save set or all save sets.

VMSINSTAL also checks if the AWD (alternate working device) option was specified. If so, it defines the logical VMI$KWD to point to the specified device.

4.2.5 Step 5

This step is a loop point for obtaining listed products from the distribution volume.

The marker file is kept current to allow for recovery from system crashes. Information maintained in the marker file includes the following:

- Target system root (VMI$ROOT)
- Facility and version (VMI$PRODUCT)
- State B (before)
- Library being updated
- State of the alternate root flag (VMI$ALTERNATE_ROOT)
- State of the common root flag (VMI$COMMON_ROOT)
- Name of the system-specific root (VMI$SPECIFIC)
- Additional information necessary to recover from a crash.

VMSINSTAL ends step 5 by announcing the product that will be installed.

4.2.6 Step 6

In step 6, VMSINSTAL begins by deleting any previous kit working directory to make space for the product's working directory. It then records, in the symbol VMI$FREE_BLOCKS, the number of free blocks available on the target system disk. The symbol VMI$KWD_FREE_BLOCKS records the number of free blocks available on the disk used for the kit working directory.

Next, VMSINSTAL checks for and processes the statistics option (S), the auto-answer option (A), and the callback trace option (C). Last, VMSINSTAL creates the new kit working directory, before giving control to step 7.

If the installer has chosen the statistics option (S), VMSINSTAL displays the following message as it starts the statistics subprocess (referred to from here on as the statistics **demon**):

```
Waiting for demon to record initial state...
```

This subprocess monitors the installation and produces a statistics report. The subprocess is executed from VMI$ROOT:[SYSUPD] and is started with a process priority greater than the parent process. The statistics demon follows this sequence of steps:

1. Records the system directories, the number of free blocks available, and the current time.
2. Opens a statistics report file.
3. Sets the file retention on the target volume so that all file accesses are marked with an expiration date.
4. Loops, while monitoring the free block count, and returns control to the parent process.
5. When the installation is completed, takes control to turn off the file retention function and report the various block statistics.

6. Reports the files added, deleted, modified, and accessed during the installation.
7. When the report is completed, the demon returns control to VMSINSTAL.

VMSINSTAL then checks for the auto-answer option (A). If the A option was specified, VMSINSTAL checks for the existence of an auto-answer file for the product. If an auto-answer file exists, VMSINSTAL reads it for information relating to installing the product. Otherwise, VMSINSTAL creates an auto-answer file to record information relating to installing the product, and generates an appropriate message to the installer.

If the callback trace option (C) was selected, VMSINSTAL creates a file (SYS$UPDATE:facvvu.CBT) to contain the trace information.

4.2.7 Step 7

Step 7 uses the RESTORE_SAVESET callback to restore the product's primary save set (save set A) to the kit's working directory.

When the primary save set is restored, VMSINSTAL checks to see if it includes the KITINSTAL command procedure, and (if the installer selected option N when VMSINSTAL was invoked) it checks for the existence of a release notes file (fac_vvu.RELEASE_NOTES).

If the restoration fails or if the restored primary save set does not include the KITINSTAL command procedure, the installation is aborted. The program branches to step 11 for general housekeeping before looking for the next product to be installed.

4.2.8 Step 8

Step 8 begins by updating the marker file to indicate that the installation of the current product has begun and to establish the appropriate safety level.

Then, VMSINSTAL invokes KITINSTAL to install the product. If the installation succeeds, the program proceeds to step 9. If not, an appropriate message is displayed. The program branches to step 11 for general housekeeping before looking for the next product in the kit.

4.2.9 Step 9

Step 9 is used to execute any callbacks that were deferred if the installation was done in safety mode. VMSINSTAL then informs the installer that the deferred files are being moved to their target directories. The marker file is updated to indicate that the deferred callbacks are being executed.

Next, safety mode terminates, and the deferred callback file is opened for reading deferred callback records. If any deferred callback fails, the installation is aborted, an appropriate message is displayed, and the program prepares to install the next product.

If the deferred callbacks execute properly, the deferred callback file is closed, and the marker file is updated to indicate that the product installation is essentially complete.

If the product installation indicated a postinstall phase, then KITINSTAL.COM is invoked again so the procedure can do additional work after all files are in place.

4.2.10 Step 10

Step 10 begins by setting the default directory to the installation working directory. VMSINSTAL then removes the VMI$CALLBACK symbol to avoid any unintentional invocation of callbacks.

Next, the product-specific startup procedure is called, if applicable.

On completion of the startup procedure, VMSINSTAL invokes the IVP (if applicable). At the completion of the IVP, VMSINSTAL resets the default directory to MISSING:[MISSING] and redefines VMI$CALLBACK. VMSINSTAL reports the results of the IVP before going on to step 11.

4.2.11 Step 11

In Step 11, VMSINSTAL performs the following tasks, then branches to step 5 for the next product:

1. It closes the deferred callback file and deletes the kit's working directory.
2. If applicable, it pauses to wait for the completion of the statistics report, while displaying an appropriate message.
3. On completion of the statistics report, VMSINSTAL closes any deferred working files that may be open and deletes the marker file.
4. At this point, the procedure checks to see if the installed product requires a reboot of the system. If reboot is not required, the procedure branches to step 5 to get the next product from the list.
5. If a reboot is required, VMSINSTAL must terminate the installation even if some products have not been installed.
6. After displaying a message stating that the system will be rebooted, VMSINSTAL determines whether all products have been installed. If some products have not been installed, the procedure displays the message, "Products that have not been installed will be skipped."
7. VMSINSTAL then proceeds to the ALL_DONE subroutine.

4.2.12 All Done

The ALL_DONE subroutine provides the functions required for normal completion of a product kit installation. The subroutine begins by disabling the command interpreter error-checking function and then turning off the statistics generator, if applicable. After allowing time for the completion of the statistics report, VMSINSTAL closes any working files that may still be open, including the following:

- VMI$CALL_FILE
- VMI$DEFER_FILE
- VMI$MARKER_FILE
- VMI$PRODUCT_FILE
- VMI$TEMP_FILE

It then searches for and deletes any existing marker data files and restores the user's original file environment.

Following this, the distribution volume is dismounted. If the kit was installed from the console device, ALL_DONE remounts the console volume. The following files are closed after the console volume is remounted:

- VMI$AUTO_FILE
- VMI$TERMINAL_FILE

Next, ALL_DONE restores the rest of the user's environment that was saved at the beginning of the installation, including the following:

- Default directory (VMI$SAVED_DIR)
- Default protection (VMI$SAVED_PROT)
- UIC (VMI$SAVED_UIC)
- Privileges (VMI$SAVED_PRIVS)
- The system message format (VMI$SAVED_MSG)

If the last product installed requires a system reboot, ALL_DONE does a minimal shutdown after alerting the installer. It then executes the reboot.

VMSINSTAL returns a status in the DCL symbol $STATUS. If the installation failed, a failure status is returned; if it succeeded, a success status is returned.

As its final task, VMSINSTAL deassigns the installation-specific logical names (VMI$ROOT, VMI$SPECIFIC, VMI$KWD) if applicable, and displays the completion message.

4.3 Special Steps

Steps 12 and 13 are invoked only under certain conditions. The normal logical sequence has step 11 yielding control to ALL_DONE, which finishes the installation.

When the system is booting, if the startup procedures find a marker data file in SYS$UPDATE, it assumes that the system crashed during an installation and immediately invokes VMSINSTAL with the booting option (B). Step 1 tests for the booting option in symbol VMI$BOOTING and branches to step 12 if VMI$BOOTING is true.

VMSINSTAL is invoked with the GET save set option (G) if the installer chooses to copy the kit save sets rather than installing the kit. After determining the product list, step 3 tests the option parameter and immediately branches to step 13 if the G option is active.

4.3.1 Step 12

Step 12 is a special step that performs recovery procedures if the system crashes during an installation. It is called from the system startup procedures if a marker file is discovered during booting.

Step 12 begins by opening a file (VMI$TERMINAL_FILE) to read data input from the terminal. It then opens another file (VMI$MARKER_FILE) to read data from the marker data file (VMIMARKERpid.DAT). The data from this file is used to determine the manner in which the recovery will proceed. VMSINSTAL does this by assigning values, found in the marker file, to the following symbols:

- VMI$PRODUCT
- VMI$ALTERNATE_ROOT

- VMI$COMMON_ROOT
- VMI$SPECIFIC

After setting up the default directory and the product working directory, step 12 disables the VMSINSTAL purge function and turns off safety mode.

Next, an appropriate message is displayed, explaining the situation and providing directions for continuing (or discontinuing) the installation, depending on the state of the installation at the time of the crash, as follows:

- If the system disk was unchanged at the time of the crash, the following message is displayed:

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

- If insignificant changes have been made to the system disk, the following message is displayed:

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

- If a library was being updated when the crash occurred, the following message is displayed:

  ```
  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.
  ```

- If the installation was not in safety mode when the crash occurred, the following message is displayed:

  ```
  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.
  ```

- If the installation was nearly completed when the crash occurred, the following message is displayed:

  ```
  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.
  ```

  In this situation, step 12 opens the defer file to begin the deferred callback executions.

- If the installation was completed when the crash occurred, the following message is displayed:

  ```
  The installation was completed satisfactorily.
  ```

From here, VMSINSTAL proceeds to do a normal completion after displaying the following message:

```
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 deleting
certain layered product files.
```

4.3.2 Step 13

Step 13 is a special step that handles the GET save set option (G). This option is used when the installer does not want to install a product immediately. Instead, the product save sets are copied from the distribution volume into a disk directory where they are available for future installation.

When save sets are copied using the G option, the directory structure originally assigned to files within the save set are not maintained. Because the directory structure is lost, this option cannot be used to copy OpenVMS operating system kits.

First, step 13 determines the target directory; that is, the directory into which the product save sets are to be copied. It then verifies that the disk or magnetic tape is mounted and ready. If the installer omits the target directory in P5 of the VMSINSTAL command line, Step 13 prompts for it. If no such directory exists, step 13 provides a suitable display and prompts for the proper directory.

Then the procedure parses the answer to determine the full device specification. If necessary, the procedure creates a top-level work directory, [VMIWORKpid], where *pid* is the current process ID. This directory is created on the system disk to restore the save sets, so that new save sets can be created in the target directory.

Following this, the installer is told to ignore redundant BACKUP error messages:

```
Because VMSINSTAL does not know how many save sets 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 save sets have
been copied.
```

Next, a loop is used to copy the product save sets for each product in the product list, appending any additional BACKUP qualifiers specified in P6. The two qualifiers appended by VMSINSTAL are /INTERCHANGE and /VERIFY.

The loop begins by telling the installer which product is being copied. Then each of the product save sets is copied to the target directory. The loop concludes by indicating the number of save sets copied for the product.

If an error occurs while any product save set is being copied, the contents of the scratch directory are deleted, and the program branches to an appropriate error-handling routine before looking for the next product. When all of a product's save sets are copied, VMSINSTAL deletes the product work directory and branches to step 3 for more products.

If an error occurs while any product save set is being copied, the following message is returned at the end of the installation. This message is displayed regardless of whether any products were copied successfully:

```
Installation terminated due to unexpected event.
```

5
VMSINSTAL Callbacks

This chapter describes the VMSINSTAL callbacks and gives guidelines for using them in your installation command procedures. The callbacks are presented in alphabetical order.

Following are rules and guidelines for using VMSINSTAL callbacks:

- Add callbacks only to KITINSTAL.COM or to files directly invoked from it.
- Specify keywords in uppercase. Do not abbreviate them.
- Begin all logical names to be defined by the callback with a prefix that consists of your product facility code followed by an underscore (for example, CHECKTRAN_). If the logical name is never referenced after being specified in the callback, use only the prefix.
- You must refer to the kit's working directory using the logical name VMI$KWD. Refer to system directories using the logical name VMI$ROOT and an explicit directory, such as VMI$ROOT:[SYSLIB].

  When referring to other directories, you must specify either an explicit device logical name (not a device number) and a directory, or a logical name for the entire file specification. Any referenced disk must be mounted and must include the specified directory.
- When a callback parameter requires only the file name and type, it automatically includes the device and directory portions of the file specification. In these instances, you are not allowed to specify the device or the directory, and you cannot use logical names or wildcards.
- When a callback parameter requires a complete file specification (disk, directory, name, and type), you can use a logical name. Use of wildcards is not allowed.
- When a parameter permits multiple options represented by single-character codes, do not use embedded spaces.
- All callbacks return an exit status. VMI$_SUCCESS is returned if the callback executed without error. VMI$_FAILURE is returned if errors occurred.
- Begin each invocation of a callback with the symbol VMI$CALLBACK, followed by the callback's name as the first parameter.
- Separate all parameters with spaces.

The following is an example of a command line for invoking a VMSINSTAL callback:

```
$ VMI$CALLBACK FIND_FILE CHECKTRAN_ VMI$ROOT:[SYSEXE]FED.EXE "" S CHECKTRAN_V1FED
```

See Example 3–1 and Example B–1 for KITINSTAL command procedures that demonstrate the use of callbacks.

Note

Currently, VMSINSTAL callbacks related to Digital's Distributed Software License Architecture (DDSLA) and License Management Facility (LMF) are not supported for third-party (non-Digital) developers of layered products.

5.1 ADD_IDENTIFIER Callback

The ADD_IDENTIFIER callback adds an identifer to the rights database. You can use this callback with the SET ACL DEVICE, SET ACL DIRECTORY, and SET ACL FILE callbacks to allow access to devices and files. See Section 5.28.1 for information on the SET ACL callback options.

See the *OpenVMS System Management Utilities Reference Manual* for more information about identifiers.

Use the following command line format to invoke the ADD_IDENTIFIER callback:

VMI$CALLBACK ADD_IDENTIFIER id_name qualifiers

Parameters on the command line indicate the following:

id_name
Use this parameter (P2) to specify the name of the identifier you are creating.

qualifiers
Use this parameter (P3) to specify qualifiers. You can specify any qualifiers of the ADD/IDENTIFIER command in the Authorize utility. Enclose a list of qualifiers in quotation marks.

Following is an example of a command line that invokes the ADD_IDENTIFIER callback:

```
$ VMI$CALLBACK ADD_IDENTIFIER nodea$ident "/ATTRIBUTES=(RESOURCE)"
```

In the previous example, the ADD_IDENTIFIER callback adds identifier **nodea$ident** and marks it as a resource.

The ADD_IDENTIFIER callback returns VMI$_SUCCESS if the identifier is successfully added; otherise, the callback returns VMI$_FAILURE.

5.2 ASK Callback

Use the ASK callback to get information from the installer. The callback prompts the installer for a response, verifies and evaluates it, and then assigns it to the global symbol you specify. The callback also provides help when the installer presses the question mark key.

Use the following command line format to invoke the ASK callback:

VMI$CALLBACK ASK symbol prompt [default_response] [options] [help]

Parameters on the command line indicate the following:

symbol
Use this parameter (P2) to specify a global symbol that is equated to the installer's response. It is a numeric value if the response is integer or Boolean data; it is a string value if the response is string data. In formatting the installer's response, the callback automatically reduces multiple blanks to a single blank, removes all leading and trailing blanks, removes comments, and changes the response to the required case. Uppercase is the default.

prompt
Use this parameter (P3) to specify in a quoted string the question to be asked. The prompt should not indicate the default value of the response. Do not include trailing spaces, a backslash (\ ), a colon (:), or a question mark (?) because these are added automatically.

default_response
Where applicable, use this parameter (P4) to specify a default response to a prompt. For example, you might specify the letter *Y* as an acceptable default response to a prompt seeking a positive Boolean data-type response.

If you do not want to specify a default response, enter a null string ("") in this parameter.

If you specify a default response, the installer accepts it by pressing the Return key. If you do not specify a default response, the callback will continue to prompt for a response until the installer makes a valid entry.

options
Where applicable, use this parameter (P5) to specify options, choosing the appropriate letters from the following list. If you specify more than one option, do not enter spaces between the selected letters. For example, to ring a bell at the installer's terminal and to require a Boolean data-type response, enter the characters *RB*.

Note that options U, L, and M control the case in which input is returned from the installer (for example, uppercase or lowercase). These options override the previous default case, including any default set by the SET ASK_CASE callback. After this callback executes, the default case returns to the previous value.

Following are options:

- A—Turns auto-answer off; does not record prompts and responses in the auto-answer file. This option always requires input from the installer.
- B—The answer must be in Boolean form, *YES* or *NO* (*Y* or *N* is also acceptable); if it is not, the question repeats until the installer responds with a Boolean entry.
- D—Generates a blank line immediately preceding the prompt.
- E—Turns terminal echo off for this prompt. Additionally, this option disables writing of the prompt and response to an auto-answer file if one is being created. This option always requires input from the installer. If reading from an auto-answer file, this option causes a read timeout if a response is not received within a predetermined time. Responses are thereby handled as classified data.

