POLYCENTER Software Installation Utility Developer's Guide

December 1995

This guide describes how to package software products using the POLYCENTER Software Installation utility. It describes the product description language, product description files, product text files, and other relevant concepts.

Revision/Update Information: This guide supersedes the *POLYCENTER Software Installation Utility Developer's Guide*, Version 6.1

Software Version: OpenVMS Alpha Version 7.0
OpenVMS VAX Version 7.0

Digital Equipment Corporation
Maynard, Massachusetts

December 1995

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Digital or an authorized sublicensor.

Digital conducts its business in a manner that conserves the environment and protects the safety and health of its employees, customers, and the community.

© Digital Equipment Corporation 1995. All rights reserved.

The following are trademarks of Digital Equipment Corporation: AXP, Bookreader, DEC, DEC Ada, DEC Fortran, DEC Rdb, DECdirect, DECnet, DECprint, DECwindows, Digital, MicroVAX, OpenVMS, POLYCENTER, Rdb/VMS, RSX, VAX, VAXcluster, VMS, VMScluster, and the DIGITAL logo.

The following are third-party trademarks:

Hewlett-Packard is a registered trademark of Hewlett-Packard Company.

Motif, OSF, OSF/1, OSF/Motif, and Open Software Foundation are registered trademarks of the Open Software Foundation, Inc.

NFS is a registered trademark of Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Ltd.

All other trademarks and registered trademarks are the property of their respective holders.

ZK5952

This document is available on CD–ROM.

Contents

Preface vii

Part I Concepts

1 Overview

1.1 Features for Software Providers 1–1
1.2 Features for System Managers 1–1
1.3 Coexistence with VMSINSTAL 1–2
1.4 Available Interfaces 1–2
1.5 Packaging a Product 1–2
1.5.1 Product Files 1–3
1.5.2 Product Database 1–3
1.5.3 Format of Software Product Kits 1–4
1.5.4 Software Product Name Conventions 1–5
1.5.4.1 Looking at Software Product Name Examples 1–6
1.5.5 Logical Names 1–7
1.6 Packaging a Platform 1–7
1.7 Managed Objects 1–8
1.7.1 Managed Object Conflict 1–8
1.7.2 Managed Object Replacement and Merging 1–9
1.7.3 Managed Object Scope and Lifetime 1–10

2 Creating the Product Description File

2.1 General Guidelines 2–2
2.2 Defining Your Environment 2–2
2.3 PDF File Naming Conventions 2–5
2.4 Structure of a PDF 2–5
2.4.1 Overview of PDL Statements 2–6
2.4.2 PDL Statement Syntax 2–7
2.4.3 PDL Function Syntax and Expressions 2–8
2.4.4 PDL Data Types and Values 2–9
2.5 Writing a PDF for a Full Kit 2–10
2.6 Writing a PDF for a Platform Kit 2–14
2.7 Writing a PDF for a Transition Kit 2–14

3 Creating the Product Text File

3.1 PTF File Naming Conventions 3–1
3.2 Structure of a PTF 3–2
3.2.1 Specifying the Product Name 3–2
3.2.2 PTF Modules 3–2
3.2.3 Including Prompt and Help Text 3–3

4 Packaging the Kit

4.1 Packaging with the DCL Interface 4–1
4.2 Product Command Example 4–4

Part II Product Description Language Statements

account PDL–3
apply to PDL–5
bootstrap block PDL–7
directory PDL–8
end PDL–10
error PDL–11
execute install...remove PDL–13
execute login PDL–15
execute postinstall PDL–16
execute release PDL–18
execute start...stop PDL–20
execute test PDL–22
file PDL–23
hardware device PDL–28
hardware processor PDL–30
if PDL–31
infer PDL–33
information PDL–35
link PDL–37
loadable image PDL–39
module PDL–41
network object PDL–43
option PDL–45
part PDL–48
patch image (VAX only) PDL–50
patch text PDL–51
process parameter PDL–52
process privilege PDL–54
product PDL–55
register module PDL–57
remove PDL–59
rights identifier PDL–61
scope PDL–63
software PDL–64

system parameter PDL–67
upgrade PDL–69

A Migrating from VMSINSTAL to the POLYCENTER Software Installation Utility

A.1 VMSINSTAL Options and Equivalents A–1
A.2 VMSINSTAL Callbacks and Equivalents A–2

B Advanced PDF Concepts

B.1 Defining the Scope of a Managed Object B–1
B.2 Updating Files B–2
B.3 Managed Object Lifetimes B–2

Glossary

Index

Examples

2–1 Checktran Product Description File 2–10
2–2 UCX Product Description File 2–11
2–3 Product Description File 2–14
2–4 FMS Product Description File 2–15

Figures

1–1 Package Operation 1–4
1–2 Integrated Platform Example 1–8

Tables

1–1 Format of *tmmnn-ue* Version Identification 1–5
1–2 PDF Kit Types and Values 1–6
2–1 Base Data Types and Values 2–9
2–2 String Data Type Constraints 2–10
4–1 Product Command Format 4–2
PDL–1 Directory Managed Object Scope and Lifetime PDL–9
PDL–2 File Managed Object Scope and Lifetime PDL–26
PDL–3 Link Managed Object Scope and Lifetime PDL–37
PDL–4 Library Types for Module Statement PDL–41
PDL–5 Library Types for Register Module Statement PDL–57
A–1 VMSINSTAL Options and Equivalents A–1
A–2 VMSINSTAL Callbacks and Equivalents A–2

Preface

Intended Audience

This guide is intended for individuals who are responsible for packaging software products.

Document Structure

This guide is organized into two parts, two appendixes, and a glossary as follows:

- Part I contains the chapters that describe the concepts you should be familiar with to create software kits with the POLYCENTER Software Installation utility. It also contains instructions on how to write a product description file and product text file. Part I contains the following chapters:
  - Chapter 1 describes utility concepts.
  - Chapter 2 describes writing the product description file. It also contains sample product descriptions.
  - Chapter 3 describes writing the product text file. It also contains sample product text files.
  - Chapter 4 describes how to package your product.
- Part II describes product description language statements.
- Appendix A contains information about migrating from the VMSINSTAL utility to the POLYCENTER Software Installation utility.
- Appendix B describes some advanced concepts such as managed object scope and lifetime.
- The Glossary lists and defines POLYCENTER Software Installation utility terminology.

Related Documents

The *OpenVMS System Manager's Manual* describes the tasks that system managers perform using the POLYCENTER Software Installation utility. It explains operations such as software installation and removal. It also demonstrates the two POLYCENTER Software Installation user interfaces: the DIGITAL Command Language (DCL) and DECwindows Motif.

For additional information on OpenVMS products and services, access the Digital OpenVMS World Wide Web site. Use the following URL:

`http://www.openvms.digital.com`

Reader's Comments

Digital welcomes your comments on this guide.

Print or edit the online form SYS$HELP:OPENVMSDOC_COMMENTS.TXT and send us your comments by:

Internet	**openvmsdoc@zko.mts.dec.com**
Fax	603 881-0120, Attention: OpenVMS Documentation, ZK03-4/U08
Mail	OpenVMS Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

How To Order Additional Documentation

Use the following table to order additional documentation or information. If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Location	Call	Fax	Write
U.S.A.	DECdirect 800–DIGITAL 800–344–4825	Fax: 800–234–2298	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	809–781–0505	Fax: 809–749–8300	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street, Suite 200 P.O. Box 11038 Metro Office Park San Juan, Puerto Rico 00910–2138
Canada	800–267–6215	Fax: 613–592–1946	Digital Equipment of Canada, Ltd. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	—	—	Local Digital subsidiary or approved distributor
Internal Orders	DTN: 264–4446 603–884–4446	Fax: 603–884–3960	U.S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063–1260

ZK–7654A–GE

Conventions

The name of the OpenVMS AXP operating system has been changed to OpenVMS Alpha. Any references to OpenVMS AXP or AXP are synonymous with OpenVMS Alpha or Alpha.