- H—Displays appropriate help text before displaying the prompt.
- I—The answer must be an integer; if it is not, the question repeats until the installer responds with an integer entry.
- L—Returns input from the installer in lowercase.
- M—Returns input from the installer in the same case entered by the installer.
- N—A null string response is acceptable; applies to situations where a string response is requested but no default value is specified.
- R—Rings the bell before displaying the prompt.
- S—The response can be any character string; this is the default response data-type.
- U—Returns input from the installer in uppercase.
- Z—Allows the installer to respond with Ctrl/Z. The return value is the two-character string ^Z.

help

Use this parameter (P6) to specify help text that may be used in conjunction with the prompt. Either you can enter actual help text, or you can enter a call to a command procedure (typically KITINSTAL) to provide the text by way of a help callback. If you enter a call to a command procedure, you must also specify the name of the help callback that is found in P1 (request code) of the call, as follows:

```
@VMI$KWD:KITINSTAL HELP_WHATEVER
```

There is no conflict with VMSINSTAL request codes because the request code is not prefixed with the string VMI$.

To illustrate the use of the ASK callback, following is an example of a typical command line that includes all six parameters:

```
              P1   P2           P3           P4 P5       P6
               \    \            \            \ \         \
$ VMI$CALLBACK ASK TST_DIGIT "Enter a number" 3 I "@VMI$KWD:KITINSTAL help_type"
```

In response to the preceding command line, the callback displays the following prompt:

```
Enter a number [3]:
```

If the installer enters ?, the following help message appears:

```
Type a digit:
```

The global symbol TST_DIGIT (P2) is assigned the value of the entered integer. If the installer presses the Return key, TST_DIGIT is assigned the default integer value (3).

If the installer enters a data type other than integer, the callback repeats the original prompt; that is:

```
Enter a number [3]:
```

The ASK callback returns VMI$_SUCCESS unless the installation is using an auto-answer file and encounters a prompt mismatch. The auto-answer file includes both the prompt and the proper response. If the prompt entered in

P3 of the callback command line does not match the prompt in the auto-answer file, a fatal error condition exists. The installation terminates, and the installer's terminal displays the following message:

```
Auto-answer file is not in synch with questions.
```

In addition to this message, the callback displays the mismatched prompts; that is, the prompt in P3 of the command line and the prompt in the auto-answer file.

5.3 CHECK_NETWORK Callback

Use the CHECK_NETWORK callback to determine whether the network is running. The installation can then either proceed, or terminate, based on the status of the network. Products should display messages that indicate whether the network should be running while performing the installation.

Use the following command line format to invoke the CHECK_NETWORK callback:

VMI$CALLBACK CHECK_NETWORK symbol

The parameter on the command line indicates the following:

symbol
Use this parameter (P2) to specify a global symbol whose value reflects the status of the network. This symbol returns a value of true (1) if the network is running and false (0) if the network is not running.

Following is an example of a command line that invokes the CHECK_NETWORK callback:

```
$ VMI$CALLBACK CHECK_NETWORK TST_NETSTAT
```

In the previous example, the CHECK_NETWORK callback defines the symbol TST_NETSTAT as true when the network is running and false when the network is not running.

The CHECK_NETWORK callback always returns VMI$_SUCCESS.

5.4 CHECK_NET_UTILIZATION Callback

The CHECK_NET_UTILIZATION callback determines whether the peak number of free blocks on the VMI$ROOT device is sufficient to successfully complete the installation and then returns the result in a global Boolean symbol. You can obtain the net block usage for a product by using the statistics option when you invoke VMSINSTAL. The net block usage refers to the total number of blocks required by the installation.

When a product is installed with the alternate working device option (AWD), the installer specifies an alternate device for the creation of the temporary working area. (The temporary working area is the location where temporary files are created and save sets are restored.) The CHECK_NET_UTILIZATION callback can be used in conjunction with the AWD option, to specify the following:

- The number of free blocks that must be available on the device associated with the alternate working directory.
- The number of free blocks that must be available to complete the installation on the target device (the device that will hold the images and command files associated with the product).

If you choose the highest safety level for your installation, the disk must have sufficient space to accommodate the highest number of blocks used at one time (peak utilization). (See Section 5.28.9 for more information on the SET SAFETY option.)

Use the following command line format to invoke the CHECK_NET_ UTILIZATION callback:

VMI$CALLBACK CHECK_NET_UTILIZATION symbol blocks [trg-blocks] [awd-blocks]

Parameters on the command line indicate the following:

symbol
Use this parameter (P2) to specify a global symbol that the callback uses to indicate whether the disk or disks have sufficient net free space to install your product. The callback returns the value true (1) for this symbol if there are sufficient net free blocks available; otherwise, its value is false (0).

blocks
Use this parameter (P3) to specify the number of net free blocks required for your product. The figure should be equal to the total number of blocks required to perform the installation, including the temporary working directory, on the target device.

trg-blocks
Use this parameter (P4) to specify the number of free blocks that must be available on the target device (VMI$ROOT) in order to accommodate the completed installation. This number does not include the number of blocks required for the temporary working area; this number is specified in P5. The values specified in P4 and P5 are used only when the product is installed using the alternate working device (AWD) option.

awd-blocks
Use this parameter (P5) to specify the number of free blocks that must be available on the device associated with the alternate working directory.

Following is an example of the command line used to invoke the CHECK_NET_ UTILIZATION callback:

```
$ VMI$CALLBACK CHECK_NET_UTILIZATION CHECK_ 20000 12000 10000
```

The preceding callback can function in the following ways, depending on whether the installer specifies the AWD option:

- If the installer does not specify the AWD option, the callback returns the value true (1) in the symbol CHECK$ when the system has at least 20000 free blocks on the VMI$ROOT device. If the system does not have at least 20000 free blocks, the callback returns a value of false (0).
- If the installer specifies the AWD option, the callback returns the value 1 (true) when the VMI$ROOT device has at least 12000 blocks, and the alternate working device has at least 10000 blocks. If either of these devices has less than the specified number of free blocks, the callback returns a value of 0 for the CHECK$ symbol.

This callback always returns VMI$_SUCCESS.

5.5 CHECK_PRODUCT_VERSION Callback

Use the CHECK_PRODUCT_VERSION callback to check the version of another layered product installed on the system. VMSINSTAL extracts the version number from the image file identification string of the image file associated with the product. VMSINSTAL then compares this version number to the minimum version specified in the command line.

You can use this callback if your product requires the existence of a specific version of another product.

Use the following command line format to invoke the CHECK_PRODUCT_VERSION callback:

```
VMI$CALLBACK CHECK_PRODUCT_VERSION symbol filespec minimum_version [option]
```

Parameters on the command line indicate the following:

symbol
Use this parameter (P2) to specify a global symbol that returns a value of true (1) when the product meets the minimum version requirement specified in P4. When the product does not meet the minimum version requirement, this symbol returns a value of false (0).

filespec
Use this parameter (P3) to indicate the full file specification of the prerequisite product's image file. This is the file from which VMSINSTAL extracts the version number.

minimum_version
Use this parameter (P4) to specify the minimum version required to install your product. Specify minimum version using the following format:

tvv.u-m

where:

t	is the type of release (for example V for released-version, T for field test version)
vv	is the major version number
u	is the update number
m	is the maintenance number

For example, you might specify the following minimum version: V3.0-5.

option
Use this parameter (P5) to specify the name of the product for which you are performing the version check. If the product does not meet the minimum version requirement specified in P4, VMSINSTAL displays the following message to the installer:

```
This kit requires at least:  product_name minimum_version
Please install a proper version of product_name
before installing this product.
```

Enclose the product name with quotation marks.

Following is an example of a command line that invokes the CHECK_PRODUCT_VERSION callback:

```
$ VMI$CALLBACK CHECK_PRODUCT_VERSION TST_CHECK VMI$ROOT:[SYSEXE]RDO.EXE V3.0 "OpenVMS AXP Rdb"
```

In the previous example, VMSINSTAL determines whether the RDO image is version 3.0 or higher. If the image is version 3.0 or higher, VMSINSTAL defines symbol TST_CHECK as true.

This callback always returns VMI$_SUCCESS.

5.6 CHECK_VMS_VERSION Callback

Use the CHECK_VMS_VERSION callback to limit the installation of a product to specified versions of the OpenVMS operating system. When invoked, this callback tests the version of the running system against the minimum and maximum versions necessary to install the product. If the version number of the running system does not meet the specified criteria, notify the installer. To do this, use the MESSAGE callback to display a message that specifies the current OpenVMS version of the installing system and the OpenVMS version necessary to install the product. Section 5.16 describes how to use the MESSAGE callback to display a message.

Digital recommends using the CHECK_VMS_VERSION callback instead of the VMI$VMS_VERSION symbol to test the version of the operating system.

This callback is useful when the developer knows that specific versions of OpenVMS will break the product.

Use the following command line format to invoke the CHECK_VMS_VERSION callback:

VMI$CALLBACK CHECK_VMS_VERSION symbol minimum_version [option][maximum_version]

Parameters on the command line indicate the following:

symbol
Use this parameter (P2) to specify a global symbol that VMSINSTAL defines as true (1) when the running version meets the criterion specified in P3 (minimum_ version) and P5 (maximum_version). When the running system does not meet the criteria, VMSINSTAL defines this symbol as false (0).

minimum_version
Use this parameter (P3) to specify the minimum OpenVMS version required to install the product. Specify minimum version in the following format:

vv.u-m

where:

vv	indicates a version
u	indicates an update
m	indicates a maintenance release

For example, 5.2-1 would indicate that the version is 5, the update is 2, and the maintenance level is 1. To specify Version 5.2, you would enter 5.2.

This parameter continues to accept the old format, VVU (for example, you can specify 052 to indicate Version 5.2); however, to specify a particular maintenance release, you must use the new format. Digital recommends you use the new format whenever possible.

option
The following options are currently available for P4:

- F—Limits the product installation to a specific field test version. This option is intended for use only if a layered product is conducting field test in conjunction with an OpenVMS field test. If you choose option F, use P3 to specify the field test version to which the product installation is limited.
- S—Suppresses message output during installation. This option is intended for range checking of the OpenVMS version; messages are not output to the screen.

maximum_version
Use this parameter, where appropriate, to specify the maximum OpenVMS version required to install the product. Specify maximum version in the following format:

vv.u-m

where:

vv	indicates a version
u	indicates an update
m	indicates a maintenance release

For example, 5.2-1 would indicate that the version is 5, the update is 2, and the maintenance level is 1. To specify Version 5.2, you would enter 5.2.

This parameter continues to accept the old format, vvu (for example, you can specify 052 to indicate Version 5.2); however, to specify a particular maintenance release, you must use the new format. Digital recommends you use the new format whenever possible.

The following is an example of the command line for the CHECK_VMS_VERSION callback:

```
$ VMI$CALLBACK CHECK_VMS_VERSION for_version 5.0 "" 5.2
```

In this example, the CHECK_VMS_VERSION callback limits product installation to OpenVMS Versions 5.0 through 5.2.

This callback always returns VMI$_SUCCESS.

5.7 COMPARE_IMAGE Callback

The COMPARE_IMAGE callback compares the image file identification string of two image files and returns a value that indicates which file is a more recent version. The image identification is specified in the following format:

fac tvv.u-m

where:

fac	is the product's registered facility name (for example, RDB)
t	is the type of release (for example V for released-version, T for field test version)
vv	is the major version number
u	is the update number
m	is the maintenance number

Use the following command line format to invoke the COMPARE_IMAGE callback:

VMI$CALLBACK COMPARE_IMAGE symbol filespec1 filespec2

Parameters on the command line indicate the following:

symbol
Use this parameter (P2) to specify a global symbol that returns one of the following values:

- VMI$K_KIT_VER_OLDER—indicates that the image file specified in P4 is an older version than the image file specified in P3
- VMI$K_KIT_VER_NEWER—indicates that the image file specified in P4 is a newer version than the image file specified in P3
- VMI$K_KIT_VER_SAME—indicates that the image files specified in P3 and P4 are the same version
- VMI$K_KIT_VER_NOMATCH—indicates that the facility names of the files specified in P3 and P4 do not match

filespec1
Use this parameter to indicate the full file specification of the image file currently installed on the system.

filespec2
Use this parameter to indicate the full file specification of the image file that is part of the kit.

The following is an example of a command line that invokes the COMPARE_IMAGE callback:

```
$ VMI$CALLBACK COMPARE_IMAGE TST_CHECK VMI$ROOT:[SYSEXE]RDO.EXE VMI$KWD:RDO.EXE
```

The previous example compares two RDO images to determine which is the newest version and defines symbol TST_CHECK accordingly.

This callback returns VMI$_SUCCESS if the file specified in P4 is found. Otherwise, it returns VMI$_FAILURE.

5.8 CONTROL_Y Callback

This callback, invoked without parameters, performs housekeeping chores that must be invoked when the installer presses Ctrl/Y.

Note

The CONTROL_Y callback returns a warning status, which executes your ON WARNING statement. If you omit this statement in your installation command procedure and the installer presses Ctrl/Y, results are unpredictable.

The installation procedure must either include an ON WARNING statement, or it must provide a branch to a label that performs cleanup and then invokes the CONTROL_Y callback. The following is an example of the required command line:

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

5.9 CREATE_ACCOUNT Callback

The CREATE_ACCOUNT callback creates a new account in SYSUAF.DAT (and in NETPROXY.DAT if it is a proxy account). You are not permitted to use this callback if you are installing the product in an alternate directory root.

In most instances, new software products do not need to create new accounts. For example, it is usually unnecessary to create a system management account for a product, because the system manager can perform all management from the standard SYSTEM account.

Your installation procedure should prompt the installer for the UIC of the new account before you invoke this callback.

Use the following command line format to invoke the CREATE_ACCOUNT callback:

VMI$CALLBACK CREATE_ACCOUNT username qualifiers

Parameters on the command line indicate the following:

username
Use this parameter (P2) to associate a user name with the new account.

qualifiers
Use this parameter (P3) to list the qualifiers of the ADD command in the Authorize utility. (See the *OpenVMS System Management Utilities Reference Manual* for more information on the qualifiers to the DCL command ADD.) Enclose a list of qualifiers with quotation marks.

The following is an example of the command line for the CREATE_ACCOUNT callback:

```
$ VMI$CALLBACK CREATE_ACCOUNT SMITH/PASSWORD=FCDREG/UIC=[360,103]"
```

The CREATE_ACCOUNT callback returns VMI$_SUCCESS if the account is successfully created; otherwise, it returns VMI$_FAILURE.

5.10 CREATE_DIRECTORY Callback

Use the CREATE_DIRECTORY callback to create system directories, system-specific directories, common directories, user directories for your product.

System directories are created in the common area and the specific area. System-specific directories are created in the system root of the installing system (not the common area). Common directories are created in the common area only.

The prefix *SYS* is reserved and should not be used to specify a product directory.

The way you invoke the callback depends on the type of directory you are creating.

5.10.1 Creating a System Directory

To create a system directory, use the following command line format:

VMI$CALLBACK CREATE_DIRECTORY SYSTEM name [qualifiers]

Parameters on the command line indicate the following:

SYSTEM
When you enter the keyword SYSTEM in this parameter (P2), the product directory is created under the system root directory and the common directory.

name
Use this parameter to specify the name of the system directory. The callback recognizes the keyword SYSTEM and makes it a rooted directory. Therefore, do not specify the system device designation, and do not include the brackets usually used in directory name specifications.

qualifiers
Use this parameter (P4) to specify one or more of the CREATE/DIRECTORY command qualifiers: /OWNER_UIC, /PROTECTION, /VERSION_LIMIT. The entire parameter must be specified as a character string; that is, within quotation marks.

For example, to create a system directory called NEWPRODUCT, enter the following command line:

```
$ VMI$CALLBACK CREATE_DIRECTORY SYSTEM NEWPRODUCT "/OWNER_UIC=[SMITH]"
```

The callback displays this message:

```
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):

$ CREATE /DIRECTORY SYS$SPECIFIC: ...
```

This callback always returns VMI$_SUCCESS.

5.10.2 Creating a System-Specific Directory

To create a system-specific directory, use the following command line format:

VMI$CALLBACK CREATE_DIRECTORY SPECIFIC name [qualifiers]

Parameters on the command line indicate the following:

SPECIFIC
When you enter the keyword SPECIFIC in this parameter (P2), the product directory is created in the system-specific root for the installing system. The directory is not created in the common area but is created in the system root on the disk for the installing system.

name
Use this parameter to specify the name of the system-specific directory. The callback recognizes the keyword SPECIFIC and makes the directory a rooted directory. Therefore, do not specify the system device designation, and do not include the brackets usually used in directory name specifications.

qualifiers
Use this parameter (P4) to specify one or more of the CREATE/DIRECTORY command qualifiers: /OWNER_UIC, /PROTECTION, /VERSION_LIMIT. The entire parameter must be specified as a character string enclosed by quotation marks.

For example, to create a system-specific directory called OLDPRODUCT, enter the following command line:

```
$ VMI$CALLBACK CREATE_DIRECTORY SPECIFIC OLDPRODUCT "/OWNER_UIC=[SMITH]"
```

In most cases, Digital recommends you create user directories rather than system-specific directories. Create system-specific directories only where absolutely necessary.

This callback always returns VMI$_SUCCESS.