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

Alpha	The Alpha icon denotes the beginning of information specific to OpenVMS Alpha.

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 Alpha or to OpenVMS VAX.

In this guide, every use of DECwindows and DECwindows Motif refers to DECwindows Motif for OpenVMS software.

The following conventions are also used in this guide:

. . .	Horizontal ellipsis points in examples indicate 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.
( )	In command format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In command 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 command format descriptions, braces indicate 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 guide.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
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.
parameters	Unless otherwise noted, all parameters that you use with product description language (PDL) statements are strings either enclosed in quotation marks or not enclosed in quotation marks.

Part I
Concepts

Part I describes the concepts you should be familiar with to create software kits. It also contains instructions for writing product description and text files.

1
Overview

The POLYCENTER Software Installation utility is a complete software installation and management utility for OpenVMS Alpha or VAX systems. It can package, install, remove, and manage software products on Alpha or VAX systems. It can also save information about software products such as system requirements, installation options, and your answers to questions that are asked during the installation process.

The POLYCENTER Software Installation utility is intended for use both by those who create (package) kits for software products, and by system managers who install and maintain these products.

1.1 Features for Software Providers

For software providers, the POLYCENTER Software Installation utility simplifies the task of packaging software because:

- Installations require less packaging effort than most conventional installation methods. This results in performance gains and reduced development time over conventional installations. The code is also nonprocedural, which means that you do not need to understand the details of how the utility functions.
- You can include both brief and detailed installation text to guide users through an installation, resulting in a higher installation success rate.
- All software and documentation can exist on a single piece of distribution media.
- The utility keeps track of which products (and versions) have been installed and removed in the execution environment. Using this information, you can design your installation procedure so that system managers can use the utility to manage version dependencies.

1.2 Features for System Managers

System managers can use the POLYCENTER Software Installation utility to:

- Install a product
- Extract release notes to a file
- Display the names of product kits found in a specified directory
- Remove an installed product
- Create a product configuration file (PCF) that contains default choices for installation options
- Modify the configuration choices for an installed product
- Register a product that was previously installed with another tool

- Register a change in the volume label for a device that contains installed products
- Copy product kits to a new location and optionally convert them to another format
- Display a list of products installed on a system
- Display information about managed objects (for example, files or directories) created during a software product installation
- Display a log of operations performed on software products

Note that the POLYCENTER Software Installation utility does not perform license management tasks.

1.3 Coexistence with VMSINSTAL

The POLYCENTER Software Installation utility is integrated into OpenVMS and coexists with the VMSINSTAL utility. Today, you use the POLYCENTER Software Installation utility to install the OpenVMS Operating System and some layered products on Alpha systems, and to install some layered products on VAX systems. The POLYCENTER Software Installation utility is the preferred installation mechanism for future layered product and OpenVMS releases.

The POLYCENTER Software Installation utility offers the following features:

- Typically faster installation and upgrade operations
- Removal (deinstallation) of previously installed software products
- A database of information on installed products that has query capabilities on the database
- Two interfaces to the utility: DCL and DECwindows Motif
- Dependency checking of software products based on product version number

If you currently use VMSINSTAL to package your software product, see Appendix A for information about migrating from VMSINSTAL to the POLYCENTER Software Installation utility.

1.4 Available Interfaces

You can perform POLYCENTER Software Installation utility operations from either the DCL interface or from the DECwindows Motif interface. You might already be familiar with the main DCL component of the POLYCENTER Software Installation utility, the PRODUCT command.

When first packaging a software product, you will probably find that the DCL interface is the most convenient to use. This document uses the DCL interface for all examples.

1.5 Packaging a Product

Note

This section introduces you to packaging a product. See Chapter 4 for complete information.

You ‘‘package’’ software components to produce a software product kit. To do this, you use the PRODUCT PACKAGE command in the following format:

```
$ PRODUCT PACKAGE product_name[,...]
     /SOURCE=file-specification
     /DESTINATION=device-name[directory-name]
     /MATERIAL=path-name
```

where *product_name* specifies the name of the software you want to package. Note that you cannot specify hyphens (-) in the *product_name*. Table 4–1 describes the PRODUCT PACKAGE command in detail.

1.5.1 Product Files

Before packaging your product using the POLYCENTER Software Installation utility, you usually create the following components:

- A **product description file** (PDF). You use Product Description Language (PDL) statements to convey all of the information the POLYCENTER Software Installation utility needs for installing either a software product or a set of software products. In addition to identifying all of the files that the product provides, PDL statements can specify configuration choices the product offers, default choices, and product requirements (such as dependencies on other software products, minimum hardware configurations, and system parameter values).
- An optional **product text file** (PTF). It provides information about the product in brief and detailed formats. The information includes product identification, copyright notice, configuration choice descriptions, and message text used primarily during product installation and configuration operations.
- The **product material** consists of the files associated with the product.

After you create these components, you can perform a package operation. The package operation creates a ready-to-install version of your product.

In addition, the system manager can create a **product configuration file** (PCF) using the PRODUCT CONFIGURE command, or as an output of the PRODUCT INSTALL command. A PCF contains responses to some or all of the installation questions for a product. It can provide default or required choices, which may differ from the default choices provided in the PDF. Creating the PCF file is described in the *OpenVMS System Manager’s Manual*.

1.5.2 Product Database

The **product database** (PDB) is created automatically by the POLYCENTER Software Installation utility. When products are installed, the files and other objects that make up the product, such as directories and accounts, are recorded in the PDB. The configuration choices made during installation are also recorded. You can use the product database to determine the status of a product, the history of a product, and what configuration choices were made.

You use the DCL command PRODUCT SHOW to query the product database to show what products are installed and the dependencies between them, to list the files and other objects that make up each product, or to show the history of installation and upgrade activity.

If you remove a product, information about the files and objects associated with the product are removed from the database. However, the history of the product’s activity from installation to removal is retained in the database.

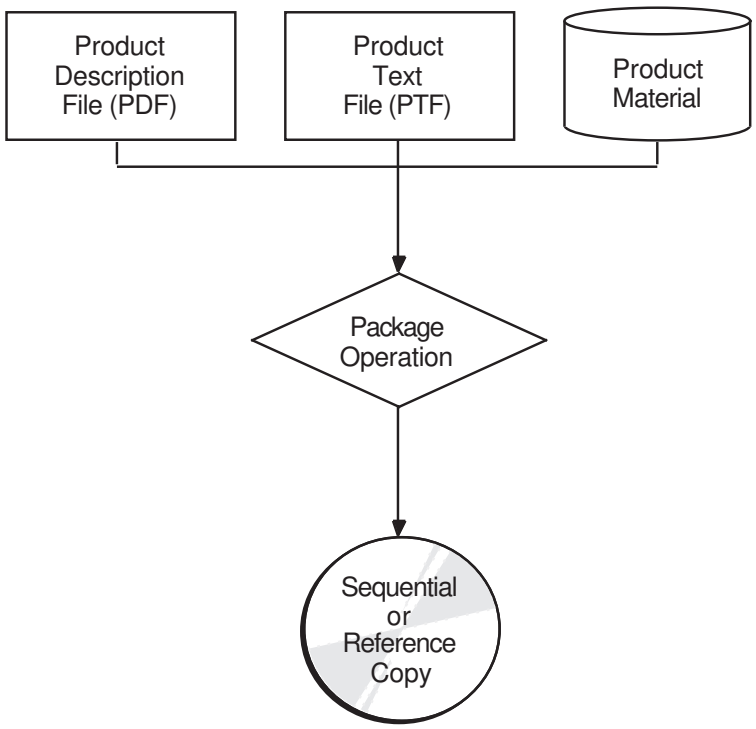
1.5.3 Format of Software Product Kits

Software product kits can be in one of two formats:

- **Sequential copy format**. In this form, the PDF, the PTF, and all files that comprise the product are packaged in a single container file. This container file can be placed either on a random-access device, such as a compact disc, or on a sequential access device, such as a magnetic tape. Most layered products are distributed in sequential copy format.
- **Reference copy format**. In this form, the PDF, the PTF, and all files that comprise the product are placed in a directory tree on a random-access device. OpenVMS is distributed in reference copy format on CD–ROM.

Figure 1–1 shows how the package operation uses the PDF, PTF, and product material to create a reference or sequential copy.

Figure 1–1 Package Operation



Although you provide the PDF and PTF as input to the package operation, they also exist in modified form in the sequential or reference kit you create.

The package operation changes the format of the output PTF (for more information, see Section 3.2). The output PDF is in the same format as the input PDF, but the utility may modify statements in the output PDF. For example, the package operation adds the size option to *file* statements in the output PDF.

1.5.4 Software Product Name Conventions

A software product kit packaged in sequential copy format has a container file named in the following format:

```
producer-base-product-version-kit_type.PCSI
```

A software product kit packaged in reference copy format has a product description file in the root directory named in the following format:

```
producer-base-product-version-kit_type.PCSI$DESCRIPTION
```

Each subfield in the file name is separated by a hyphen. The length of the file name string (including all required hyphens) cannot exceed 39 characters. In addition, the `producer-base-product` portion of the string must uniquely identify the software product. The subfields are defined as follows:

- *producer* is the legal owner of the software product. For Digital software products this part of the PDF file name is DEC.
- *base* denotes the hardware and software combination that the product requires. For OpenVMS Alpha systems use AXPVMS; for OpenVMS VAX systems use VAXVMS.
- *product* is the name of the software product.
- *version* identifies the version of the software product expressed in tmmnn-ue format, as shown in Table 1–1.

Table 1–1 Format of *tmmnn-ue* Version Identification

t	The type of version (a single uppercase alphabetic character) in the range of A through Z; W through Z are reserved to Digital.
mm	The major version number (decimal integer 01 through 99).
nn	The minor version number (decimal integer 00 through 99).
-	The hyphen is required in all cases. When both update level (u) and maintenance edit level (e) are omitted, the version will end with a hyphen and the file name will have a double hyphen (- -) preceding the kit type.
u	The update level (decimal integer 1 through 999). The level is optional.
e	The maintenance edit level (one or more alphanumeric characters beginning with an alphabetic character). This level is optional.

- *kit_type* identifies a kit type specified as a value from 1 through 7, as shown in Table 1–2.

Table 1–2 PDF Kit Types and Values

Value	Type of Kit	Description
1	Full	Application software (complete kit).
2	Operating system	Operating system software (complete kit).
3	Partial	An update to currently installed software that replaces or provides some of the product's files. The version is changed.
4	Patch	A correction to currently installed software. The version is not changed.
5	Platform	An integrated set of software products.
6	Transition	Used to register a product that was not installed by the POLYCENTER Software Installation utility. Transition kits include only a product definition file and (optionally) a product text file; they do not provide product material.
7	Mandatory update	A required correction to currently installed software. Functionally the same as a patch kit. The version is not changed.

Reading a Displayed Version

The *tmmnn-ue* format used in file names is very similar to the format used to display versions or to enter versions using the /VERSION qualifier. However, when versions are displayed, leading zeros are omitted in *mm* and *nn*, and if neither *u* nor *e* is present, the hyphen (-) is omitted. When you read a displayed version, keep the following guidelines in mind:

- If a hyphen is present and the first character after the hyphen is a digit, then the leading digits after the hyphen are the update level. If nondigit characters are present, the maintenance edit level consists of the first nondigit character and all following characters. If nondigit characters are not present, the maintenance edit level is blank.
- If a hyphen is present and the first character after the hyphen is a nondigit character, the update level is zero (0) and the maintenance edit level consists of all of the characters after the hyphen.
- If no hyphen is present, the update level is zero and the maintenance edit level is blank.

1.5.4.1 Looking at Software Product Name Examples

The following examples illustrate product naming conventions:

- A sequential copy format kit for DEC Softwindows for OpenVMS VAX has the following format:

  ```
  DEC-VAXVMS-SOFTWIN-V0101--1.PCSI
  ```

  This format shows that the *producer* is DEC (Digital), the *base* is VAXVMS (OpenVMS VAX), the *product* is SOFTWIN, and the *version* is V1.1. The type of version is V, the major and minor version numbers are each 1. There are no update or maintenance edit levels, and a double hyphen precedes the kit type. The *kit_type* is 1 (full).

- A product description file in a reference copy format kit for OpenVMS Alpha has the following format:

  ```
  DEC-AXPVMS-VMS-V0602-1H2-2.PCSI$DESCRIPTION
  ```

  This format shows that the *producer* is DEC (Digital), the *base* is AXPVMS (OpenVMS Alpha), the *product* is VMS (OpenVMS), and the *version* is V6.2-1H2. The type of version is V, the major version number is 6, the minor version number 2, the *update level* is 1, and the *maintenance edit level* is H2. The *kit_type* is 2 (operating system).

- A sequential copy format for WXYZ Corporation's Super Draw product for OpenVMS Alpha has the following format:

  ```
  WXYZ-AXPVMS-SUPER_DRAW-T0205-14-1.PCSI
  ```

  This format shows that the *producer* is WXYZ, the *base* is AXPVMS (OpenVMS Alpha), the *product* is Super Draw, and the *version* is V2.5. The type of version is T, the major version number is 2, and the minor version number is 5. The *update level* is 14 and the *maintenance edit level* is not specified (blank). The *kit_type* is 1 (full).

1.5.5 Logical Names

When installing your product, system managers must specify a location where the software kit resides and a location in which to install the software. Two methods are available for identifying these locations:

- Defining logical names
- Specifying /SOURCE and /DESTINATION qualifiers on the command line, or selecting a source and destination when using DECwindows Motif interface.

The system manager can also define logical names, and then override them by using the /SOURCE and /DESTINATION qualifiers.

PCSI$SOURCE defines the location of the software kits to install. By default, the user's default device and directory are used. PCSI$DESTINATION defines the location in which to install the software.

If the system manager does not define PCSI$DESTINATION or use the /DESTINATION qualifier, the utility installs the software product in SYS$SYSDEVICE:[VMS$COMMON] and directories under it. If this is not appropriate for your product, make sure that your installation instructions describe how to specify the /DESTINATION qualifier, or how to define the PCSI$DESTINATION logical name.