5.10.3 Creating a Common Directory

To create a system directory in the common area only, use the following command line format:

VMI$CALLBACK CREATE_DIRECTORY COMMON name [qualifiers]

Parameters on the command line indicate the following:

COMMON
When you enter the keyword COMMON in this parameter (P2), the product directory is created under the system common area only.

name
Use this parameter to specify the name of the directory. The callback recognizes the keyword COMMON and makes the directory a rooted directory. Therefore, do not specify the system device designation, and do not include the brackets usually used in directory name specifications.

qualifiers
Use this parameter (P4) to specify one or more of the CREATE/DIRECTORY command qualifiers: /OWNER_UIC, /PROTECTION, /VERSION_LIMIT. The entire parameter must be specified as a character string enclosed by quotation marks; for example, you might specify "/OWNER_UIC=[SMITH]".

The following is an example of a command line that creates the directory [VMS$COMMON.MYPROD.LOGS]:

```
$ VMI$CALLBACK CREATE_DIRECTORY COMMON MYPROD.LOGS
```

This callback always returns VMI$_SUCCESS.

5.10.4 Creating a User Directory

To create a user directory, use the following command line format:

VMI$CALLBACK CREATE_DIRECTORY USER name [qualifiers]

Parameters on the command line indicate the following:

USER
When you enter the keyword USER in this parameter (P2), you must specify whether the directory is being created on the system disk or a user disk.

Since most optional software products are associated with a particular system root and should reside in that root, avoid creating product directories on user disks unless absolutely necessary.

name
Use this parameter to specify the user directory. The device portion of the specification must be a valid device specification.

qualifiers
Use this parameter (P4) to specify one or more of the CREATE/DIRECTORY command qualifiers: /OWNER_UIC, /PROTECTION, /VERSION_LIMIT. The entire parameter must be specified as a character string; that is, within quotation marks.

For example, to create a directory called USER1 on user disk DISK1, enter the following command line:

```
$ VMI$CALLBACK CREATE_DIRECTORY USER DISK1:[USER1] "/PROTECTION=(S:RD,O:RW)"
```

If you specify the VMI$ROOT value for the user directory, the callback alerts the installer by displaying the following message:

```
This product creates system disk directory ...
```

This callback always returns VMI$_SUCCESS.

5.11 DELETE_FILE Callback

The DELETE_FILE callback deletes all versions of the specified file, typically an obsolete file created by a previous installation. You may need to set privileges or change the protection to delete certain files because the BYPASS privilege is not enabled for callbacks.

Note

Do not include the version number of the specified file name. All versions of a file are deleted.

If the callback is invoked while the installation is in safety mode, the command line is written to the defer file for subsequent execution.

If you are deleting a file that has been installed with the Install utility, the callback uses the Install utility to delete the image from the *known file* data base. (See the *OpenVMS System Management Utilities Reference Manual* for more information.)

Use the following command line format to invoke the DELETE_FILE callback:

```
VMI$CALLBACK DELETE_FILE filename
```

Parameters on the command line indicate the following:

filename
Use this parameter (P2) to enter the complete file specification (device, directory, name, and type) of the file to be deleted. Wildcard characters may be used in the filename.

This callback returns VMI$_SUCCESS unless it is unable to find the specified file, in which case a VMI$_FAILURE status is returned.

5.12 FIND_FILE Callback

VMSINSTAL uses the FIND_FILE callback to locate files either in a system directory or in the installation working directory. Typically, this callback is used to assign a logical name to the located file or to indicate whether the file exists and, if so, where it is located.

Files in the kit's working directory (VMI$KWD) may be referenced without using the FIND_FILE callback, but any attempt to reference a system file without using a callback is likely to fail.

Use the following command line format to invoke the FIND_FILE callback:

```
VMI$CALLBACK FIND_FILE logical filespec [default-spec] locate [symbol]
```

Parameters on the command line indicate the following:

logical
Use this parameter (P2) to assign a process logical name to the located file and use this logical name in all subsequent references to the file.

filespec
Use this parameter (P3) to specify, either fully or partly, the file to be located.

default-spec
Where applicable, use this parameter (P4) to provide a default for partially specifying the file to be located. Typically, this parameter provides a default value for the device, directory, and type parts of the file specification. The parameter value is then used with the previous parameter (P3) value to derive the full file specification. If you do not use this parameter, be sure to specify a null string ("") in P3.

locate
Use this parameter (P5) to specify how the file is to be located by choosing the appropriate characters from the following list. If you list more than one character, do not use any intervening spaces.

- W—Look in the kit's working directory for a file with the matching name and type.
- S—Look in the directory specified by the file specification (P3) and default (P4) parameters.
- E—If the file is not found, the callback displays an error message and returns VMI$_FAILURE status.
- O—Look in the system-specific directory.

symbol
Where applicable, use this parameter (P6) to assign a symbol whose return value will be a single-character code from the following list:

- W—The file was found in the kit's working directory.
- S—The file was found in the specified directory.
- E—The file was not found and an error was reported.
- ""— None of the above.

The following is an example of the command line of the FIND_FILE callback:

```
$ VMI$CALLBACK FIND_FILE SYSUAF VMI$ROOT:[SYSEXE]SYSUAF.DAT "" SE
```

In this example, the callback is used to search for the SYSUAF file in the system area. A default device, directory, and type are provided in P4 and the file is assigned the logical name SYSUAF. P5 returns an error message sequence if the file is not found.

Note that you can also use the logical name VMI$FIND to invoke the FIND_FILE callback. VMSINSTAL equates VMI$FIND to VMI$CALLBACK FIND_FILE, as illustrated in the following command line:

```
$ VMI$FIND SYSUAF VMI$ROOT:[SYSEXE]SYSUAF.DAT SE
```

If the specified file is found, or if no error message is returned when the file is not found, the callback exits with VMI$_SUCCESS. If the file specification parameters, P3 and P4, cannot be parsed, or if the file is not found and an error message is returned, the callback exits with VMI$_FAILURE.

5.13 GET_IMAGE_ID Callback

The GET_IMAGE_ID callback extracts the image file identification string data for a file.

Use the following command line format to invoke the GET_IMAGE_ID callback:

VMI$CALLBACK GET_IMAGE_ID symbol filespec

Parameters on the command line indicate the following:

symbol
Use this parameter (P2) to specify a global symbol that VMSINSTAL equates to the image file identification string from the image header section.

filespec
Use this parameter (P3) to specify the full file specification of the file for which you want the file identification string.

The following is an example of a command line that invokes the GET_IMAGE_ID callback:

```
$ VMI$CALLBACK GET_IMAGE_ID TST_IMAGE_ID VMI$ROOT:[SYSEXE]RDO.EXE
```

This callback returns VMI$_SUCCESS if the file is found; if the file is not found, it returns VMI$_FAILURE.

5.14 GET_PASSWORD Callback

The GET_PASSWORD callback obtains a system generated or installer specified password. The password is returned through a global symbol. Passwords are not echoed on the installer's terminal.

This callback should be used in the question portion of the product installation procedure.

Use the following command line format to invoke the GET_PASSSWORD callback:

```
VMI$CALLBACK GET_PASSWORD symbol [keyword] [min-size]
```

Parameters on the command line indicate the following:

symbol
Use this parameter (P2) to specify a global symbol that VMSINSTAL defines with the obtained password. If a password cannot be obtained, the symbol is defined as the null string.

keyword
Use this parameter (P3) to make the appropriate selection:

- AUTO—Use this keyword to indicate that you want the system to generate the password to be used. The password selected is not made known to the installer. This is the default if you do not specify a keyword.
- SPECIFY—Use this keyword to indicate that you want the installer to specify the password to be used. The prompt and password will not be entered into the auto-answer file if one is being created. This prompt always requires input.

min-size
Use this parameter (P4) to specify the minimum length of the password. For AUTO, this must be a number from 1 to 10; the default value is 10. For SPECIFY, this must be a number from 1 to 31; the default value is 8.

The GET_PASSWORD callback always returns VMS$_SUCCESS.

5.15 GET_SYSTEM_PARAMETER Callback

The GET_SYSTEM_PARAMETER callback returns the current value of any system (SYSGEN) parameter by using the F$GETSYI lexical function. (See the *OpenVMS System Management Utilities Reference Manual* for a list of system parameters, and the *OpenVMS DCL Dictionary* for more information on the F$GETSYI lexical function.)

Use the following command line format to invoke the GET_SYSTEM_ PARAMETER callback:

```
VMI$CALLBACK GET_SYSTEM_PARAMETER symbol/filename_type name [option]
```

Parameters on the command line indicate the following:

symbol/filename_type
Use this parameter (P2) to equate a global symbol to the value of the SYSGEN parameter. If you specify the T option in P4, use parameter (P2) to specify a file name.

name
Use this parameter (P3) to specify the full name of the SYSGEN parameter being evaluated. If you specify the T option in P4, enter a null string ("") in P3.

option
Where applicable, use this parameter (P4) to specify options by entering the appropriate option letter. Currently, the only available option for this callback is the T option.

Use the T option to supply an input file (filename_type in P2) containing a list of global symbols and names. The input file must conform to the following rules:

- The file must reside in VMI$KWD.
- Each symbol and name must be separated by a space.
- Each symbol and name entry must be on a separate line in the input file.
- You can include comment lines and blank lines in the input file.

The following is an example of the contents of an input file:

```
fac_GBL GBLPAGES fac_NODE SCSNODE fac_VAXC VAXCLUSTER
```

Following are two examples of the command line for the GET_SYSTEM_PARAMETER callback:

```
$ VMI$CALLBACK GET_SYSTEM_PARAMETER TST_MAX WSMAX
```

In this example, the symbol TST_MAX is equated to the value of the SYSGEN parameter WSMAX.

```
$ VMI$CALLBACK GET_SYSTEM_PARAMETER INPUT.DAT "" T
```

The preceding example shows how to use the T option to specify an input file.

This callback returns VMI$_SUCCESS. If the input file is not found, it returns VMI$_FAILURE.

5.16 MESSAGE Callback

The MESSAGE callback displays a message in the standard OpenVMS format on the terminal controlling the installation.

Use this callback only to display important messages about the status of the installation. To display large blocks of tutorial text, direct output to SYS$INPUT.

Use the following command line format to invoke the MESSAGE callback:

```
VMI$CALLBACK MESSAGE severity_code id text
```

Parameters on the command line indicate the following:

severity_code
Use this parameter (P2) to specify the severity level for the message. Following are the severity level choices:

- S—severe
- I—information
- W—warning
- E—error

id
Use this parameter (P3) to specify the message identification. The message identification enables cross-referencing in the product installation guide.

text
Use these parameters (P4 through P8) to specify (in quotation marks) up to five message lines. The first line is prefixed with a percent sign (%), and the remaining lines are prefixed with a hyphen (-). The hyphen serves as a line delimiter. Use P5, P6, P7, and P8 as quoted parameters for additional message lines.

Following is an example of the command line for the MESSAGE callback:

```
$ VMI$CALLBACK MESSAGE W NOFILE "File does not exist" "Please read the manual"
```

The preceding example produces the following output for a product named PAIX010:

```
%PAIX010-W-NOFILE, File does not exist
-PAIX010-W-NOFILE, Please read the manual
```

This callback always returns VMI$_SUCCESS.

5.17 PATCH_IMAGE Callback (VAX Only)

VAX

On VAX systems, the PATCH_IMAGE callback patches existing native-mode images as part of the product installation. For each patch, the installation kit must contain a patch file that includes the commands needed to patch the specified image. Specify the image to be patched either in the image parameter or as a comment on the first line of the patch file.

The PATCH_IMAGE callback invokes the following callbacks:

- UPDATE_FILE—to update an existing patch journal file
- PROVIDE_FILE—to store a newly created patch journal file in a system directory
- PROVIDE_IMAGE—to store the patched image in a system directory

Use the following command line format to invoke the PATCH_IMAGE callback:

VMI$CALLBACK PATCH_IMAGE logical patch_file [image] [options]

Parameters on the command line indicate the following:

logical
Use this parameter (P2) to assign a process logical name to the image being patched. Use this logical name in all subsequent references to the file.

patch_file
Use this parameter (P3) to enter a partial file specification of the patch file, its name and type. The patch file must include only the commands needed to patch the image. Since the device and directory are not included in the file specification, the file *must* reside in the kit's working directory. If disk space is critical, the file can be deleted upon completion of the callback.

image
Where applicable, use this parameter (P4) to specify the image being patched. If the image is specified as a comment in the patch file, enter a null string ("") in this parameter.

options
Where applicable, use this parameter (P5) to specify options for patching and subsequent disposition of the image. Select the options from the following list of characters. If you specify multiple options, do not use spaces between them.

- I—Use this option to move a shareable image's symbol table to the system's shareable image library (SYS$LIBRARY:IMAGELIB.OLB) when the patch is completed.
- J—Use this option to create a journal file of the patch or update an existing journal file in the same directory as the image.
- K—Use this option to keep old copies of the image file (do not purge).
- O—Use this option to move the file to SYS$SPECIFIC. However, you should only use this option if absolutely necessary. If your product requires the option, be sure to use the TELL_QA callback to inform the Digital OpenVMS Software Quality Management group.
- R—Use this option to reinstall the image when the patch is completed.

The options can either be specified in P5 of the call to PATCH_IMAGE, or be included as an argument on the first line of the patch file using the following format:

! filespec options

You must separate the image name and the options with a space. When this format is used, the option list on the first line of the patch file is merged with the one specified in the callback, if applicable.

Following is an example of the command line for the PATCH_IMAGE callback:

```
$ VMI$CALLBACK PATCH_IMAGE NEWIMAGE IMAGE.FIX
```

This callback returns VMI$_SUCCESS. It returns VMI$_FAILURE if the patch file or the image is not found or if the calls to UPDATE_FILE or PROVIDE_FILE return failure status.♦

5.18 PRINT_FILE Callback

The PRINT_FILE callback queues a print job to SYS$PRINT. The print job is named after the product, and multiple copies can be specified.

Use the following command line format to invoke the PRINT_FILE callback:

VMI$CALLBACK PRINT_FILE filespec [copies]

Parameters on the command line indicate the following:

filespec
Use this parameter (P2) to enter the specification of the file to be printed. Do not include a node name in the file specification.

copies
Where applicable, use this parameter (P3) to specify the number of copies to be printed. Its default value is 1.

Following is an example of the command line for the PRINT_FILE callback:

```
$ VMI$CALLBACK PRINT_FILE VMI$KWD:CHECKTRAN.TXT 3
```

This callback returns VMI$_SUCCESS unless the callback cannot find the file.

5.19 PRODUCT Callback

When a layered product requires additional product-specific callbacks, include a command procedure in SYS$UPDATE that includes those product-specific callbacks. The PRODUCT callback provides access to these callbacks by invoking the command procedure. The callback command procedure acts as an extension of VMSINSTAL and is structured like it.

Use the following command line format to invoke the PRODUCT callback:

VMI$CALLBACK PRODUCT procedure:callback parameter ...

Parameters on the command line indicate the following:

procedure:callback
Use this parameter (P2) to specify the command procedure (default type is COM) in SYS$UPDATE that contains the product-specific callback, together with the name of the callback itself. For example, the entry might be as follows:

```
$ VMI$CALLBACK PRODUCT CHECKTRAN:FIND_CHECK parameter ...
```

parameter . . .
Use the remaining parameters on the command line to pass the appropriate values to the desired product-specific callback.

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 given in Appendix C.

5.20 PROVIDE_DCL_COMMAND Callback

The PROVIDE_DCL_COMMAND callback either adds a command to the system DCL command tables and the command tables for the current process or replaces an existing DCL command. The callback does this by creating an updated version of DCLTABLES.EXE in the installation working directory using the appropriate command language description (CLD) file, which you must include in the installation kit.

AXP

On AXP systems, PROVIDE_DCL_CALLBACK checks for the logical definition DCLTABLES. If a logical definition for DCLTABLES is present, this callback uses the table defined by this logical.♦

On both AXP and VAX systems, when the installation progresses beyond safety mode, the callback moves the updated version of DCLTABLES.EXE to the system directory.

The callback can be invoked as often as needed to add more commands. If the callback finds an existing version of DCLTABLES.EXE in the working directory, it assumes the image was created by a previous call that has already provided for transfer of the image when the installation is completed. In this way, multiple updates can be made to the DCLTABLES.EXE image during the installation without requiring redundant transfers of the image to the system directory.

Use the following command line format to invoke the PROVIDE_DCL_ COMMAND callback:

VMI$CALLBACK PROVIDE_DCL_COMMAND name_type

Parameters on the command line indicate the following:

name_type
Use this parameter (P2) to enter the name and type of the CLD file used to update the tables. Only the name and type are required because the file must reside in the installation working directory. The file can be deleted when the callback returns if space is critical.

Following is an example of the command line for the PROVIDE_DCL_COMMAND callback:

```
$ VMI$CALLBACK PROVIDE_DCL_COMMAND CHECKTRAN.CLD
```

The callback returns VMI$_SUCCESS unless the CLD file or DCLTABLES.EXE is not found, or where applicable, the PROVIDE_IMAGE callback returns failed status.

5.21 PROVIDE_DCL_HELP Callback

The PROVIDE_DCL_HELP callback adds help, appropriate for the installed product, to the DCL help library (SYS$HELP:HELPLIB.HLB). The callback does this by having the UPDATE_LIBRARY callback update the library using a help file provided with the kit.

AXP

On AXP systems, PROVIDE_DCL_HELP checks for the logical definition HELPLIB. If a logical definition for HELPLIB is present, this callback uses the table defined by this logical.♦

On AXP and VAX systems, multiple help entries can be added to the library using this callback as needed, but only top-level help entries are allowed.