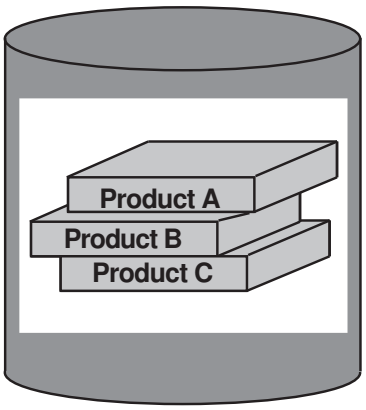
For the package operation, the logical names PCSI$SOURCE and PCSI$DESTINATION are not used. You must use the /SOURCE and /DESTINATION qualifiers. Note that certain PDL statements define the logical names PCSI$SOURCE, PCSI$DESTINATION, and PCSI$SCRATCH for use with command procedures executing in the context of a subprocess.

1.6 Packaging a Platform

In addition to packaging individual products, the POLYCENTER Software Installation utility gives you the means to assemble **integrated platforms**. An integrated platform is a combination of several products. Functionally, a platform is the same as a full kit, except that it has the designation ''PLATFORM''. A platform is intended to reference other products, but it can also supply files.

Figure 1–2 shows an example of an integrated platform.

Figure 1–2 Integrated Platform Example



ZK–5242A–GE

To package a platform, you create a **platform PDF** and **platform PTF**. In addition to other statements, the platform PDF contains *software* statements that specify the products that make up the platform. The individual products have their own PDFs and PTFs (independent of the platform PDF and PTF). For more information about platform PDFs, see Section 2.6.

1.7 Managed Objects

Managed objects exist to support the proper functioning of your product. Files, directories, and accounts are all types of managed objects. An executable image is an example of a file managed object.

Using PDL statements, you can specify the names and properties of the managed objects that are necessary for your product. At installation time, the utility uses your product description file (PDF) to create the managed objects for your product and records information about these objects in the product database. During removal of a product, the utility uses the data in the product database to delete managed objects from the system.

Most PDL statements create corresponding managed objects. For example, you use the *file* statement to specify a file managed object. In contrast, **utility directives** are PDL statements that do not specify managed objects. Utility directives affect the operation of the utility but do not affect the execution environment. For example, the *error* statement is a utility directive; it displays a text module to the user and does not affect the execution environment.

1.7.1 Managed Object Conflict

Occasionally, your product will supply a managed object that conflicts with another managed object (for example, two products supplying files with identical names that are placed in the same directory). When the utility detects conflict, it displays an informational message.

The following statements detect managed object conflict and display informational messages:

- *account*

- *bootstrap block*
- *directory*
- *file*
- *link*
- *loadable image*
- *module*
- *network object*
- *patch image*
- *patch text*
- *register module*
- *rights identifier*
- *software*

In some cases, the POLYCENTER Software Installation utility allows you to anticipate and resolve conflict before it occurs. For example, the generation option to the *file* statement lets you resolve managed object conflict. During installation, if the utility attempts to create a file that already exists, it compares the generation numbers of the files (if present), preserving the file with the highest generation number. The following statements provide some level of conflict resolution:

- *directory*
- *file*
- *module*
- *register module*

The description of these statements in Part II indicates how each one resolves managed object conflict.

1.7.2 Managed Object Replacement and Merging

As described in Section 1.7.1, managed objects occasionally have characteristics that conflict with each other. The POLYCENTER Software Installation utility handles this situation differently depending on the kit type.

For full, operating system and platform kit types, the utility deletes the existing object and replaces it with the object and characteristics provided by the new version of the product. For partial, patch, and mandatory update kit types, the utility preserves the characteristics of existing objects. For example, the security environment you establish for your product is preserved when you install a partial, patch, or mandatory update kit.

If you want to provide new characteristics for a managed object in a partial, patch, or mandatory update kit, use the *remove* statement to delete the existing object and then respecify it with the desired characteristics.

For more information about kit types, see Table 1–2.

1.7.3 Managed Object Scope and Lifetime

The **scope** of a managed object defines the degree of sharing that the managed object permits. For example, some objects are available only to certain processes, and some can be shared by all processes. The utility usually ensures that managed objects have the correct scope.

Occasionally, an advanced PDF writer might need to use the *scope* statement to give a managed object a scope other than its default. For more information about specifying the scope of a managed object, see Appendix B.

2

Creating the Product Description File

The product description file (PDF) is a required component of any software product kit that you create using the POLYCENTER Software Installation utility. The PDF

- Specifies all files that comprise the product.
- Identifies configuration options that are presented to the user at installation time.
- Specifies any dependencies the product may have on other software products.
- Defines various actions that must be performed during installation.

Developing a PDF is analogous to the kitting process as a whole. The following sequence summarizes the multistep process:

1. Locate all product material that will be included in the software product kit. These input files can remain in the directories generated by the software engineering team, you can copy them to one or more staging directories, or you can copy them to a directory tree (in reference format) as they would appear after installation.
2. Determine the required characteristics of the execution environment for your product or platform. For example, you must determine where files will be placed, if DCL tables or help libraries need to be updated, if system or process parameters need to be checked, and if you need to provide any command procedures to perform product specific tasks.
3. From product description language (PDL) statements listed in Part II, write the PDF using a text editor.
4. Create a product text file (PTF) using a text editor. You must provide a PTF if your PDF contains any statements that reference text modules to display product identification, configuration choices, informational text, or error message text. Creating a PTF is described in Chapter 3.
5. Use the DCL command PRODUCT PACKAGE to create a test kit. This command will determine if the PDF and PTF are syntactically correct and verify that all listed product material files can be found.
6. Once a kit has been successfully produced, use the PRODUCT INSTALL, PRODUCT SHOW, and PRODUCT REMOVE commands to verify the installation and removal of the product. Check for correct file placement and protection, review message text, modify configuration options, verify that execution environment requirements are satisfied, and so forth.

2.1 General Guidelines

The POLYCENTER Software Installation utility is intended to simplify the job of system managers, making products quick and easy to install and manage. Use the following guidelines when writing PDFs:

- Minimize installation activity (such as linking images and building databases). Instead, include all material required for product execution on the reference copy.
- Make your products adapt to the target environment at execution time rather than installation time. This practice keeps products consistent across varying configurations.
- Avoid requiring system parameter settings on the target system that would require rebooting the system.
- Minimize configuration choices at installation time.
- Ensure that the PDF expresses all the known requirements that your product needs to execute. Use the checklist in Section 2.2 to define the requirements for the target environment.

2.2 Defining Your Environment

To define the environment for your product, use the following checklist. (Part II of this guide describes each PDL statement.)

☐ **Does your product depend on other software?**

For example, your product may require a specific version of the operating system or optional software products. To express these software requirements, use the *software* statement or function. Note that software you reference with a *software* statement must be registered in the product database to be recognized by the POLYCENTER Software Installation utility. If you install a product using a mechanism other than the POLYCENTER Software Installation utility, the product database does not store information about the product unless you register a full or **transition product description**. For more information about creating transition product descriptions, see Section 2.7.

☐ **If you are creating a platform, what software products comprise the platform?**

If you are creating a platform, you must specify the software products that make up the platform. To specify the products that comprise your platform, use the *software* statement with the component option.

☐ **Does your product require specific hardware devices?**

For example, your product may require that the system have access to certain peripheral devices, such as a compact disc drive or printer. To display a message to users expressing these hardware requirements, use the *hardware device* statement.

☐ **Does your product run only on specific computer models?**

Some products run only on certain computer models. For example, recent versions of the OpenVMS operating system are no longer supported on the VAX–11/725 computer. If this is the case with your product, use the *hardware processor* statement to display a message to users.

☐ **Does your product require specific images, files, or directories?**

All the files, images, and directories that your product requires should be expressed in *file* or *directory* statements.

☐ **Does your product require a special account on the system?**

Some products require a dedicated account on the system. If this is the case for your product, use the *account* statement to supply the account.

☐ **Does your product require network objects?**

Some products require network objects on the system. If this is the case for your product, use the *network object* statement to supply the required network objects.

☐ **Do you want to set up rights identifiers?**

If you want to set up rights identifiers for your product, use the *rights identifier* statement.

☐ **Does your product supply an image to the system loadable images table?**

To supply an image to the system loadable images table, use the *loadable image* statement.

☐ **Does your product have several options that the user can choose?**

Although it is a good practice to limit the number of options that the user can choose, you may need to present the user with options during installation. To present options to the user, use the *option* statement.

☐ **Do you need to patch an executable image?**

To patch an executable image, use the *patch image* statement (VAX only).

☐ **Do you need to patch a text file?**

To patch a text file, use the *patch text* statement.

☐ **Does your product have specific security requirements?**

If the files and directories for your product require special protection or access controls, you can express this in the product description. See the descriptions of the *directory* statement and the *file* statement. You can also supply a rights identifier using the *rights identifier* statement.

☐ **Does your product require certain values for system parameters?**

Many software products require that system parameters have certain values for the product to function properly. Use the *system parameter* statement to display system parameter requirements to users.

☐ **Does your product require certain values for process parameters?**

If your product requires certain values for process parameters, use the *process parameter* statement to display these requirements to users.

☐ **Does your product require certain values for process privileges?**

If your product requires certain process privileges, use the *process privilege* statement to display these requirements to users.

☐ **Do you want to include a functional test with your product?**

If you have a functional test for your product, you can include it in the product material to verify that your product installed correctly. To execute the functional test for your product, use the *execute test* statement.

☐ **Are there commands that your installation procedure needs to execute that are outside the domain of the POLYCENTER Software Installation utility?**

If you have commands that your installation procedure needs to execute that have no POLYCENTER Software Installation utility equivalent, use the *execute* statement.

☐ **Does your product have specific pre- or postinstallation tasks?**

You can use the POLYCENTER Software Installation utility to automate these tasks; however, there may be some tasks you want users to perform that are outside the capabilities of the utility. You can inform users of such tasks using the *information* statement. You can also use several of the *execute* statements to perform these tasks.

☐ **Does your product require command, help, macro, object, or library modules?**

You should express the following types of modules in your PDF:

- DIGITAL Command Language (DCL) command definition modules
- DCL help modules
- Macro modules

- Object modules
- Text modules

You can express these types of modules using the *module* statement.

☐ **What happens to existing product files?**

You should make sure that your product's files are handled correctly during an installation or upgrade. The POLYCENTER Software Installation utility deletes obsolete files that are replaced when you install a full, operating system, or platform kit. In partial, patch, and mandatory update kits, the existing files are preserved. To remove obsolete files, use the *remove* statement and *file* statement options.

☐ **Does your product require documentation?**

You may want to include online documentation (such as release notes) with your product. To express the documentation requirements for your product, use the release notes option to the *file* statement.

2.3 PDF File Naming Conventions

You supply the PDF as input to the PRODUCT PACKAGE command. The PDF can have any valid OpenVMS file name and file type. For example:

```
TEST.PDF
BLACKJACK-V2.TEXT
ABC_CO-AXPVMS-BLACKJACK-V0201-17-1.DESCRIPTION
```

The execution of the PRODUCT PACKAGE command creates an output PDF. Its format is the same as the input PDF, namely a sequential file containing PDL statements. The contents of the output PDF, however, may differ slightly from that of the input PDF. For example, the POLYCENTER Software Installation utility adds the size option to every *file* statement and supplies the actual size of the file in disk blocks.

The name of the output PDF consists of the product's stylized file name and a file type of .PCSI$DESCRIPTION:

```
producer-base-product-version-kit_type.PCSI$DESCRIPTION.
```

For example:

```
ABC_CO-AXPVMS-BLACKJACK-V0201-17-1.PCSI$DESCRIPTION
```

See Section 1.5.4 for a description of the product naming syntax.

2.4 Structure of a PDF

A PDF is a text file that contains a sequence of PDL statements. A PDF must begin with a *product* statement and end with an *end-product* statement. The *product* statement uniquely identifies the product and specifies the type of kit to build (full, partial, patch, and so forth). Each file that is part of the product material must be specified with a *file* statement. The following example shows a complete PDF for a product that places one file named test.exe in SYS$COMMON:[SYSEXE].

```
product dec vaxvms test v1.0 full ;
    file [sysexe]test.exe ;
end product ;
```

2.4.1 Overview of PDL Statements

The product description language consists of 43 statements that are defined in Part II of this manual. As an overview, these statements are listed below in classes according to their main function.

- Statement groups are defined by a pair of opening and closing statements; by convention the closing statement is the keyword *end* followed by the keyword of the opening statement. Statement groups operate on statements lexically contained within their begin-end pair. Many statement groups can be nested within other groups.

  The following statement groups are used to conditionally process other statements:

  - *if* and *end if* (*else* and *else if* statements optionally can be used within the statement group)
  - *option* and *end option*

  The following statement groups unconditionally process all statements at their inner level:

  - *part* and *end part*
  - *product* and *end product*
  - *remove* and *end remove*
  - *scope* and *end scope*

- Statements that create, delete, or modify managed objects include:
  - *account*
  - *directory*
  - *file*
  - *link* (create an alias directory entry)
  - *loadable image*
  - *module*
  - *network object*
  - *patch image*
  - *patch text*
  - *rights identifier*

- Statements that enforce software dependencies and hardware requirements by testing the execution environment and taking appropriate action include:
  - *hardware device*
  - *hardware processor*
  - *software*
  - *upgrade*

- Statements whose main purpose is to display a message to the user and in some cases query the user for a response are as follows:
  - *error*
  - *information*
  - *process parameter*
  - *process privilege*
  - *system parameter*
- Statements that cause producer-supplied command procedures to execute or instruct the user to manually perform a task include:
  - *execute install...remove*
  - *execute login*
  - *execute postinstall*
  - *execute release*
  - *execute start...stop*
  - *execute test*
- Miscellaneous statements include:
  - *apply to*
  - *bootstrap block*
  - *infer*
  - *register module*

Many software products require only the use of a small subset of these PDL statements to create their PDF. Commonly used statements are as follows:

- *product* and *end product*
- *file*
- *directory*
- *module*
- *software*
- *option* and *end option*
- *if* and *end if*
- *execute install...remove*
- *execute test*

2.4.2 PDL Statement Syntax

A product description language (PDL) statement consists of:

- A keyword phrase that identifies the statement (required).
- Zero or more parameter values (which may be expressions in certain contexts).
- Zero or more options each specified as a keyword phrase and value pair.

- A semicolon (;) that terminates the statement (required).

Additional Syntax Rules

- Statements can span multiple lines and whitespace can be used freely to improve readability or show relationship through indentation levels.
- Case is not significant, except within a quoted string.
- A keyword phrase consists of one or more keywords as defined by the PDL statement.
- A comment is a sequence of two consecutive hyphens (–) followed by characters up to and including end-of-line.

  When a string containing consecutive hyphens is passed as a parameter or option value, enclose the string in quotes. For example, ''`a--b.dat`''. This prevents the hyphens from being parsed as the start of a comment.
- Lexical element separators are used to set off keywords, values, expressions, and so on. They include end-of-line, comment, and the following characters (except when they appear within a quoted string): space, horizontal tab, form feed, and vertical tab.
- Delimiters are required syntax in many situations. They consist of the following characters: semicolon (;), comma (,), left parenthesis (( ), right parenthesis ( )), left angle bracket (<), and right angle bracket (>).