Use the following command line format to invoke the PROVIDE_DCL_HELP callback:

VMI$CALLBACK PROVIDE_DCL_HELP name_type

The parameters on the command line indicate the following:

name_type
Use this parameter (P2) to specify the name and type of the file used to update the library. The file must reside in the kit's working directory and can be deleted after the callback returns.

Following is an example of the command line for the PROVIDE_DCL_HELP callback:

```
$ VMI$CALLBACK PROVIDE_DCL_HELP CHECKTRAN.HLP
```

The PROVIDE_DCL_HELP callback cannot be invoked while the help library is in use. If the library is in use, PROVIDE_DCL_HELP tries to access the library for five minutes; a message that states that the library is in use appears on the screen.

VAX

On VAX systems, if the DCL help library is still in use at the end of five minutes, PROVIDE_DCL_HELP exits with a failure status.♦

AXP

On AXP systems, if the DCL help library is still in use at the end of five minutes, PROVIDE_DCL_HELP provides new help files and informs the installer. PROVIDE_DCL_HELP then exits with a success status.♦

On both AXP and VAX systems, this callback exits with the status returned by the UPDATE_LIBRARY callback.

5.22 PROVIDE_FILE Callback

The PROVIDE_FILE callback adds a new file to the system from the product kit by accessing a source file in the installation working directory.

If the installation is in safety mode, the addition is deferred until the installation progresses beyond safety mode.

The new file takes the name and type of the source file. If the system previously included a file with the same name, the new file becomes the current version.

Note

You cannot issue the PROVIDE_FILE callback more than once per file unless you specify the C option described in this section under Options. Do not use this callback to add new product images; instead, use the PROVIDE_IMAGE callback. See Section 5.23 for more information on the PROVIDE_IMAGE callback.

Use the following command line format to invoke the PROVIDE_FILE callback:

```
VMI$CALLBACK PROVIDE_FILE logical filename_type destination [options]
```

Parameters on the command line indicate the following:

logical
Use this parameter (P2) to assign a process logical name to the new file. Use the logical name in all subsequent references to the file. If you specify the T option in P5, enter a null string ("") in P2 because the logical name assignment for each file is included in the input file.

filename_type
Use this parameter (P3) to specify the source file. Only the name and type are required because the procedure assumes that the file resides in the installation working directory. Do not specify a version number. If you specify the T option in P5, enter the name of the input file in P3.

destination
Use this parameter (P4) to specify the disk and directory name where the new file is to reside. You must specify the null string ("") for this parameter, if you also specify the T option.

options
Where applicable, use this parameter (P5) to specify options using the appropriate letters from the following list. If you list more than one option, do not use intervening spaces between the selected letters.

- C—Use this option if the file must reside in more than one location. When you specify option C, the file is copied from VMI$KWD to its destination. A copy of the file remains in VMI$KWD.
- K—Use this option to keep old copies of the file (do not purge).
- O—Use this option to add the file to SYS$SPECIFIC. You should only use this option if absolutely necessary. If your product requires the option, be sure to use the TELL_QA callback to inform the Digital OpenVMS Software Quality Management group.

- T—Use this option to specify an input file that contains a list of logical names for the source files and their respective destinations. This option offers faster file processing; however, the system does not return file-specific error messages. That is, if there is an error in one of the files listed in the input file, the system only returns the general message "File Not Found" rather than identifying the specific file that contains the error.

  The input file must conform to the following rules:

  - The file must reside in VMI$KWD.
  - Each source file entry must use the same format as the command line, as follows:

    logical filename_type destination [options]

  - Each source file entry must be on a separate line in the input file.
  - You can include comment lines and blank lines in the input file.

  The following is an example of the contents of an input file for a product named TEST:

```
! PROVIDE_FILE data file
TEST_FREDE FREDE.TXT VMI$ROOT:[SYSEXE] C
TEST_FREDC FREDC.COM VMI$ROOT:[SYSUPD] K
TEST_FREDH FREDH.DAT VMI$ROOT:[SYSHLP]
TEST_FREDM FREDM.NTS VMI$ROOT:[SYSLIB]
```

Following are two examples of the command line for the PROVIDE_FILE callback:

```
$ VMI$CALLBACK PROVIDE_FILE CHECKTRAN_ CHECKTRAN.OLD -
VMI$ROOT:[SYSLIB] CK
```

The preceding example shows how to specify more than one option in the command line.

```
$ VMI$CALLBACK PROVIDE_FILE "" INPUT.DAT "" T
```

The preceding example shows how to use the T option to specify an input file.

The callback returns VMI$_SUCCESS unless the new file is not found in the working directory or the transfer fails.

5.23 PROVIDE_IMAGE Callback

The PROVIDE_IMAGE callback adds a new image file to the system from the product kit by accessing a source image file in the installation working directory.

If the installation is in safety mode, the addition is deferred until the installation progresses beyond safety mode.

The new file takes the name and type of the source image file. If the system previously included an image file with the same name, the new file becomes the current version. If a previous version of the file was known by the Install utility, the new version will replace it.

You cannot issue the PROVIDE_IMAGE callback more than once per image unless you specify the C option described in this section under Options.

The file can be either an executable image or a shareable (nonexecutable) image. If it is a shareable image, you can specify the I option in P5 to direct a copy of the image's system table to the system's shareable image library (SYS$LIBRARY:IMAGELIB.OLB).

The E option provides for patching outstanding Engineering Change Orders (ECOs) into the image, if applicable. This function should be used very carefully and only when necessary. It is preferable to use the PATCH_IMAGE callback for this purpose or to add images that do not require patching.

Use the following command line format to invoke the PROVIDE_IMAGE callback:

```
VMI$CALLBACK PROVIDE_IMAGE logical name_type destination [options] [ECO_list]
```

Parameters on the command line indicate the following:

logical
Use this parameter (P2) to assign a process logical name to the new image file. Use the logical name in all subsequent references to the file. If you specify the T option in P5, use a null string ("") in P2 because the logical name assignment for each file is included in the input file.

name_type
Use this parameter (P3) to specify the name and type of the source image file. Only the name and type are required because the callback assumes that the file resides in the installation working directory. Do not specify a version number. If you specify the T option in P5, enter the name of the input file in P3.

destination
Use this parameter (P4) to specify the disk and directory where the new image file is to reside. You must specify the null string ("") for this parameter if you also specify the T option.

options
Where applicable, use this parameter (P5) to specify options using the appropriate letters from the following list. If you list more than one option, do not use spaces between the selected letters.

- C—Use this option if the file must reside in more than one location. When you specify option C, the file is copied from VMI$KWD to its destination. A copy of the file remains in VMI$KWD.
- E—Outstanding ECOs listed in the next parameter (P6) are to be patched dynamically into the new image file. The callback builds a temporary file containing the patch commands in the following form:

  ```
  SET ECO nnn
  UPDATE
        .
        .
        .
  SET ECO nnn
  UPDATE
  ```

  It extracts the variable values (*nnn*) from the ECO list in P6. Then the completed file is used as the input to the Patch utility, which sets the ECO numbers in the specified image. Only ECO numbers are set. You must use the PATCH callback to apply instruction or data patches.
- I—Use this option to move a shareable image's symbol table to the system's shareable image library (SYS$LIBRARY:IMAGELIB.OLB).

- K—Use this option to keep old copies of the image file (do not purge).
- O—Use this option to move the file to SYS$SPECIFIC. However, you should only use this option if absolutely necessary. If your product requires the option, be sure to use the TELL_QA callback to inform the Digital OpenVMS Software Quality Management group.
- T—Use this option to specify an input file that contains a list of logical names for the source image files and their respective destinations. This option offers faster file processing; however, the system does not return file-specific error messages. That is, if there is an error in one of the files listed in the input file, the system only returns the general message "File Not Found" rather than identifying the specific file that contains the error.

  The input file must conform to the following rules:

  – The file must reside in VMI$KWD.
  – Each source file entry must use the same format as the command line, as follows:

    logical filename_type destination [options] [ECO_list]

  – Each source file entry must be on a separate line in the input file.
  – You can include comment lines and blank lines in the input file.

  The following is an example of the contents of an input file for the product named TEST:

```
! PROVIDE_IMAGE data file
TEST_FRED FRED.EXE VMI$ROOT:[SYSEXE] CKE 1,2,3
TEST_RALPH RALPH.EXE VMI$ROOT:[SYSEXE]
TEST_JULIE JULIE.EXE VMI$ROOT:[SYSEXE] KO
```

ECO_list

Where applicable, use this parameter (P6) to enter a comma-separated list of ECOs that you want applied to the new image file. The ECO numbers are maintained in the image header and can be examined using the DCL command ANALYZE/IMAGE. (See the *OpenVMS DCL Dictionary* for more information on the ANALYZE/IMAGE command.)

The callback ignores this parameter if P5 does not specify the E option.

Following are two examples of the command line for the PROVIDE_IMAGE callback:

```
$ VMI$CALLBACK PROVIDE_IMAGE TEST_ TEST.EXE -
VMI$ROOT:[SYSMSG] KE 1,2,3
```

This example includes options K and E, and ECOs 1, 2, and 3.

```
$ VMI$CALLBACK PROVIDE_IMAGE "" INPUT.DAT "" T
```

This example shows how to use the T option to include an input file.

This callback returns VMI$_SUCCESS unless the callback cannot find the source image file or a subordinate callback fails.

5.24 RENAME_FILE Callback

The RENAME_FILE callback changes the file name and type for all versions of the designated file. You can use this callback to rename a file within a directory or to relocate a file in another directory on the same device by changing its directory specification.

You cannot use the RENAME_FILE callback to relocate a file from one device to another.

If the installation is in safety mode, the name change is deferred until the installation progresses beyond safety mode.

Use the following command line format to invoke the RENAME_FILE callback:

VMI$CALLBACK RENAME_FILE filespec new_filespec

The parameters on the command line indicate the following:

filespec
Use this parameter (P2) to enter the complete specification of the file being renamed.

new_filespec
Use this parameter (P3) to specify the new directory, name and type of the file being renamed. You may omit the directory specification if the file will remain in the same directory. Wildcards may be used.

Following is an example of the command line for the RENAME_FILE callback:

```
$ VMI$CALLBACK RENAME_FILE VMI$KWD:CHECKTRAN.OLD CHECKTRAN.NEW
```

The callback returns VMI$_SUCCESS unless it cannot find the file.

5.25 RESTORE_SAVESET Callback

Where applicable, use the RESTORE_SAVESET callback to restore secondary save sets. Secondary save sets are recommended for installations where only part of the product is being installed, or for larger products where it is more practical to package the product in multiple save sets. Using secondary save sets also ensures that the CHECK_NET_UTILIZATION callback functions properly. Selective structuring of the save sets can reduce the peak disk utilization and simplify the installation.

The following conventions govern the use of multiple save sets:

- The save sets must be restored in alphabetical order, regardless of whether all of the save sets are restored. Note that save set A is automatically restored first.
- All of the save sets for a particular product must be restored from the same physical device and directory. For example, if the primary save set is restored from MFA0, all of the remaining save sets must be restored from MFA0. Save sets can span multiple volumes, however.
- The installation command procedure KITINSTAL.COM must be part of the primary save set (A).

If the distribution media is magnetic tape, the callback will rewind the distribution media once to look for a save set. If the save set is not found following the rewinding of the tape, the callback returns VMI$_FAILURE.

Restored files are assigned owner UIC [1,4] but retain the protection assigned them when they were saved. This protection should be the same as the default value for the installation process. If applicable, the owner UIC and protection should be changed after files are restored, using the SECURE_FILE callback. If the RSP (restore save set and pause) option is specified, the procedure will pause if appropriate.

Use the following command line format to invoke the RESTORE_SAVESET callback:

VMI$CALLBACK RESTORE_SAVESET saveset [option]

Parameters on the command line indicate the following:

saveset
Use this parameter (P2) to identify the save set you want to restore. The secondary save set is identified using the file type B, the next save set using C, and so forth.

For the first 26 save sets, use the alphabetic characters A through Z as file types. If your product requires more than 26 save sets, the addition file types must begin with VMI_0027, VMI_0028, and so forth up to a maximum of VMI_9999.

option
Where applicable, use this parameter (P3) to specify options by entering the appropriate option letter. Currently, the only available option for this callback is the N option (next volume).

The N option is used to indicate that the save set begins on the next volume of the distribution volume set, and the callback responds by prompting for the next volume. Even if you do not specify this option, the callback may prompt for the next volume if it cannot locate the save set on the current volume.

Following is an example of the command line for the RESTORE_SAVESET callback:

```
$ VMI$CALLBACK RESTORE_SAVESET B N
```

The callback returns VMI$_SUCCESS if it successfully restores the save set. If the attempt is unsuccessful, the callback returns VMI$_FAILURE and displays the appropriate message from the following list:

- Save set cannot be restored.
- Null save-set name specified to RESTORE_SAVESET.
- Illegal save-set name specified to RESTORE_SAVESET.

5.26 RUN_IMAGE Callback

Use the RUN_IMAGE callback when your product installation requires running an image. If the image requires input, provide an input file in the kit working directory because interactive input is not permitted.

Where applicable, the callback defines SYS$INPUT as the input file before it executes the image.

VAX

On VAX systems, use the following command line format to invoke the RUN_IMAGE callback:

VMI$CALLBACK RUN_IMAGE image-spec [input-file] ♦

AXP

On AXP systems, use the following command line format to invoke the RUN_IMAGE callback:

VMI$CALLBACK RUN_IMAGE image-spec [input-file] [option] ♦

Parameters on the command line indicate the following:

image-spec
Use this parameter (P2) to specify the required image. If the image is not in the kit working directory, use a full file specification to identify it.

input-file
Use this parameter (P3) to specify a file to provide inputs to the image. During image execution, the callback defines this file as SYS$INPUT.

AXP

option
On AXP systems, use this parameter (P4) to specify when image execution should occur. Valid values are:

- D—Image execution is deferred if the installation is in safety mode.
- I—Image executes immediately. This is the default.

The following is an example of an OpenVMS AXP command line for the RUN_IMAGE callback:

```
$ VMI$CALLBACK RUN_IMAGE CHECKTRAN.EXE "" D♦
```

If the image or a specified input file is not found, the callback exits returning VMI$_FAILURE; otherwise, it returns VMI$_SUCCESS.

5.27 SECURE_FILE Callback

The SECURE_FILE callback is used to specify the UIC and file protection code for product files. Typically, you use this callback to restore special security values to a file that has been assigned default values by another callback.

If the installation is in safety mode, this action is deferred until the installation has progressed beyond safety mode.

When the callback looks for the specified file, it looks first in the installation working directory. It then goes to the system area to ensure the proper protection of the file regardless of the safety mode.

Use the following command line format to invoke the SECURE_FILE callback:

VMI$CALLBACK SECURE_FILE filespec [owner] [protection]

Parameters on the command line indicate the following:

filespec
Use this parameter (P2) to enter the specification of the file being assigned special security.

owner
Where applicable, use this parameter (P3) to assign the file a specific owner UIC value using the standard [group,member] format.

protection
Where applicable, use this parameter (P4) to assign the file a specific protection code. Use the standard format, but omit the parentheses.

Following is an example of the command line for the SECURE_FILE callback:

```
$ VMI$CALLBACK SECURE_FILE VMI$ROOT:CHECKTRAN.DAT [111,333] S:RED,O:RWED,G,W
```

This callback returns VMI$_SUCCESS unless it cannot find the file.

5.28 SET Callback

Use the SET callback to specify installation conditions by supplying the appropriate SET option in parameter 2. Currently, VMSINSTAL provides the following SET options:

- SET ACL
- SET ASK_CASE
- SET FILE (AXP only)
- SET IVP
- SET POSTINSTALL
- SET PRODUCT_NAME (AXP only)
- SET PURGE
- SET REBOOT
- SET SAFETY
- SET SEMANTICS (AXP only)
- SET SHUTDOWN
- SET STARTUP

The options are invoked using SET as the first parameter and the appropriate option name (for example, IVP) as the second parameter.

5.28.1 SET ACL Option

The SET ACL option invokes the DCL command SET ACL to modify the access control list (ACL) of a device, directory, or file. For information about modifying ACLs, see the SET ACL command in the *OpenVMS DCL Dictionary*.

Use the following command line format to invoke the SET ACL option:

```
VMI$CALLBACK SET ACL object_type aces object_name qualifiers
```

Parameters on the command line indicate the following:

object_type
Use this parameter (P3) to specify the object for which you are modifying an ACL. You can specify the following objects:

- DEVICE—To modify the ACL of a device.
- DIRECTORY—To modify the ACL of a directory.
- FILE—To modify the ACL of a file.

aces
Use this parameter (P4) to list the ACEs that you are adding or modifying. Enclose the list with quotation marks.

object_name
Use this parameter (P5) to specify the name of the device, directory, or file for which you are modifying the ACL. If you specify multiple file names or directory specifications, separate them with commas. You cannot specify multiple device names.

qualifiers
Use this parameter (P6) to list qualifiers. You can specify any of the qualifiers available for the DCL command, SET ACL. Enclose the qualifier list with quotation marks.