  When a string contains a delimiter character that is passed as a parameter or option value, enclose the string in quotes. For example, to pass the numeric UIC string [1,1] as an option value, use the quoted string form of ''[1,1]'' because it contains a comma character.

2.4.3 PDL Function Syntax and Expressions

Certain PDL statements have a function form that tests for a condition in the execution environment and returns a Boolean value of true or false. A function is syntactically similar to its corresponding statement except that a function is enclosed in left and right angle brackets (<...>) instead of being terminated by a semicolon (;).

The following statements have corresponding functions:

- *hardware device*
- *hardware processor*
- *option*
- *software*
- *upgrade*

Expressions are used in *if* statements to produce a Boolean value for the if-condition test. An expression is delimited by opening and closing parentheses ((...)). It contains one or more functions and, optionally, one or more of the keywords AND, OR, and NOT, which are used as logical operators.

An expression has one of the following forms, where each term is either another expression or a function:

- (term)
- (term AND term)
- (term OR term)

- (NOT term)

The following example shows an *if* statement using a compound expression:

```
if ( (not <hardware device MUA0:>) and
     (<software ABC VAXVMS TEST version below 2.0>) ) ;
    .
    .
    .
end if ;
```

2.4.4 PDL Data Types and Values

The PDL has several base data types that you must use when passing parameters to the PDL statements listed in Part II. Table 2–1 describes the PDL base data types and their values. PDL statements may restrict the range of values that can be used as parameters.

Table 2–1 Base Data Types and Values

Data Type	Values
Boolean	The number 0 (false), the number 1 (true), the keywords **false**, **true**, **no**, and **yes**.
String	A sequence of 0 to 255 ISO Latin-1 characters. In the context of PDF language statements, • *abc* is an unquoted string. • *''abc''* is a quoted string. • *''''double_quoted_string''''* is a quoted string that maintains original quotation marks. You must use the quoted string form if the string contains any PDL delimiters (open/close parenthesis, comma, open/close angle brackets, and semicolons) or lexical element separators (double hyphen, space, horizontal tab, form feed, or vertical tab). For example, ''/privilege=(tmpmbx, netmbx)''. Table 2–2 lists the additional constraints on PDL strings.
Signed integer	Specifies a positive, negative, or zero integral value in the range of -2147483648 to 2147483647.
Unsigned integer	Specifies a zero or positive integral value in the range of 0 through 4294967295.
Version identifier	See the description in Section 1.5.4.
Text module name	Specifies a unique name for a text module using the printable ISO Latin-1 characters, excluding horizontal tab, space, exclamation point, and comma. The name can be from 1 to 31 characters in length.

Table 2–2 describes additional constraints on the string data type.

Table 2–2 String Data Type Constraints

String Type	Values	Examples
Unconstrained	None; any character may appear in any position.	
Access control entry (ACE)	Specifies an ACE for a directory or file.	''(IDENTIFIER=[KM],ACCESS=READ)''
Command	Specifies an operating system command that you want to execute during a specific operation.	@SYS$TEST:PROD$IVP.COM
Device name	Specifies the name of a hardware device.	DUB6:
File name	Specifies a file name (without a device or directory specification).	STARTUP.DAT
Identifier name	Specifies a rights identifier.	DOC
Module name	Specifies the name of a module in a library.	FMSHELP
Processor model name	Specifies the model identification of a particular computer system.	7
Relative directory specification	Specifies the directory name and, if necessary, the directory path, relative to the root directory specification.	[MY_PRODUCT]
Relative file specification	Specifies the directory path and file name, relative to the root directory path.	[MY_PRODUCT]DRIVER.DAT
Root directory specification	Specifies the directory name and a trailing period (.). If you specify a directory name and omit the period, it is inserted. If necessary, you can add the device name.	[TEST.] SYS$SYSDEVICE:[VMS$COMMON.]

2.5 Writing a PDF for a Full Kit

Example 2–1 shows a PDF for Checktran. You can see how the requirements of the execution environment were translated into statements in the PDF.

Example 2–1 Checktran Product Description File

(continued on next page)

Example 2–1 (Cont.) Checktran Product Description File

```
$ TYPE CHECKTRAN-TEST.PDF

product DEC VAXVMS CHECKTRAN V4.3 full ; ❶
    software DEC VAXVMS VMS version minimum V5.0 ; ❷
    file [SYSEXE]CHECKTRAN.EXE ; ❸
    module [SYSUPD]CHECKTRAN.CLD type command module CHECK ; ❹
    module [SYSUPD]CHECKTRAN.HLP type help module HELP ; ❺
    file [SYSTEST]CHECKTRAN$IVP.COM ; ❻
    file [SYSEXE]CHECKTRAN.DAT ;
    execute test @SYS$TEST:CHECKTRAN$IVP ; ❼
end product ; ❽
```

The following list describes the function of the PDL statements in Example 2–1:

❶ The *product* statement identifies the product as DEC VAXVMS CHECKTRAN Version 4.3. The full option specifies that the kit is a complete software distribution and not an update or patch.

❷ The *software* statement specifies that to run Checktran, the execution environment must be running at least VMS Version 5.0.

❸ The first *file* statement in the PDF supplies the file [SYSEXE]CHECKTRAN.EXE.

❹ The first *module* statement in the PDF installs the command module for Checktran in the default command library [SYSLIB]DCLTABLES.EXE.

❺ The second *module* statement in the PDF installs the help module for Checktran in the default help library [SYSHLP]HELPLIB.HLB.

❻ The next statement installs the DCL command procedure that will be used for the functional test.

❼ The *execute test* statement executes the functional test for the product.

❽ The *end product* statement closes the group of product statements.

In this example, the PDF for Checktran is relatively brief. Example 2–2 shows a PDF for DEC TCP/IP Services for OpenVMS, which is more complex.

Example 2–2 UCX Product Description File

```
$ TYPE UCX.DES

-- "DEC TCP/IP Services for VMS" ❶

product DEC VAXVMS UCX V2.0 full ; ❷

software DEC VAXVMS VMS version minimum V5.4 ; ❸

process parameter ASTLM minimum 24 ; ❹
process parameter BIOLM minimum 18 ;
process parameter BYTLM minimum 32768 ;
process parameter DIOLM minimum 18 ;
process parameter ENQLM minimum 200 ;
process parameter FILLM minimum 100 ;

rights identifier UCX$NFS_REMOTE ; ❺

execute start "@SYS$STARTUP:UCX$STARTUP.COM" ❻
        stop "@SYS$STARTUP:UCX$SHUTDOWN.COM" ;
```

(continued on next page)