Following is an example of the command line for the SET ACL option:

```
$ VMI$CALLBACK SET ACL FILE -
_$ "(identifier=NODEA$IDENT,ACCESS=NONE)" VMI$ROOT:[SYSUPD]FRED.DAT ""
```

In this example, the SET ACL option adds an ACE to the ACL for the file FRED.DAT.

This callback returns VMI$_SUCCESS if the ACL is successfully modified. The callback returns VMI$_FAILURE if the ACL is not successfully modified.

5.28.2 SET ASK_CASE Option

The SET ASK_CASE option determines the default case (for example, upper or lower) in which input from the installer is returned to the installation procedure. By default, input is returned in uppercase.

You can override the default case for a specific ASK callback by specifying the appropriate ASK option (U, L, or M).

Use the following command line format to invoke the SET ASK_CASE option:

VMI$CALLBACK SET ASK_CASE case

case
Use this parameter (P3) to specify the default case. You can specify the following values for case:

- UPPER—When you specify UPPER, all input is returned in uppercase.
- LOWER—When you specify LOWER, all input is returned in lowercase.
- MAINTAIN—When you specify MAINTAIN, all input is returned in the same case as that entered by the installer.

The SET ASK_CASE option always returns VMI$_SUCCESS.

5.28.3 SET_FILE Option (AXP Only)

AXP

On AXP systems, the SET_FILE option applies the passed qualifiers for the specified file.

Use the following command line format to invoke the SET_FILE option:

SET_FILE file-spec qualifiers [options]

Parameters on the command line indicate the following:

file_spec
Use this parameter (P2) to specify the full file specification.

qualifiers
Use this parameter (P3) to specify qualifiers. Enclose a list of qualifiers in quotation marks.

options
Where applicable, use this parameter (P4) to specify the following option.

- T—Use this option to indicate that the file-spec parameter represents a data file that contains the input file and qualifier information. This option offers faster file processing; however, the system does not return file-specific error messages. That is, if there is an error in one of the files listed in the input file, the system only returns the general message "File Not Found" rather than identifying the specific file that contains the error.

The input file must conform to the following rules:

- The file must reside in VMI$KWD.
- Each source file entry must use the same format as the command line, as follows:

  logical filename_type destination [options]

- Each source file entry must be on a separate line in the input file.
- You can include comment lines and blank lines in the input file.

The following is an example of the contents of an input file for a product named TEST:

```
! PROVIDE_FILE data file
TEST_FREDE FREDE.TXT VMI$ROOT:[SYSEXE] C
TEST_FREDC FREDC.COM VMI$ROOT:[SYSUPD] K
TEST_FREDH FREDH.DAT VMI$ROOT:[SYSHLP]
TEST_FREDM FREDM.NTS VMI$ROOT:[SYSLIB]
```

Following are examples of command lines that invoke the SET_FILE option:

```
$ VMI$CALLBACK SET_FILE VMI$ROOT:[SYSUPD]DUMMY.DAT "/VERSION=2/ERASE"
$ VMI$CALLBACK SET_FILE FILE-LIST.DAT "" T
```

This SET_FILE option returns VMI$_FAILURE if the file is not found; otherwise, it returns VMI$_SUCCESS. ♦

5.28.4 SET IVP Option

The SET IVP option specifies whether an Installation Verification Procedure (IVP) is to be used with the product installation, assuming that the product includes an IVP. If you do not enter a SET IVP command, VMSINSTAL specifies that the IVP does not run.

Use the following command line format to invoke the SET IVP option:

VMI$CALLBACK SET IVP keyword [help]

The parameters on the command line indicate the following:

keyword
Use this parameter (P3) to make the appropriate selection from the following list:

- ASK—Use this keyword to prompt the installer for a decision on whether to run the IVP. If help is specified in P4, the prompt is prefaced by a help message.

  The IVP will run if the installer responds by entering a *YES* or presses the Return key.
- YES—Use this keyword to specify that the IVP will run at the completion of the installation.
- NO—Use this keyword to specify that the IVP will not run at the completion of the installation.

help
Enter the letter *H* in this parameter (P4) if you want to precede the ASK prompt with the following help message:

```
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.
```

The SET IVP option always returns VMI$_SUCCESS.

5.28.5 SET POSTINSTALL Option

Use the SET POSTINSTALL option when you want a product's KITINSTAL.COM procedure to be called after all files have been moved to their target directories. This callback is useful when you must have all new files in place before completing an installation. By default, there is no postinstall phase.

This callback passes the string VMI$_POSTINSTALL to the KITINSTAL.COM procedure in P1. The KITINSTAL.COM procedure is executed immediately before the IVP phase.

Use the following command line format to invoke the SET POSTINSTALL option:

```
VMI$CALLBACK SET POSTINSTALL keyword
```

keyword
Use this parameter (P3) to make the appropriate selection from the following list:

- YES—Use this keyword to indicate the postinstall phase will be called when all files have been moved.
- NO—Use this keyword to disable the postinstall phase.

The SET POSTINSTALL option always returns VMI$_SUCCESS.

5.28.6 SET PRODUCT_NAME Option (AXP Only)

AXP

On AXP systems, the SET PRODUCT_NAME option defines the global symbol VMI$PRODUCT_NAME as the value of the passed parameter. Use this value in the *name* column of the history file.

Use the following command line format to invoke the SET PRODUCT_NAME option:

```
SET PRODUCT_NAME string
```

Parameters on the command line indicate the following:

string
Use this parameter (P3) to specify a string that represents the value you want to assign to VMI$PRODUCT_NAME. This string can be one word or several words. If the string is more than one word, or if the string is case-sensitive, this parameter must be passed to VMI$PRODUCT_NAME as a quoted string.

Following are examples of command lines for the SET PRODUCT_NAME option:

```
$ VMI$CALLBACK SET PRODUCT_NAME RDB
$ VMI$CALLBACK SET PRODUCT_NAME "OpenVMS AXP Rdb"
```

This option always returns VMI$_SUCCESS.♦

5.28.7 SET PURGE Option

The SET PURGE option specifies whether files replaced during the installation are to be purged. Files accessed by callbacks that specify the KEEP (K) option are not purged by the installation procedure, regardless of whether the SET PURGE option was selected. If you do not use the SET PURGE option, VMSINSTAL purges replacement files.

Use the following command line format to invoke the SET PURGE option:

VMI$CALLBACK SET PURGE keyword [help]

Parameters on the command line indicate the following:

keyword
Use this parameter (P3) to make the appropriate selection from the following list:

- ASK—Use this keyword to prompt the installer for a decision on whether replaced files are to be purged. If help is specified in P4, the prompt is prefaced by a help message.

  Replaced files will be purged if the installer responds with *YES* or presses the Return key.
- YES—Use this keyword to specify that the replaced files will be purged.
- NO—Use this keyword to specify that the replaced files will not be purged.

help
Enter the letter *H* in this parameter (P4) if you want to precede the ASK prompt with the following help message:

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

The SET PURGE option always returns VMI$_SUCCESS.

Note

The SET PURGE option must precede all PROVIDE callbacks.

5.28.8 SET REBOOT Option

The SET REBOOT option, which was supported in previous releases of VMSINSTAL, is obsolete. The SET SHUTDOWN option should be used in its place (see Section 5.28.11).

AXP

On AXP systems, VMSINSTAL issues a message stating that SET REBOOT is obsolete and automatically calls SET SHUTDOWN.♦

5.28.9 SET SAFETY Option

The SET SAFETY option establishes the safety level of the installation; that is, the installation's ability to recover from a system failure. Because implementation of the safety feature requires a higher peak utilization of disk space, parameters should be selected only after weighing the need for installation safety against anticipated space availability.

The SET SAFETY option also records the level of safety in the marker file to establish appropriate crash recovery procedures.

If you omit this option, VMSINSTAL automatically specifies safety mode unconditionally; that is, without regard to available disk space. If the system does not have sufficient disk space to support it, the installation terminates with failure status.

Use the following command line format to invoke the SET SAFETY option:

```
VMI$CALLBACK SET SAFETY keyword [peak]
```

Parameters on the command line indicate the following:

keyword
Use this parameter (P3) to specify the required safety level for the installation from the following list. Note that an entry is required in P4 if you select CONDITIONAL.

- YES—Specifies safety mode. You may specify a required peak disk space utilization as the fourth parameter in the command line. If there is insufficient disk space, the option returns VMI$_FAILURE and the installation terminates.

  If you enter YES, note that the safety sensitive callbacks will defer action that would otherwise jeopardize the safety level of the installation. For example, the PROVIDE_FILE callback defers movement of the new file until the installation progresses beyond safety mode.

  Deferred action on files by the safety-sensitive callbacks occurs only if safety is set to on; otherwise, the action occurs immediately. For small-disk systems, you may want to invoke these callbacks only after the affected file is no longer needed for the installation.

- CONDITIONAL—Specifies safety mode only if there is enough space to support it. You must specify required peak disk space utilization as the fourth parameter in the command line. This is the suggested method for setting the safety level.

- NO— Indicates that you do not want to specify safety mode.

peak
Use this parameter (P4) to enter the peak disk utilization level in blocks required for your installation. If you omit this parameter and have entered either YES or CONDITIONAL in P3, the option responds as though there is sufficient disk space for safety operations.

The SET SAFETY option returns VMI$_SUCCESS unless you specify safety mode without sufficient disk space to support it, in which case VMI$_FAILURE is returned.

5.28.10 SET_SEMANTICS Option (AXP Only)

AXP

On AXP systems, the SET_SEMANTICS option sets a semantic (file characteristic) for the specified file. You can specify only one semantic.

Use the following command line format to invoke the SET_SEMANTICS option:

SET_SEMANTICS file-spec semantic

Parameters on the command line indicate the following:

file_spec
Use this parameter (P2) to specify the full file specification.

semantic
Use this parameter (P3) to specify the semantic to be set for the specified file.

Following is an example of a command line that invokes the SET_SEMANTICS option:

```
$ vmi$callback SET_SEMANTICS VMI$ROOT:[SYSUPD]DUMMY.DDIF DDIF
```

This SET_SEMANTICS option returns VMI$_FAILURE if the file is not found; otherwise, it returns VMI$_SUCCESS.♦

5.28.11 SET SHUTDOWN Option

The SET SHUTDOWN option specifies that a reboot is necessary to complete the product installation. This option allows you to indicate:

- if a system reboot is allowed at this time
- how long to wait before system is shutdown
- whether to do an automatic reboot

Use this option in the question portion of the product installation procedure. If you are updating or upgrading in an alternate root directory, or if you omit a SET SHUTDOWN command from your installation procedure, the system will not be shut down following the installation.

Note

This callback should replace the SET REBOOT option.

Use the following command line format to invoke the SET SHUTDOWN option:

VMI$CALLBACK SET SHUTDOWN keyword symbol [installation-action]

Parameters on the command line indicate the following:

keyword
Use this parameter (P3) to make the appropriate selection:

- ASK—Use this keyword to specify a system shutdown is required and to prompt the installer for shutdown information.
- NO—Use this keyword to specify a system shutdown is not required or cancelled.

symbol
Use this parameter (P4) to specify a global symbol that indicates whether the installer is allowing the system to shut down. VMSINSTAL defines this symbol as a Boolean true or false.

installation-action
Use this parameter (P5) to specify the action the installation procedure will take if the shutdown is not allowed. This parameter is only required with the keyword option ASK.

- EXIT—Use this keyword to indicate that the product installation procedure will exit immediately thereby not placing the product files on the system.
- CONTINUE—Use this keyword to indicate that the product installation procedure will continue installing the product on the system but requires the system to be rebooted before the product is completely installed and available for use.

The SET SHUTDOWN option always returns VMI$_SUCCESS.

5.28.12 SET STARTUP Option

The SET STARTUP option specifies a product-specific startup command procedure. VMSINSTAL executes the product-specific startup command procedure immediately following the installation and just prior to running the IVP, where applicable.

Before invoking this option, use the PROVIDE_FILE callback to store the product-specific startup command procedure in the SYS$STARTUP directory.

Use the following command line format to invoke the SET STARTUP option:

VMI$CALLBACK SET STARTUP filename [parameters]

The parameters on the command line indicates the following:

filename
Use this parameter (P3) to specify the file name and type of the product startup command procedure.

parameters
Use this parameter (P4) to pass parameters to the startup procedure. Separate parameters with a space and enclose the list in quotation marks. For example, to pass VMSINSTAL as P1 and DOALL as P2, type the following command line:

```
$ VMI$CALLBACK SET STARTUP CHECKTRAN_STARTUP.COM "VMSINSTAL DOALL"
```

The SET STARTUP option always returns VMI$_SUCCESS.

5.29 SUMSLP_TEXT Callback

The SUMSLP_TEXT callback is used to edit text files with the SUMSLP batch mode line editor and an appropriate editing file. (See the *OpenVMS SUMSLP Utility Manual* for more information.) In addition to the editing function, the SUMSLP_TEXT callback supports the use of the checksum function to determinine whether the file has been incompatibly altered.

After you build the editing input file, the editing input file must reside in the installation kit working directory. In addition to editing standard data files, you can use the callback to edit modules of the following ASCII-based library types: help, macro, and text.

With library editing, the callback extracts the appropriate library module, edits it, and then returns it to the library using the UPDATE_LIBRARY callback. For nonlibrary files, the callback returns the file to the source directory using the PROVIDE_FILE callback.

To utilize the checksum functions, you must provide two checksums: one representing the file prior to editing and one representing the file after editing. To determine the appropriate values, run the CHECKSUM program on the current version of the file, and then run it on the new version of the file. In each case, the program stores the checksum value in the DCL symbol CHECKSUM$CHECKSUM. Use the value from the current version of the file as P6 in the SUMSLP_TEXT callback command line. Use the value from the new version of the file as P7.

The callback begins by looking for the editing file in the installation working directory. After finding the editing file, the callback checks to see if the target (file or module to be edited) is specified in the command line. If it is not, the callback builds the target specification together with the rest of the command line from the first line of the editing file.

Next, the callback determines whether the target is a data file or a library module.

If it is a data file, the callback parses P4 to make the file specification format compatible with the PROVIDE_FILE command structure. Then the callback runs a checksum on the file, edits it and, if the edited file was found in a system directory, returns it with a PROVIDE_FILE callback.

If the target is a library module, the callback looks for the specified library, extracts the specified module into a file in the installation work directory, and then runs the CHECKSUM program on the file.

If the checksum agrees with the value in P6, the callback invokes the SUMSLP editor to edit the file. It then uses the UPDATE_LIBRARY callback to put the new version of the file in the library.

Use the following command format line to invoke the SUMSLP_TEXT callback:

```
VMI$CALLBACK SUMSLP_TEXT logical edit-file target type
current-checksum [new-checksum] [option]
```

Parameters on the command line indicate the following:

logical
Use this parameter (P2) to assign a logical name to the file or library module being edited. Use the logical name in all subsequent references to the file or library module.

edit-file
Use this parameter (P3) to specify the name and type of the editing file. Do not specify the device or directory because the callback assumes that the file is located in the installation working directory. To save disk space, you can delete this file after the callback returns.

target
If the callback is editing a file, use this parameter (P4) to specify the file being edited.

If the callback is editing a library module, use this parameter to identify the target by entering the full library specification, followed by a comma and then the name of the library module. For example, if the callback is editing the CONVERT module in the help library, enter the following line:

```
$ VMI$ROOT:[SYSHLP]HELPLIB.HLB,CONVERT
```

type
If the callback is editing a file, use this parameter (P5) to enter the keyword FILE.

If the callback is editing a library module, enter one of the following to describe the type of library module being edited:

- HELP
- TEXT
- MACRO

current-checksum
Use this parameter (P6) to specify the checksum for the current version of the target. If the target checksum does not agree with this value, the callback checks to see if the target was previously edited; that is, it matches new-checksum. If the target was not previously edited, the callback assumes the target has been tampered with, displays an appropriate message, and returns VMI$_FAILURE.

new-checksum
Where applicable, use this parameter (P7) to specify the checksum for the new version of the target. If the target checksum agrees with this value, the callback assumes the target has already been edited and returns VMI$_SUCCESS.

option
Where applicable, use this parameter (P8) to specify options by entering the appropriate option letter. Currently, only the K option is available.

- K—Use this option to retain the current version of the edited file.

Instead of using the callback command line, you can specify the target, checksum data, and parameters P4 through P8 on the first line of the edit file. If you do this, use the following command line format:

-;! target type current-checksum new-checksum options

Following is an example of the command line for the SUMSLP_TEXT callback:

```
$ VMI$CALLBACK SUMSLP_TEXT NEW$FILE TEXT.FIX -
_$ VMI$KWD:CHECKTRAN.DOC FILE 1627371981 1563932641 K
```

The callback returns VMI$_SUCCESS if it succeeds in editing the target or if it finds that editing was done by a previous callback. The callback exits with VMI$_FAILURE if either the editing file or the target cannot be found, if it detects file tampering, or if the attempt to move the edited target back to its source fails.

5.30 TELL_QA Callback

The TELL_QA callback allows you to identify peculiarities in your product that should be noted if special technical considerations exist. The TELL_QA callback generates specific messages for test environments only.

These messages appear only if the installation is being done in quality assurance (QA) mode. Therefore, to use this callback, you must specify the QA mode option (Q) in the command line when you invoke VMSINSTAL.