Example 2–2 (Cont.) UCX Product Description File

```
execute test "@SYS$TEST:UCX$IVP.COM" ; ❼

information PRE_INSTALL confirm ; ❽

information POST_INSTALL phase after ; ❾

directory [SYSTEST.UCX] ; ❿
directory [SYSHLP.EXAMPLES.UCX] ;

module [000000]UCX$TOP_LEVEL_HELP.HLP type help module UCX ; ⓫
module [000000]UCX.CLD type command module UCX ; ⓬

file [SYSEXE]UCX$SERVICE.DAT ; ⓭
file [SYSEXE]UCX$FTPSERVER.COM ;
file [SYSEXE]UCX$SNMP_AGENT.EXE ;
file [SYSEXE]UCX$UCP.EXE ;
   .
   .
   .
file [SYSTEST]UCX$IVP.COM ;
file [SYSTEST.UCX]UCX$INET_IVP.EXE ;
file [SYSUPD]UCX$CLEANUP.COM ;

option EXAMPLES ; ⓮

  file [SYSHLP.EXAMPLES.UCX]UCX$IOCTL_ROUTINE.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$TCP_CLIENT_IPC.C ;
   .
   .
   .
  file [SYSHLP.EXAMPLES.UCX]UCX_TRACE.EXE ;
  file [SYSHLP.EXAMPLES.UCX]TRACEROUTE.EXE ;

end option ;

option NFS ; ⓯

  file [SYSEXE]UCX$CONVERT.FDL ;
  file [SYSEXE]UCX$CONVERT.COM ;
  file [SYSEXE]UCX$SERVER_NFS.EXE ;

  file [SYSHLP]UCX$VMS_FILES.DOC ;

  file [SYSLIB]UCX$CFS_SHR.EXE ;

end option ;

option APPLICATIONS ; ⓰

  module [000000]FTP.CLD type command module FTP ;
  module [000000]RLOGIN.CLD type command module RLOGIN ;
  module [000000]RSH.CLD type command module (RSH,RSHELL) ;
  module [000000]TELNET.CLD type command module TELNET ;
  module [000000]UCX$LPQ_CLD.CLD type command module LPQ ;
  module [000000]UCX$LPRM_CLD.CLD type command module LPRM ;

  file [SYSEXE]UCX$ENCODE.COM ;
  file [SYSEXE]UCX$DECODE.COM ;
   .
   .
   .
  file [SYSEXE]UCX$LPD_RCV.EXE ;
  file [SYSEXE]UCX$LPRSETUP.EXE ;

  file [SYSHLP]UCX$FTP_HELP.HLB ;
  file [SYSHLP]UCX$TELNET_HELP.HLB ;
```

(continued on next page)

Example 2–2 (Cont.) UCX Product Description File

```
  file [SYSLIB]UCX$SMTP_MAILSHR.EXE ;
  file [SYSLIB]UCX$LPD_SHR.EXE ;

end option ;

end product ; ⓱
```

The following list describes the function of the PDL statements in the UCX example:

❶ The first line of the PDF is a comment line (identified by two hyphens).

❷ The *product* statement identifies the product as DEC VAXVMS UCX V2.0. The full option specifies that the kit is a complete software distribution and not an update or patch.

❸ The *software* statement specifies that to run UCX, the execution environment must be running at least VMS Version 5.4.

❹ The *process parameter* statements specify the process parameters that the product requires. The utility will display a message about these requirements to users when they install UCX.

❺ The *rights identifier* statement causes the utility to create the specified rights identifier.

❻ The first *execute* statement specifies command procedures for product startup and shutdown.

❼ The second *execute* statement specifies the functional test for the product.

❽ The first *information* statement displays a message about preinstallation tasks. The utility prompts the user for confirmation of these tasks.

❾ The second *information* statement displays a message about postinstallation tasks.

❿ The *directory* statements create the directories [SYSTEST.UCX] and [SYSHLP.EXAMPLES.UCX].

⓫ The first *module* statement installs the help module for UCX in the default help library [SYSHLP]HELPLIB.HLB.

⓬ The second *module* statement installs the command module for UCX in the default command library [SYSLIB]DCLTABLES.EXE.

⓭ The next *file* statements install several required files for UCX.

⓮ The EXAMPLES option group contains programming example files that the user can select.

⓯ The NFS option group contains NFS protocol support files that the user can select.

⓰ The APPLICATIONS option group contains optional application support files and modules that the user can select.

⓱ The *end product* statement closes the group of *product* statements.

2.6 Writing a PDF for a Platform Kit

As mentioned in Section 1.6, the POLYCENTER Software Installation utility gives you the means to create a software kit that is a combination of products (an integrated platform). To do this, you must create a platform PDF.

A platform PDF is distinguished from other types of PDFs by using the **platform** keyword in the *product* statement. In a platform PDF, you can use any statements to express the execution environment for the platform. For example, you can use the component option of the *software* statement to specify the software products that make up the platform.

When you package a platform, the PDFs and PTFs of the referenced products do not need to be present. However, when you copy a platform, products that are referenced with the component option of the *software* statement must be present.

Example 2–3 shows a product description file for the office_plat product.

Example 2–3 Product Description File

```
product DEC VAXVMS OFFICE_PLAT V1.0 platform ;
    file [SYSEXE]SOFTWARE.TST ;
    software DEC VAXVMS VMS
             version minimum V5.5 ;
    software DEC VAXVMS PROD_B
             version required 4.3 component ;
    software DEC VAXVMS PROD_C component ;
end product ;
```

The PDF in this example is for a platform named OFFICE_PLAT. The software products PROD_C and PROD_B are components of the platform. VMS Version 5.5 or greater must be present along with Version 4.3 of PROD_B.

2.7 Writing a PDF for a Transition Kit

If you install a product using a mechanism other than the POLYCENTER Software Installation utility, the product database does not store information about the product. A **transition product description file** allows you to register a product in the product database even if it was installed using a mechanism other than the POLYCENTER Software Installation utility.

Once users register a product using the POLYCENTER Software Installation utility, they can perform utility operations on that product (for example, reconfigure or remove operations). Registering a product also allows you to use *software* statements or functions to reference that product.

A transition PDF must use the **transition** keyword in the *product* statement. A transition PDF can reference managed objects such as files, directories, modules, and so forth. However, none of these objects is created or modified when the product is registered using the DCL command PRODUCT REGISTER PRODUCT. In addition, files specified in *file* statements do not need to be present when a transition kit is packaged because product material is not included in this type of kit.

In a transition PDF, the *infer* statement is useful for testing the target system to determine if a product or product version is available.

An alternative to providing a transition kit and having the user register it with a PRODUCT REGISTER PRODUCT command is to instruct the user to register the product by using the SYS$UPDATE:PCSI$REGISTER_PRODUCT.COM procedure. This procedure prompts the user to enter product name, version, and producer information. Registering a product with minimal information is sufficient to reference the product with a *software* statement from another kit.

Example 2–4 shows a transition PDF for the FMS product:

Example 2–4 FMS Product Description File

```
product DEC AXPVMS FMS V2.4 transition ; ❶
    infer version from [SYSLIB]FDVSHR.EXE ; ❷
    file [SYSLIB]FDVSHARE.OPT ; ❸
    module [SYSUPD]FDV.OBJ type object module FDV ; ❹
    module [SYSUPD]FDVMSG.OBJ type object module FDVMSG ;
    module [SYSUPD]FDVDAT.OBJ type object module FDVDAT ;
    module [SYSUPD]FDVERR.OBJ type object module FDVERR ;
    module [SYSUPD]FDVTIO.OBJ type object module FDVTIO ;
    module [SYSUPD]FDVXFR.OBJ type object module FDVXFR ;
    module [SYSUPD]HLL.OBJ type object module HLL ;
    module [SYSUPD]HLLDFN.OBJ type object module HLLDFN ;
end product ;
```

The following list describes the statements in this example:

❶ The **transition** keyword to the *product* statement indicates that this is a transition PDF.

❷ The *infer version* statement tests the execution environment to determine whether the file FDVSHR.EXE is present. If it is, the utility infers the version that is installed.

❸ The *file* statement indicates that the [SYSLIB]FDVSHARE.OPT file is part of the FMS kit.

❹ The *module* statements describe object modules in the default object library [SYSLIB]STARLET.OLB that are part of the FMS kit.