Use the following command line format to invoke the TELL_QA callback:

VMI$CALLBACK TELL_QA message

message
Use this parameter (P2) to enter the message in the form of a quoted string. In some situations, it may be advisable to preface the installation software with appropriate TELL_QA callbacks.

The TELL_QA callback always returns VMI$_SUCCESS.

5.31 UNWIND Callback

The UNWIND callback provides a quick way to exit the installation. When UNWIND is invoked, VMSINSTAL runs through its cleanup code, signals that the installation has failed, and executes a STOP command to return control to the user.

Use the UNWIND callback to exit the installation after errors or Ctrl/Y interrupts, where the installation is several levels deep into DCL; you do not have to return status back up a list of invoked command procedures.

Use the following command line format to invoke the UNWIND callback:

VMI$CALLBACK UNWIND

5.32 UPDATE_ACCOUNT Callback

The UPDATE_ACCOUNT callback uses the Authorize utility to update accounts in SYSUAF and NETPROXY. It should be used only on accounts created through the CREATE_ACCOUNT callback.

Use the following command line format to invoke the UPDATE_ACCOUNT callback:

VMI$CALLBACK UPDATE_ACCOUNT username qualifiers

Parameters on the command line indicate the following:

username
Use this parameter (P2) to identify the account being updated.

qualifiers
Use this parameter (P3) to list appropriate qualifiers; that is, qualifiers related to the AUTHORIZE command MODIFY, such as /DEVICE and /DIRECTORY. (See the *OpenVMS System Management Utilities Reference Manual* for more information about the MODIFY command qualifiers.) Enclose the qualifier string in quotation marks.

Following is an example of the command line for the UPDATE_ACCOUNT callback:

```
$ VMI$CALLBACK UPDATE_ACCOUNT SMITH "/PASSWORD=FCDAUX"
```

The UPDATE_ACCOUNT callback returns VMI$_SUCCESS unless it does not find the SYSUAF file.

5.33 UPDATE_FILE Callback

The UPDATE_FILE callback makes a system file available for subsequent updating by copying it to the installation working directory if a copy is not currently there.

Note

Do not use this callback to update library files (see the UPDATE_ LIBRARY callback). If a new version of an existing library file is needed, create the new version in the working directory and then use the PROVIDE_FILE callback to replace the current version.

If the installation is in safety mode and a copy of the file is needed in the installation working directory, the callback copies the file to the working directory using the Backup utility. The file retains the original file ownership.

If the installation is not in safety mode, or if the file is found in the installation working directory, the callback exits without taking any action other than assigning P2 to its file specification.

When the update is completed, the file is moved back to the system directory. Note, however, that the move can be deferred if the installation is in safety mode.

Use the following command line format to invoke the UPDATE_FILE callback:

VMI$CALLBACK UPDATE_FILE logical filespec [option]

Parameters on the command line indicate the following:

logical
Use this parameter (P2) to assign a logical name to the file being updated. Use the logical name in all subsequent references to the file.

filespec
Use this parameter (P3) to enter the complete specification of the file being updated.

option
Where applicable, use this parameter (P4) to specify options by entering the appropriate option letter. Currently, only the K option is available.

Use the K option to move the updated file to a location other than that specified in P3. Normally, when the update is completed, the file is moved back to the system directory. When you specify the K option, the file is kept in the kit working directory (VMI$KWD) after the update is completed.

Following is an example of the command line for the UPDATE_FILE callback:

```
$ VMI$CALLBACK UPDATE_FILE FORMAT VMI$ROOT:[SYSEXE]FORMAT.EXE
```

The UPDATE_FILE callback returns VMI$_SUCCESS unless the file is not found or any internal callbacks are unsuccessful.

5.34 UPDATE_IDENTIFIER Callback

The UPDATE_IDENTIFIER callback modifies an identifier in the rights database.

Use the following command line format to invoke the UPDATE_IDENTIFIER callback:

VMI$CALLBACK UPDATE_IDENTIFIER id-name qualifiers

Parameters on the command line indicate the following:

id-name
Use this parameter (P2) to specify the name of the identifier to be modified.

qualifiers
Use this parameter (P3) to specify qualifiers for the UPDATE_IDENTIFIER callback. You can specify the same qualifiers that are available for the MODIFY /IDENTIFIER command of the Authorize utility. For more information about the MODIFY/IDENTIFIER command, see the *OpenVMS System Management Utilities Reference Manual*.

Following is an example of a command line that invokes the UPDATE_ IDENTIFIER callback:

```
$ VMI$CALLBACK UPDATE_IDENTIFIER ACCOUNTING "/VALUE=UIC:[300,21]"
```

In this example, the UPDATE_IDENTIFIER callback specifies a new value for the ACCOUNTING identifier.

This callback returns VMI$_SUCCESS if the identifier is successfully modified. The callback returns VMI$_FAILURE if the identifier is not successfully modified.

5.35 UPDATE_LIBRARY Callback

The UPDATE_LIBRARY callback is used to immediately update an existing library. It can be used with either the OpenVMS Librarian or the RSX LBR utility. The callback records the update in the marker file immediately to aid in crash recovery procedures if needed.

Use the following command line format to invoke the UPDATE_LIBRARY callback:

VMI$CALLBACK UPDATE_LIBRARY logical filespec type qualifiers source_file

Parameters on the command line indicate the following:

logical
Use this parameter (P2) to assign a process logical name to the library file being updated. Use the logical name in all subsequent references to the file.

filespec
Use this parameter (P3) to enter the complete specification of the library file being updated.

type
Use this parameter (P4) to enter the LIBRARY command qualifier that identifies the type of library being updated. If an RSX library is being updated, enter "RSX." If the library type is RSX, the target system must have VAX–11 RSX installed.

If a OpenVMS library is being updated, enter one of the following:

- HELP
- MACRO
- OBJECT
- SHARE
- TEXT

You must make an entry in this parameter.

qualifiers
Use this parameter (P5) to list qualifiers for the LIBRARY command. Enclose the list in quotation marks. (See the *OpenVMS Command Definition, Librarian, and Message Utilities Manual* for more information on the Librarian utility.)

source_file
Where applicable, use this parameter (P6) to enter the full specification for the source file being used to update the library. You can use wildcards. If you need to save disk space, you can delete this file when the callback finishes.

Following is an example of the command line for the UPDATE_LIBRARY callback:

```
$ VMI$CALLBACK UPDATE_LIBRARY VAXFMS$STARLET VMI$ROOT:[SYSLIB]STARLET.OLB -
        OBJECT "/DELETE=(FDV,FDVMSG,FDVDAT,FDVERR,FDVTIO,FDVXFR,HLL,HLLDFN)"
```

The UPDATE_LIBRARY callback returns VMI$_SUCCESS. If the file is not found, it returns VMI$_FAILURE.

A

Symbols and Logical Names

This appendix lists and briefly describes the symbols and logical names used by VMSINSTAL.

VMI$	VMSINSTAL assigns this variable global symbol to user responses and to command line parameters that will have only short-term use.
VMI$ALTERNATE_ROOT	When active, this local symbol indicates that the product is being installed in a root on a disk other than the running system disk.
VMI$ARCHITECTURE	VMSINSTAL uses this global symbol to indicate the architecture (Alpha AXP or VAX) of the target disk.
VMI$AUTO_FILE	VMSINSTAL uses this logical name to access the auto-answer file.
VMI$AUTO_OPTION	VMSINSTAL stores the value of the auto-answer mode (read or write) in this local symbol.
VMI$BACKUP_OPENIN	This global symbol is used to store the value of the target system's BACKUP-E-OPENIN error message.
VMI$BOOTING	This local Boolean symbol is true only if the startup procedure calls VMSINSTAL to recover from a crash during the installation.
VMI$CALLBACK	VMSINSTAL equates this local symbol to *@SYS$UPDATE:VMSINSTAL*. The KITINSTAL command procedure uses it to call various VMSINSTAL subroutines (also referred to as callbacks).
VMI$CALL_FILE	VMSINSTAL uses this logical name to access the callback trace file.
VMI$CLD	When a new command is to be added to the system, VMSINSTAL assigns this process logical name to the command table file that is to be modified.
VMI$COM	This is the process logical name assigned to the patch file when it is found by the PATCH_IMAGE callback.
VMI$COMMAND_P1	VMSINSTAL uses this local symbol to determine when all products have been installed.
VMI$COMMON_ROOT	VMSINSTAL uses this Boolean symbol to determine the root directory for the installation.
VMI$CONSOLE	This Boolean symbol is set true if the installation is being done from a console device, and is used to initiate console-related functions.
VMI$DEBUG	When this local Boolean symbol is set true, the installer has included the kit debug option in the VMSINSTAL command line.
VMI$DEFER_FILE	VMSINSTAL uses this logical name to access the defer file.
VMI$DEMON	VMSINSTAL uses this logical name when the statistics report subprocess is running.

VMI$DEVICE	This local symbol is equated to the name of the device that contains the distribution volume during the installation.
VMI$_FAILURE	This local symbol, equated to %X10F50000, indicates that the installation was unsuccessful.
VMI$FIL	This local symbol stores the results of file searches done by the UPDATE_FILE subroutine.
VMI$FIND	VMSINSTAL equates this local symbol to *@SYS$UPDATE:VMSINSTAL FIND_FILE*. KITINSTAL uses it to call the frequently invoked *FIND_FILE* routine.
VMI$FIRST_NEXT	VMSINSTAL uses this local symbol as a flag for prompting the user to enter the product from the first or next distribution volume.
VMI$FREE_BLOCKS	VMSINSTAL assigns this local symbol to the number of free blocks available for the installation.
VMI$INPUT	This local symbol is defined to be the file name for the input file to the callback. See the VMI$TABLE symbol.
VMI$INS	This local symbol is used by the DELETE_FILE callback to determine if the file was installed.
VMI$INSTALLATION_STATUS	This local symbol is used to indicate the status of the installation.
VMI$INSTALLING	When the installation begins, VMSINSTAL sets this local Boolean symbol to logical true.
VMI$IVP	When this global Boolean symbol is set true, the Installation Verification Process (IVP) is in progress.
VMI$JNL	VMSINSTAL assigns this process logical name to the journal file used in image patching, where applicable.
VMI$KWD	This is the process logical name VMSINSTAL assigns to the kit working directory. VMI$KWD is the subdirectory of [SYSUPD] used to store the installation files.
VMI$KWD_DELETE	This global symbol specifies a boolean value indicating whether or not the working directory will be deleted at the end of the installation.
VMI$LIB	This process logical name is assigned to library modules being updated by the SUMSLP_TXT subroutine.
VMI$LIST	VMSINSTAL uses this local symbol to store the string value of the command line parameter that lists the products to be installed.
VMI$LIST_COUNT	VMSINSTAL uses this local symbol as a counter to determine the number of products in the distribution kit.
VMI$LIST_DONE	VMSINSTAL compares the value of this counter with VMI$LIST_COUNT to determine when the product kit is installed.
VMI$LIST_FILE	This local symbol is used to open and read the file defined by the VMI$LIST symbol.
VMI$MARKER_FILE	VMSINSTAL uses this logical name to access the marker file.
VMI$MSG	This local symbol is equated to *@SYS$UPDATE:VMSINSTAL MESSAGE VMSINSTAL*. VMSINSTAL uses this symbol to call the MESSAGE callback when an error message is required.
VMI$NEW	When an image is moved from the installation directory to a system directory, VMSINSTAL assigns this process logical name to the specified image.

VMI$NO_ERROR	This local symbol is equated to *DEFINE/USER SYS$ERROR NL:* and is used by VMSINSTAL to turn off error messages until the next DCL image executes.
VMI$NO_OUTPUT	This local symbol is equated to *DEFINE/USER SYS$OUTPUT NL:* and is used by VMSINSTAL to turn off the output until the next DCL image executes.
VMI$NO_SUCH_SAVESET	VMSINSTAL sets this Boolean flag to true when it cannot find the first save set for a product.
VMI$OLD	When a file is copied from the installation working directory to a system directory, VMSINSTAL assigns this process logical name to the specified file.
VMI$PID	This local symbol contains the process ID of the current process.
VMI$PLACE	VMSINSTAL uses this global symbol to indicate the location of the distribution volume.
VMI$PRETTY	VMSINSTAL equates this local symbol to the name of the product being installed. VMI$PRETTY is structured as formatted ASCII output and is used primarily for outputting product-related messages during the installation.
VMI$PRODUCT	This local symbol is also equated to the name of the product being installed; however, it is implemented for internal logic functions only.
VMI$PRT	When a file is to be printed, VMSINSTAL assigns this process logical name to the specified file before issuing the PRINT command.
VMI$PURGE	When this global Boolean symbol is set true, files that have been replaced during the installation should be deleted.
VMI$QA	This Boolean symbol is set true if the Q option has been selected in the command that calls VMSINSTAL.
VMI$QA_FAIL	This global Boolean symbol is set true if the installation fails.
VMI$QUE	This symbol is defined to be the queue to which release notes are printed. See the RELEASE_NOTES callback for more information.
VMI$REBOOT	When this global Boolean symbol is true, VMSINSTAL will automatically reboot the system upon completion of the installation.
VMI$REL	This local symbol is defined to be the name of the release notes file.
VMI$REMOTE	This local Boolean symbol is set true when the distribution volume is located on a remote node.
VMI$REMOUNT	This local Boolean symbol is set true when the console distribution volume is mounted on a console device or mounted /FOREIGN on a disk device. Upon completion of the installation, the console distribution volume must be remounted.
VMI$REN	This process logical name is assigned to a subroutine used by KITINSTAL to clean up files using minimal overhead.
VMI$ROOT	This logical name is assigned the value of the root directory being used for the installation. The value is SYS$COMMON if you do not select the alternate root option.
VMI$SAFETY	This global Boolean symbol is set true if the installation is being done in safety mode. The decision to install the product in safety mode is in part determined by the amount of disk space available. Your KITINSTAL procedure must make this determination.

VMI$SAVED_DIR	This local symbol is used to store the default device and directory of the current process during the installation.
VMI$SAVED_MSG	VMSINSTAL uses this local symbol to store the message format of the current process during the installation.
VMI$SAVED_PRIVS	This local symbol is used to store the privileges of the current process during the installation.
VMI$SAVED_PROT	VMSINSTAL uses this local symbol to store the default protection code of the current process during the installation.
VMI$SAVED_UIC	This local symbol is used to store the UIC of the current process during the installation.
VMI$SET_MSG_OFF	VMSINSTAL equates this local symbol to the following: SET MESSAGE/NOFACILITY/NOSEVERITY/NOIDENT/NOTEXT.
VMI$SET_MSG_ON	VMSINSTAL equates this local symbol to the following: SET MESSAGE/FACILITY/SEVERITY/IDENT/TEXT.
VMI$SPECIFIC	VMSINSTAL uses this logical name to reference the target system's system-specific top-level directory.
VMI$STARTUP	This global symbol is equated to the name of the product startup command procedure, typically part of SYSTARTUP.COM.
VMI$STAT_FILE	VMSINSTAL uses this logical name to access the statistics file.
VMI$_SUCCESS	This local symbol, equated to %X10F50001, indicates a successful installation.
VMI$SYS_ARCHITECTURE	VMSINSTAL uses this global symbol to indicate the architecture (Alpha AXP or VAX) of the system that is running VMSINSTAL.
VMI$TABLE	This local symbol determines whether or not the input to a callback is in a file. This symbol is used for the following callbacks: GET_SYSTEM_PARAMETER, PROVIDE_FILE, and PROVIDE_IMAGE.
VMI$TEMP_FILE	VMSINSTAL uses this logical name to access the temporary file, which in turn reads the patch file.
VMI$TERMINAL_FILE	VMSINSTAL assigns this process logical name to the file used for terminal operations.
VMI$TXT	This process logical name is assigned to the target file in text correction operations.
VMI$_UNSUPPORTED	This local symbol, equated to %X10F50000, indicates that the installation was unsuccessful because it attempted to execute an unsupported feature.
VMI$UNWIND	This symbol is defined as true if the UNWIND callback is invoked.
VMI$UPDATE_MARKER	This local symbol is used to transfer marker data during the installation.
VMI$VERSION	This local symbol stores the version number of the operating system.
VMI$VMS_VERSION	This local symbol contains the value of the operating system version level. It is used to determine whether or not the product being installed is compatible with the operating system.

B

Sample OpenVMS Installation Procedure

Example B–1 shows a sample OpenVMS installation procedure. The product used in the sample is DECRam_for_OpenVMS.

Note

Until the VMI$ARCHITECTURE logical is implemented on OpenVMS systems, you must use one of the following mechanisms to determine the architecture of the system running a KITINSTAL.COM that is common for OpenVMS VAX and OpenVMS AXP systems.

For an OpenVMS VAX or OpenVMS AXP product that requires OpenVMS Version 5.5 or later, use the following code sequence to determine the system architecture:

```
$  SYM = F$GETSYI("ARCH_TYPE")
$  IF SYM .EQ. 1 THEN DO_VAX_TASKS
$  IF SYM .EQ. 2 THEN DO_AXP_TASKS
```

This F$GETSYI("ARCH_TYPE")lexical call returns 1 for OpenVMS VAX platforms and 2 for OpenVMS AXP platforms.

For an OpenVMS VAX or OpenVMS AXP product that requires a version of OpenVMS prior to OpenVMS Version 5.5, use the following sequence:

```
$ SYM = F$GETSYI("HW_MODEL")
$ IF SYM .LE. 1023 THEN DO_VAX_TASKS
$ ELSE DO_AXP_TASKS
```

The F$GETSYI("HW_MODEL") lexical returns a value greater than 1023 for processors designed under the OpenVMS Alpha AXP architecture.

Example B–1 Sample DIGITAL OpenVMS KITINSTAL.COM

```
$ ! .TITLE KITINSTAL - DECRam_for_OpenVMS Installation Procedure
$ ! .IDENT /V2.1/
$ !
$ ! COPYRIGHT (C) 1985, 1987, 1988, 1989, 1990, 1991, 1992 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 OF 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 WHICH IS NOT SUPPLIED BY DIGITAL.
$ !
$ !
$ !++
$ !
$ ! FACILITY:  VMSINSTAL
$ !
$ ! ABSTRACT:
$ !
$ ! This procedure installs the DECRam_for_OpenVMS Product using VMSINSTAL
$ !
$ ! ENVIRONMENT: DCL
$ !
$ !     Setup error handling
$ !
$ ON CONTROL_Y THEN GOTO CONTROL_Y
$ !
$ !     Handle INSTALL, IVP and unsupported parameters passed by VMSINSTAL
$ !
$ DECRAM$VERSION :== "T1.1"
$ IF P1 .EQS. "VMI$_INSTALL" THEN GOTO INSTALL
$ IF P1 .EQS. "VMI$_IVP" then goto DECRAM_RUN_IVP
$ EXIT VMI$_UNSUPPORTED
$ !
$ INSTALL:
$ !
$ DECRAM_TMP = F$TYPE(VMI$ARCHITECTURE)
$ DECRAM_ARCH = 0 ! Initialize to VAX
$ IF DECRAM_TMP .EQS. "" THEN DECRAM_ARCH = 0
$ IF DECRAM_TMP .EQS. "INTEGER" THEN DECRAM_ARCH = 0
$ IF DECRAM_TMP .NES. "STRING" THEN GOTO DONE_ARCH
$ IF VMI$ARCHITECTURE .EQS. "AXP" THEN DECRAM_ARCH = 1
$ IF VMI$ARCHITECTURE .EQS. "VAX" THEN DECRAM_ARCH = 0
$ !
$ DONE_ARCH:
$! At this point DECRAM_ARCH is = 0 if VAX or = 1 if AXP
$!
$ WRITE SYS$OUTPUT ""
$ WRITE SYS$OUTPUT "$!  Copyright (c) 1992 Digital Equipment Corporation.  All rights reserved."
$ WRITE SYS$OUTPUT ""
$ !
$ ! Before starting anything OpenVMS version
$ !
$ !*****************************************************************
$ !* Must be OpenVMS V5.2 or later...
$ !*****************************************************************
$ !
```

(continued on next page)

Example B–1 (Cont.) Sample DIGITAL OpenVMS KITINSTAL.COM

```
$ IF DECRAM_ARCH .EQ. 1 THEN GOTO DECRAM_VERSION_DONE
$ !
$ CHECK_VMS_VERSION:
$ ON ERROR THEN GOTO DECRAM_VERSION_ERROR
$ ! Version check may blow up on pre-V4 system
$ !
$ !     Check OpenVMS version for supported release.
$ !
$ !     Define earliest released version of OpenVMS supported by DECRAM
$ !
$   DECRAM$REL_VERSION = "052"
$ !
$ !     Define earliest field test version of OpenVMS supported by DECRAM
$ !
$   DECRAM$FT_VERSION = "052"
$ !
$ !     Extract current running version number of OpenVMS
$ !
$   DECRAM$VMS_VERSION = "''F$ELEMENT(0,",",VMI$VMS_VERSION)'"
$   DECRAM$VMI_VERSION == F$ELEMENT(1,",",VMI$VMS_VERSION)
$ !
$ DECRAM_RELEASED:
$ !
$ !     Check for RELEASED version.
$ !
$ !     NOTE: use String comparison to check in case version number contains
$ !           characters.
$ !
$   IF "''DECRAM$VMS_VERSION'" .NES. "RELEASED" THEN GOTO DECRAM_UPDATE
$   VER = "V" + "''F$EXTRACT(1,1,DECRAM$REL_VERSION)'" + "." +"''F$EXTRACT(2,2,DECRAM$REL_VERSION)'"
$   TVER = "''F$ELEMENT(1,",",VMI$VMS_VERSION)'"
$   VVER = "V" + "''F$EXTRACT(1,1,TVER)'" + "." + "''F$EXTRACT(2,2,TVER)'"
$   IF "''TVER'" .GES. "''DECRAM$REL_VERSION'" then goto DECRAM_VERSION_OK
$   GOTO DECRAM_VERSION_ERROR
$ !
$ DECRAM_UPDATE:
$ !
$ !     Check for Field Test version.
$ !
$   IF "''DECRAM$VMS_VERSION'" .NES. "UPDATE FT" then goto DECRAM_BASELEVEL
$   VER = "T" + "''F$EXTRACT(1,1,DECRAM$FT_VERSION)'" + "." + -
        "''F$EXTRACT(2,2,DECRAM$FT_VERSION)'"
$   TVER = "''F$ELEMENT(1,",",F$ELEMENT(1," ",VMI$VMS_VERSION))'"
$   VVER = "T" + "''F$EXTRACT(1,1,TVER)'" + "." + "''F$EXTRACT(2,2,TVER)'"
$   IF "''TVER'" .GES. "''DECRAM$FT_VERSION'" THEN GOTO DECRAM_VERSION_OK
$ !
$DECRAM_VERSION_ERROR:
$ !
$   VMI$CALLBACK    MESSAGE E BADVMSVER -
     "DECram ''DECRAM$VERSION' requires OpenVMS version ''VER' or
                       later to install"
$   EXIT VMI$_FAILURE
$ !
$DECRAM_BASELEVEL:
$ !
$ !     Always allow installations on baselevels of OpenVMS.  Required to ensure
$ !     early testing of the product by Digital
$ !
$   VER = "''F$ELEMENT(1,",",F$ELEMENT(1," ",VMI$VMS_VERSION))'"
$   VVER = "''F$EXTRACT(1,1,VER)'" + "." + "''F$EXTRACT(2,2,VER)'"
$   IF "'DECRAM$VMS_VERSION'" .NES. "UPGRADE FT" THEN EXIT VMI$_FAILURE
$   VMI$CALLBACK    MESSAGE I VMBLOK -
        "Installing DECram ''DECRAM$VERSION' on OpenVMS baselevel ''VVER'"
$ DECRAM_LATEST = 1
$ !  Set LATEST to TRUE force installation of V54 version of DECRAM
$   GOTO DECRAM_VERSION_DONE
$ !
```

(continued on next page)

Example B–1 (Cont.) Sample DIGITAL OpenVMS KITINSTAL.COM

```
$ DECRAM_VERSION_OK:
$ !
$   VMI$CALLBACK    MESSAGE I VMSOK -
        "Installing DECram ''DECRAM$VERSION' on OpenVMS Version ''VVER'"
$ DECRAM_LATEST = 0
$ !  Set LATEST to False use VVER to determine what to install
$   VMI$CALLBACK CHECK_VMS_VERSION DECRAM_ADDCLD 5.3-1 S 5.3-2
$
$DECRAM_VERSION_DONE:
$  ON WARNING THEN GOTO ERR_EXIT
$  ON ERROR THEN GOTO ERR_EXIT ! back to normal error mode
$ !
$ !     Check for enough free blocks on system disk.
$ !     Need a minimum of (?).
$ !
$ DECRAM$MIN_BLOCK == 1300
$ VMI$CALLBACK CHECK_NET_UTILIZATION DECRAM$ 'DECRAM$MIN_BLOCK'
$ IF .NOT. DECRAM$ THEN VMI$CALLBACK MESSAGE E NOSPACE -
     "This kit requires at least ''DECRAM$MIN_BLOCK' free disk blocks."
$ IF .NOT. DECRAM$ THEN EXIT VMI$_FAILURE
$ !
$ !
$ !     Let VMSINSTAL ask about the files to be purged.
$ !
$ VMI$CALLBACK SET PURGE ASK
$       vmi$callback SET IVP ASK
$ !
$ ! License Management Facility Registration
$ !
$ IF DECRAM_ARCH .EQ. 1 THEN GOTO DECRAM_AXP_LICENSE
$ ON ERROR THEN GOTO license_err   ! don't quit if license not installed
$       vmi$callback CHECK_LICENSE -
                        DECRAM$license_confirmed -
                        DECRAM -
                        DEC -
                        1.0-           ! Version (updated every release)
                        24-Jun-1991     ! Product Release Date
$       if DECRAM$license_confirmed then goto license_confirmed
$ !
$ license_err:
$   ON ERROR THEN GOTO ERR_EXIT     ! back to normal error mode
$ TYPE SYS$INPUT

       A license for DECram has not been registered and loaded.

                DECram will not be installed.

$       goto DECRAM_NO_INSTALL
$ !
$!
$ DECRAM_AXP_LICENSE:
$ ON ERROR THEN GOTO license_err   ! don't quit if license not installed
$       vmi$callback CHECK_LICENSE -
                        DECRAM$license_confirmed -
                        AV-DECRAM -
                        DEC -
                        1.0-           ! Version (updated every release)
                        24-Jun-1991     ! Product Release Date
$       if DECRAM$license_confirmed then goto license_confirmed
$ !
$ goto license_err
$!
$license_confirmed:
$   ON ERROR THEN GOTO ERR_EXIT     ! back to normal error mode
$license_not_confirmed:
$   ON WARNING THEN GOTO DECRAM_NOT_INSTALLED
$   TYPE SYS$INPUT
```

(continued on next page)

Example B–1 (Cont.) Sample DIGITAL OpenVMS KITINSTAL.COM

```
        The installation will check for the DECram driver already
        installed in the system.  If there is no driver installed in the
        system you will receive the following message:

        "%SYSTEM-W-NOSUCHDEV, no such device available"

        This message can be ignored, the installation will proceed normally.

$   SHOW DEVICE MD:
$   DECRAM_INSTALLED = 1
$   GOTO DECRAM_CONTINUE
$ !
$DECRAM_NOT_INSTALLED:
$   DECRAM_INSTALLED = 0
$ !
$DECRAM_CONTINUE:
$   ON WARNING THEN GOTO ERR_EXIT
$   IF .not. DECRAM_INSTALLED THEN GOTO DECRAM_ASK_IVP
$ !
$   TYPE SYS$INPUT

        A DECram Driver is already installed on the system.

        It is recommended that you reboot your system, rather than
        doing a SYSGEN RELOAD to cause the new DECram Driver
        software to be invoked.
$!
$   DECRAM_IVP = 0
$   GOTO DECRAM_NO_ASK_IVP
$ !
$DECRAM_ASK_IVP:
$ !
$DECRAM_NO_ASK_IVP:
$!
$ DECRAM_DOCS:
$!  Here's where we move in the documentation
$  VMI$CALLBACK RESTORE_SAVESET B
$  VMI$CALLBACK PROVIDE_FILE FUD3 DECRAM_INSTAL_GUIDE.PS -
                       VMI$ROOT:[SYSHLP]
$  VMI$CALLBACK PROVIDE_FILE FUD4 DECRAM_INSTAL_GUIDE.TXT -
                       VMI$ROOT:[SYSHLP]
$  VMI$CALLBACK PROVIDE_FILE FUD5 DECRAM_DRIVER_MANUAL.PS -
                       VMI$ROOT:[SYSHLP]
$  VMI$CALLBACK PROVIDE_FILE FUD6 DECRAM_DRIVER_MANUAL.TXT -
                       VMI$ROOT:[SYSHLP]
$!  Here's where we move in the examples
$  VMI$CALLBACK RESTORE_SAVESET C
$!  Don't do EXAMPLES for File Test
$!  VMI$CALLBACK PROVIDE_FILE FUD7 RAMDISK$STARTUP.COM -
$!                       VMI$ROOT:[SYSHLP.EXAMPLES]
$!  VMI$CALLBACK PROVIDE_FILE FUD8 RAMDISK$NODE.DAT -
$!                       VMI$ROOT:[SYSHLP.EXAMPLES]
$!  VMI$CALLBACK PROVIDE_FILE FUD8 SCRATCHRAM$STARTUP.COM -
$!                       VMI$ROOT:[SYSHLP.EXAMPLES]
$!  VMI$CALLBACK PROVIDE_FILE FUD8 SCRATCHRAM$NODE.DAT -
$!                       VMI$ROOT:[SYSHLP.EXAMPLES]
$!  VMI$CALLBACK PROVIDE_FILE FUD9 DECRAM$CONFIG.COM -
$!                       VMI$ROOT:[SYSHLP.EXAMPLES]
$!
$  IF DECRAM_ARCH .EQ. 1 THEN GOTO DECRAM_AXP
$  DECRAM_LINK = 1
$  IF DECRAM_LATEST THEN GOTO DECRAM_LOADLATEST
$  IF VVER .EQS. "V5.2" THEN GOTO DECRAM_LOAD52
$  IF VVER .EQS. "V5.3" THEN GOTO DECRAM_LOAD53
$ DECRAM_LOADLATEST:
$  VMI$CALLBACK RESTORE_SAVESET F
$  DECRAM_CLD = 0
$  GOTO DECRAM_INST_COM
```

(continued on next page)

Example B–1 (Cont.) Sample DIGITAL OpenVMS KITINSTAL.COM

```
$ DECRAM_LOAD53:
$  VMI$CALLBACK RESTORE_SAVESET E
$  DECRAM_CLD = 1
$!  VMI$CALLBACK RENAME_FILE V53X_INIT.CLD INIT.CLD
$  GOTO DECRAM_INST_COM
$ DECRAM_LOAD52:
$  VMI$CALLBACK RESTORE_SAVESET D
$  DECRAM_CLD = 1
$  GOTO DECRAM_INST_COM
$ DECRAM_AXP:
$  VMI$CALLBACK RESTORE_SAVESET G
$  DECRAM_CLD = 0
$  DECRAM_LINK = 0
$  GOTO DECRAM_INST_COM
$ DECRAM_INST_COM:
$ VMI$CALLBACK TELL_QA "DECRam Executables/IVP are placed in SYS$SPECIFIC."
$ IF DECRAM_LINK .EQ. 0 THEN GOTO DECRAM_NOLINK
$ LINK/NOSYSSHR/NOTRACEBACK/NODEBUG/SHARE=VMI$KWD:MDDRIVER/CONTIGUOUS -
 /MAP=VMI$KWD:MDDRIVER/FULL/CROSS -
 VMI$KWD:MDDRIVER, -
 sys$system:SYS.STB/SELECTIVE,VMI$KWD:MDDRIVER/OPTION
$ DECRAM_NOLINK:
$ IF .NOT. DECRAM_CLD THEN GOTO DECRAM_INST_COM05
$ IF .NOT. DECRAM_ADDCLD THEN GOTOT DECRAM_INST_COM01
$  VMI$CALLBACK PROVIDE_DCL_COMMAND V53X_INIT.CLD
$  GOTO DECRAM_INST_COM02
$ DECRAM_INST_COM01:
$  VMI$CALLBACK PROVIDE_DCL_COMMAND INIT.CLD
$ DECRAM_INST_COM02:
$  VMI$CALLBACK PROVIDE_IMAGE FUD1 DECRAM$SIZE.EXE VMI$ROOT:[SYSEXE] 0
$ DECRAM_INST_COM05:
$  VMI$CALLBACK PROVIDE_FILE FUD2A DECRAM$IVP.COM VMI$ROOT:[SYSTEST] 0
$  IF DECRAM_ARCH .EQ. 1 THEN GOTO DECRAM_INST_AXP
$  VMI$CALLBACK PROVIDE_IMAGE FUD2 MDDRIVER.EXE VMI$ROOT:[SYS$LDR] 0
$  GOTO INST_SUCCESS
$ DECRAM_INST_AXP:
$  VMI$CALLBACK PROVIDE_IMAGE FUD2 SYS$MDDRIVER.EXE VMI$ROOT:[SYS$LDR] 0
$  GOTO INST_SUCCESS
$ INST_SUCCESS:
$ !
$ ! Successful exit
$ !
$ EXIT VMI$_SUCCESS
$ !
$ DECRAM_NO_INSTALL:
$ EXIT VMI$_FAILURE
$ ERR_EXIT:
$ S = $STATUS
$ EXIT S
$ !
$ CONTROL_Y:
$ VMI$CALLBACK CONTROL_Y
$ !
$ DECRAM_RUN_IVP:
$ @sys$test:decram$ivp
$ if $status then exit vmi$_success
$ exit vmi$_failure
```

C

Product-Specific Callback Conventions

A callback is a recursive invocation of a specialized subroutine in VMSINSTAL. Example C–1 in this appendix provides a product-specific callback procedure used with the VMSINSTAL PRODUCT callback. (See Section 5.19 for more information about the PRODUCT callback.)

Assume that NEWAID is the hypothetical base product for a product group. NEWAID must provide a command procedure containing a callback (INCREMENT), which increments quantities passed to it.

The command procedure should be named NEWAIDINSTALL.COM to reflect the product name, followed by the word INSTALL. The command procedure should be stored in SYS$UPDATE.

To enable another product to invoke the INCREMENT callback, it must include the following line in its installation procedure:

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

In this example, NEWAIDINSTALL.COM parses the following parameters:

- P1—INCREMENT
- P2—PROD$INC
- P3—COUNTFILE.DAT
- P4—VMI$ROOT:[SYSUPD]
- P5—2
- P6—""
- P7—""
- P8—""

The parameter order for product-specific callbacks should follow those of standard callbacks. Use a standard callback that is similar to the one you are designing as a model for parameter order.

The following conventions must be adhered to when designing a product-specific callback procedure:

- 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. Use a GOTO statement with this parameter to branch to the appropriate callback.

- 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$, the VMSINSTAL facility code, because the callback procedure is a logical extension of VMSINSTAL.
- The return status from a standard callback must be checked with an IF statement to determine success or failure. Because a failure indication is limited to a warning, the error handler will not be invoked.
- The callback must return either a VMI$_SUCCESS or VMI$_FAILURE status.

Example C–1 demonstrates a product-specific callback procedure.

Example C–1 Product_Specific Callback Procedure

```
$! First, 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
$
$! Use the first parameter to branch to the desired callback.
$!
$ 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$FIND '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
$
```

(continued on next page)

Example C–1 (Cont.) Product_Specific Callback Procedure

```
$! 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
```

D

How to Use the VMI$VMS_VERSION Symbol

Example D–1 shows some techniques for using the VMI$VMS_VERSION symbol to determine whether your product is compatible with the version level of the target system. Although this example only checks for a specific version, the techniques shown here may be adapted to check for a range of versions.

For more information about the CHECK_VMS_VERSION callback, see Section 5.6.

Example D–1 Template for Using the VMI$VMS_VERSION Symbol

```
$ !     ************************************************
$ !     *                                              *
$ !     *            VMI$VMS_VERSION Template          *
$ !     *                                              *
$ !     ************************************************
$ !
$ !     Modify the definitions of PRODUCT$REL_VERSION and
$ !     PRODUCT$FT_VERSION to define any version restrictions required
$ !     for your product.
$ !
$ !
$ !     Define earliest released version of OpenVMS supported by the product
$ !
$       PRODUCT$REL_VERSION = "040"
$ !
$ !     Define earliest field test version of OpenVMS supported by the product
$ !
$       PRODUCT$FT_VERSION = "041"
$ !
$ !     Extract current running version number of OpenVMS
$ !
$       PRODUCT$TYPE = F$ELEMENT(0,",",VMI$VMS_VERSION)
$ !
$V_RELEASED:
$ !
```

(continued on next page)

Example D–1 (Cont.) Template for Using the VMI$VMS_VERSION Symbol

```
$ !     Check for RELEASED version of OpenVMS
$ !
$ !     NOTE: use String comparison to check in case version number contains
$ !           characters.
$ !
$       IF PRODUCT$TYPE .NES. "RELEASED" THEN GOTO V_UPDATE
        IF  F$ELEMENT(1," ",F$ELEMENT(1,",",VMI$VMS_VERSION)) .GES. -
           PRODUCT$REL_VERSION THEN GOTO V_OK
$ !
$V_REL10:
$ !
$       VER="V"+F$EXTRACT(1,1,PRODUCT$REL_VERSION) -
             +"."+F$EXTRACT(2,2,PRODUCT$REL_VERSION)
$       VMI$CALLBACK MESSAGE E VERSION -
      "THIS PRODUCT REQUIRES OpenVMS VERSION ''VER' OR LATER TO INSTALL"
$       EXIT VMI$_FAILURE
$ !
$V_UPDATE:
$ !
$ !     Check for Field Test version of OpenVMS
$ !
$       IF PRODUCT$TYPE .NES. "UPDATE FT" THEN GOTO V_BASELEVEL
        IF  F$ELEMENT(1," ",F$ELEMENT(1,",",VMI$VMS_VERSION)) .GES. -
           PRODUCT$FT_VERSION THEN GOTO V_OK
$ !
$V_UPD10:
$ !
$       VER="T"+F$EXTRACT(1,1,PRODUCT$FT_VERSION) -
             +"."+F$EXTRACT(2,2,PRODUCT$FT_VERSION)
$       VMI$CALLBACK MESSAGE E FIELDTEST -
             "THIS PRODUCT REQUIRES OpenVMS VERSION ''VER' OR LATER TO INSTALL"
$       EXIT VMI$_FAILURE
$ !
$ !
$V_BASELEVEL:
$ !
$ !     Always allow installations on baselevels of OpenVMS.  Required to ensure
$ !     early testing of the product by Digital
$ !
$       BASELEVEL = F$ELEMENT(1," ",F$ELEMENT(1,",",VMI$VMS_VERSION))
$       IF PRODUCT$TYPE .NES. "UPGRADE FT" THEN EXIT VMI$_FAILURE
$       VMI$CALLBACK MESSAGE I BASELEVEL -
        "INSTALLING ON OpenVMS BASELEVEL ''BASELEVEL', INSTALLATION CONTINUING."
$ !
$ !
$ V_OK:
```

E

Product Registration

Product registration is a free service offered by Digital to customers. Product registration prevents software conflicts among layered products by ensuring that the following elements of each product are unique:

- Facility code
- Logical name prefixes
- Shareable image names

Participation in registration is voluntary. However, Digital cannot guarantee that nonregistered products will not conflict with other products.

For more information about product registration (including an application form), contact the Registrar at the following address:

Digital Equipment Corporation
Attn: OpenVMS Product Registrar
110 Spit Brook Road
Nashua, NH 03062-9987

Index

A

ADD_IDENTIFIER callback, 5–2
ALL_DONE subroutine, 4–7
Alternate root option (R), 1–7, 1–10
Alternate working device option (AWD), 1–9
Answer file, 1–8
ASK callback, 5–2
Auto-answer option (A), 1–8, 4–6

B

Booting option (B), 1–14

C

Callbacks
  accessing files, 3–3
  ADD_IDENTIFIER, 5–2
  ASK, 5–2
  CHECK_NETWORK, 5–5
  CHECK_NET_UTILIZATION, 5–5
  CHECK_PRODUCT_VERSION, 5–7
  CHECK_VMS_VERSION, 5–8
  COMPARE_IMAGE, 5–9
  CONTROL_Y, 5–10
  CREATE_ACCOUNT, 5–11
  CREATE_DIRECTORY, 5–11
  definition, 1–1
  DELETE_FILE, 5–14
  examples, 3–7
  FIND_FILE, 5–15
  for creating directories, 3–5
  for deleting files, 3–5
  for moving files, 3–4
  for updating files, 3–4
  for updating libraries, 3–5
  GET_IMAGE_ID, 5–16
  GET_PASSWORD, 5–17
  GET_SYSTEM_PARAMETER, 5–17
  guidelines for, 5–1
  invoking, 3–1
  MESSAGE, 5–18
  parameter restrictions, 5–1
  PATCH_IMAGE, 5–19
  PRINT_FILE, 5–20
  PRODUCT, 5–21
Callbacks (cont'd)
  product-specific, 3–1
    conventions for, C–1
  PROVIDE_DCL_COMMAND, 5–21
  PROVIDE_DCL_HELP, 5–22
  PROVIDE_FILE, 5–23
  PROVIDE_IMAGE, 5–24
  RENAME_FILE, 5–27
  RESTORE_SAVESET, 5–27
  RUN_IMAGE, 5–28
  SECURE_FILE, 5–29
  SET, 5–30
  SET PRODUCT_NAME, 5–33
  SET_SEMANTICS, 5–36
  specifying keywords for, 5–1
  SUMSLP_TEXT, 5–38
  TELL_QA, 5–40
  tracing, 1–11, 4–6
  UNWIND, 5–40
  UPDATE_ACCOUNT, 5–40
  UPDATE_FILE, 5–41
  UPDATE_IDENTIFIER, 5–42
  UPDATE_LIBRARY, 5–43
Callback trace option (C), 1–11, 4–6
CHECK_NETWORK callback, 5–5
CHECK_NET_UTILIZATION callback, 5–5
CHECK_PRODUCT_VERSION callback, 5–7
CHECK_VMS_VERSION callback, 5–8
Command abbreviation restriction, 2–3
COMPARE_IMAGE callback, 5–9
Compatibility mode, 2–4
CONTROL_Y callback, 5–10
CREATE_ACCOUNT callback, 5–11
CREATE_DIRECTORY callback, 5–11
  restrictions, 3–5
Creating diskette kit, 2–2
Creating magnetic tape kit, 2–3
Creating TU58 cartridge kit, 2–3

D

DCL command abbreviation
  restriction, 2–3
Debugging tools, 1–11
Defaults
  KITINSTAL, 3–2

Defer file, 1–13
DELETE_FILE callback, 5–14
Device number
  restriction, 5–1
Directory
  creating using callbacks, 3–5
  restrictions on creating, 3–5
Diskette kit, 2–2
  specifying BACKUP qualifiers, 2–2
Disk space utilization, 1–11, 1–13, 5–5, 5–35

E

Error handling
  KITINSTAL, 2–3

F

Facility code, 2–1
  registration, 2–1
Failures, 1–13, 4–8
File log option (L), 1–9
File protection, 2–2
Files
  marker, 1–13, 4–5
  using callbacks to access, 3–3
  using callbacks to delete, 3–5
  using callbacks to move, 3–4
  using callbacks to update, 3–4
FIND_FILE callback, 5–15

G

Get save set option (G), 1–7, 1–9, 4–10
GET_IMAGE_ID callback, 5–16
GET_PASSWORD callback, 5–17
GET_SYSTEM_PARAMETER callback, 5–17
Global state
  changing, 2–4

H

Help libraries, 2–6
History file, 1–12

I

Inhibit initial prompts option (I), 1–11
Installation
  achieving design goals, 1–1
  creating a work directory, 1–5
  installation phase, 3–1
  IVP phase, 2–4, 3–2
Installation Verification Procedure
  See IVP
Install utility (INSTALL), 2–4
Internationalization
  generating VMSINSTAL messages for, 2–5
IVP (installation verification procedure)
  definition, 1–2
IVP (Installation Verification Procedure), 3–2
  exit requirements, 3–3
  guidelines for, 2–4

K

Kit debug option (K), 1–11
KITINSTAL procedure
  compatibility mode, 2–4
  defaults, 3–2
  design specifications, 3–6
  error handling requirement, 2–3
  example, 3–7, B–1
  guidelines and conventions, 3–1
    IVP phase, 3–2
  installation for, 3–1
  location in kit, 4–2
  logical references, 3–1
  name prefixes, 2–3
  request codes, 3–6
  requirements of, 3–6
Kits
  diskette, 2–2
  magnetic tape, 2–3
  TU58 cartridge, 2–3

L

Libraries
  using callbacks to update, 3–5
Logical names
  definitions, A–1
  naming conventions, 2–3
  prefixing, 5–1
  use restrictions, 5–1
  VMI$AUTO_FILE, 4–8
  VMI$CALL_FILE, 4–7
  VMI$DEFER_FILE, 1–13, 4–7
  VMI$KWD, 1–5, 3–1, 5–1
  VMI$MARKER_FILE, 1–13, 4–7, 4–8
  VMI$PRODUCT_FILE, 4–7
  VMI$ROOT, 3–1, 4–3, 4–5, 5–1
  VMI$SPECIFIC, 3–1, 4–3
  VMI$TEMP_FILE, 4–7
  VMI$TERMINAL_FILE, 4–8

M

Magnetic tape kit
  specifying BACKUP qualifiers, 2–3
Marker file, 1–13, 4–5
MESSAGE callback, 5–18

Messages
internationalization of, 2–5
Modes
compatibility, 2–4

N

Naming convention
global symbol and logical name, 2–3, 3–1
KITINSTAL, 3–1
product, 2–1
release notes, 2–6
save set, 2–1
subroutine, 2–3
Network
checking status of, 5–5

O

Options
See VMSINSTAL command procedure, Options

P

PATCH_IMAGE callback, 5–19
PRINT_FILE callback, 5–20
Procedure
checking products installed, 1–12
PRODUCT callback, 5–21
Product file
protection code, 2–2
Product identification, 2–1
Product installation file, 1–12
Product kit, 1–2, 2–1
Product name registration, 2–1
Product referencing
options, 2–4
Product save set identification, 2–1
Product-specific callback, 3–1, C–1
Prompting for input, 2–5, 5–2
Protection code
for product files, 2–2
PROVIDE_DCL_COMMAND callback, 5–21
PROVIDE_DCL_HELP callback, 5–22
PROVIDE_FILE callback, 5–23
PROVIDE_IMAGE callback, 5–24

Q

QA (Quality Assurance) mode option (Q), 1–11

R

Release notes
guidelines for including, 2–6
in Help library, 2–6
Release notes option (N), 1–2, 1–9, 2–2, 2–6
RENAME_FILE callback, 5–27
Request code
KITINSTAL, 3–6
VMI$_IVP, 3–3
Restore save set and pause option (RSP), 1–11
RESTORE_SAVESET callback, 5–27
Restrictions
callback, 2–3, 3–1
CREATE_DIRECTORY callback, 3–5
logical name use, 5–1
Root
specifying alternate, 1–7
RSP option
See Restore save set and pause option (RSP)
RUN_IMAGE callback, 5–28

S

Safety mode, 1–13
Save set
copying to disk, 1–9, 4–10
identification, 1–2
naming, 2–1
restoring, 1–11, 4–6
sequence identifier, 2–2
SECURE_FILE callback, 5–29
Sequence identifier
See Save set
SET ACL option, 5–2, 5–30
SET ASK_CASE option, 5–31
SET callback, 5–30 to 5–37
SET command
restriction, 3–2
SET IVP option, 5–32
SET POSTINSTALL option, 5–33
SET PRODUCT_NAME option, 5–33
SET PURGE option, 5–34
SET REBOOT option, 5–35
SET SAFETY option, 5–35
SET SHUTDOWN option, 5–36
SET STARTUP option, 5–37
SET_FILE option, 5–31
SET_SEMANTICS option, 5–36
SHOW command
restriction, 3–2
SPKITBLD command procedure, 1–2
invoking, 1–4
Statistics option (S), 1–11, 4–5
Statistics subprocess, 4–5
Subroutines
See also Callbacks
ALL_DONE, 4–7
limitations, 2–3
naming conventions, 2–3

SUMSLP_TEXT callback, 5–38
Symbols
  definitions, A–1
  naming conventions, 2–3
SYS$HELP, 2–6
SYS$LIBRARY
  restriction, 3–1
SYS$SPECIFIC
  restriction, 3–1
System failure
  recovery, 4–8
  Recovery, 1–13

T

TELL_QA callback, 5–40
TU58 cartridge kit
  specifying BACKUP qualifiers, 2–3

U

UNWIND callback, 5–40
UPDATE_ACCOUNT callback, 5–40
UPDATE_FILE callback, 5–41
UPDATE_IDENTIFIER callback, 5–42
UPDATE_LIBRARY callback, 3–5, 5–43

V

Verification
  disabling, 1–11
  using SET commands, 3–2
VMI$, 4–1
VMI$ALTERNATE_ROOT, 4–5
VMI$BOOTING, 4–2, 4–8
VMI$CALLBACK, 3–1, 4–7
  for invoking callbacks, 5–1
VMI$COMMON_ROOT, 4–5
VMI$DEFER_FILE, 1–13
VMI$FREE_BLOCKS, 4–5
VMI$KWD, 3–1, 5–1
VMI$KWD_FREE_BLOCKS, 4–5
VMI$LIST, 4–4
VMI$MARKER_FILE, 1–13
VMI$PLACE, 4–4
VMI$ prefix, 2–3
VMI$PRODUCT, 4–5
VMI$ROOT, 3–1, 5–1
VMI$SAVED_DIR, 4–8
VMI$SAVED_MSG, 4–8
VMI$SAVED_PRIVS, 4–8
VMI$SAVED_PROT, 4–8
VMI$SAVED_UIC, 4–8
VMI$SPECIFIC, 3–1, 4–5
VMI$TERMINAL_FILE, 4–2
VMI$VERSION, 4–2
VMI$VMS_VERSION, 4–2
  guidelines for using, D–1
VMI$_FAILURE status, 5–1
VMI$_IVP request code, 3–3
VMI$_SUCCESS status, 5–1
VMIDEFER.COM, 1–13
VMIMARKERpid.DAT, 1–13
VMSINSTAL command procedure
  alternate root, 1–10
  answer file, 1–8
  functional steps, 4–1 to 4–10
  guidelines and conventions, 2–1 to 2–5
  history file, 1–12
  installation phase, 1–13, 3–1
  invoking, 1–6
  IVP phase, 2–4
  list installed products procedure, 1–12
  logic sequence, 4–1
  name prefixes, 2–3
  options, 1–8 to 1–11
    developer's, 1–11
    installer's, 1–8
  OPTIONS keyword, 1–6
  outline of logic, 1–5
  parameter, 1–6 to 1–7
  product installation file, 1–12
  recovery procedure, 4–8
  safety mode, 1–13
Volume labeling
  conventions, 2–2

W

Wildcards
  restrictions, 5–1
Work directory
  installation, 1–5
  on an alternate device, 1–9
  VMI$KWD, 1–5