3
Creating the Product Text File

The product text file (PTF) is an optional component of a software product kit. However, most kits created using the POLYCENTER Software Installation utility include a PTF. You must supply a PTF to the kitting process if you want to use PDF statements that display text to users during product installation. The following PDF statements have corresponding text modules in the PTF:

- *error*
- *information*
- *option*
- *part*
- *product*

For each text module in the PTF you can provide both brief and detailed descriptions. By default, the brief version of the text module is displayed. The user requests detailed explanations by including the /HELP qualifier on the PRODUCT INSTALL command line. If you choose to provide only brief text for a given text module and the user asks for detailed text, the brief version is displayed. By providing detailed descriptions and other informational help text, you should be able to reduce or eliminate hardcopy installation documentation.

3.1 PTF File Naming Conventions

The PTF provided as input to the PRODUCT PACKAGE command must reside in the same directory as the PDF. Furthermore, it must have the same file name as the PDF and a file type of .PCSI$TEXT.

The following are examples of valid input PDF and PTF names:

```
TEST.PDF
TEST.PCSI$TEXT

BLACKJACK-V2.TEXT
BLACKJACK-V2.PCSI$TEXT

ABC_CO-AXPVMS-BLACKJACK-V0201-17-1.DESCRIPTION
ABC_CO-AXPVMS-BLACKJACK-V0201-17-1.PCSI$TEXT
```

The execution of the PRODUCT PACKAGE command transforms the input PTF into an output PTF. The input PTF is a sequential file containing header lines and text module lines. The output PTF is an OpenVMS library file. Its name consists of the product's stylized file name and a file type of .PCSI$TLB:

```
producer-base-product-version-kit_type.PCSI$TLB
```

For example:

```
ABC_CO-AXPVMS-BLACKJACK-V0201-17-1.PCSI$TLB
```

3.2 Structure of a PTF

A PTF is a text file that contains packaging directives, module header lines, and module text. The PTF must begin with the =product directive line that uniquely identifies the product and specifies the type of kit. The rest of the file contains one or more text modules. Each text module entry consists of:

- A module header line that identifies the name of the text module.
- An =prompt directive line that includes text for a brief display.
- Zero or more lines of text that are combined with the brief text to form the detailed display associated with the text module.

The user chooses whether to receive brief or detailed explanations by use of the /HELP qualifier on the PRODUCT INSTALL command.

Brief text format (the default) is restricted to one line of text; that is, the text in the =prompt directive line. Detailed or help text can include any number of lines of text. The formatting of the information is preserved on output, except that the POLYCENTER Software Installation utility may indent the entire block of text displaying information about configuration options or software requirements.

Comment lines are not permitted in a PTF.

3.2.1 Specifying the Product Name

You must use the =product directive to specify product information in the PTF. The information that you specify with the =product directive must match the information you specify with the *product* statement in the PDF.

The =product directive has the following format:

=product *producer base product version kit_type*

See Section 1.5.4 for the naming conventions.

3.2.2 PTF Modules

PTF text modules are text blocks that you want to present to the user. The POLYCENTER Software Installation utility does not process text blocks sequentially, so the order of the text modules in the PTF does not matter.

Text modules are identified by a module header line in the following format:

1 module-name

The module header line consists of the number 1, followed by a space or tab and the name of the module. The *module-name* must be from 1 to 31 ISO Latin-1 characters, excluding the horizontal tab, space, exclamation point (!), and comma (,) characters. For example:

```
1 SAMPLE
```

The POLYCENTER Software Installation utility uses the name of the module to associate the text module with a line from the PDF. For example, the SAMPLE module could correspond to an option in the PDF:

```
option SAMPLE ;
```

When a user chooses the SAMPLE option, the utility displays the help text for the choice. Note that the format of the help text in the PTF (for example, spacing and blank lines) is preserved when the utility displays it to the user.

The utility also allows you to specify text modules that are not associated with statements in the PDF. These text modules are preceded by an apostrophe ('). Use the following module names to specify information about your product:

- The 'LICENSE module specifies licensing information.
- The 'NOTICE module specifies copyright, ownership, and similar legal information.
- The 'PRODUCER module specifies a brief description of the producer of the product.
- The 'PRODUCT module specifies a brief functional description of the product.

For example, a product might contain the following modules:

```
=product DEC VAXVMS C V1.0 full
1 'PRODUCT
=prompt DEC C++ for OpenVMS
DEC C++ for OpenVMS VAX is a native compiler that implements the C++
programming language and includes:

o  A C++ compiler that implements C++ as defined by The Annotated C++
   Reference Manual, Ellis & Stroustrup, reprinted with corrections,
   May 1991. The compiler implementation includes templates but ex-
   cludes exception handling. DEC C++ generates optimized object code
   without employing an intermediate translation to C.

o  The DEC C++ Class Library, which consists of the following class li-
   brary packages: iostream, complex, generic, Objection, Stopwatch,
   String, task, messages, and vector.
1 'NOTICE
=prompt © Digital Equipment Corporation 1992. All rights reserved.
Unpublished rights reserved under the copyright laws of the United States.

This software is proprietary to and embodies the confidential technology of
Digital Equipment Corporation. Possession, use, or copying of this software
and media is authorized only pursuant to a valid written license from Digital
or an authorized sublicensor.

Restricted Rights:  Use, duplication, or  disclosure by the U.S.
Government is subject to restrictions as set forth in subparagraph (c)(1)(ii)
of DFARS 252.227-7013, or in FAR 52.227-19, or in FAR 52.227-14 Alt. III,
as applicable.
1 'LICENSE
=prompt This product uses the PAKs: <xxx> and <xxx-RT>.
This software is furnished under the licensing provisions of Digital
  Equipment Corporation's Standard Terms and Conditions. For more in-
  formation about Digital's licensing terms and policies, contact your
  local Digital office.
1 'PRODUCER
=prompt Digital Equipment Corporation
This software product is sold by Digital Equipment Corporation.
```

To see how the POLYCENTER Software Installation utility displays this text, see the *POLYCENTER Software Installation Utility User's Guide*.

3.2.3 Including Prompt and Help Text

You can include prompt and help text in your PTF using the =prompt directive. Prompt text cannot exceed one line of text. Help text is similar to prompt text, except that it can span multiple lines. The help text follows the =prompt line. You can also include blank lines in help text.

The following example shows prompt text:

```
=prompt This option provides files for programming support.
```

The following example shows a sample product text file. Note the prompt and help text:

```
=product DEC VAXVMS UCX V2.0 full
1 'PRODUCT
=prompt DEC TCP/IP Services for OpenVMS
 DEC TCP/IP Services for OpenVMS is an OpenVMS layered software product that
 promotes interoperability and resource sharing between OpenVMS systems,
 UNIX systems, and other systems that support the TCP/IP and NFS
 protocol suites.

 The product provides capabilities for file access, remote terminal
 access, remote command execution, remote printing, mail, and application
 development, including three major functional components:

 o  The Run-Time component, which is based on the Berkeley Standard
    Distribution, brings TCP/IP communications to OpenVMS computer systems.
    It also includes a suite of application development tools
    (DECrpc, C socket programming interface, and QIO programming
    interface).

 o  The Applications component includes the popular user-oriented protocols
    for file transfer, remote processing, remote printing, and mail: File
    Transfer Protocol (FTP), Telnet Protocol (Telnet), Berkeley R commands
    (rsh, rlogin, rexec), remote printing, and Simple Mail Transfer
    Protocol (SMTP).

 o  The DEC NFS component supports Network File System (NFS) V2.0 proto-
    col specifications. NFS is an Application layer protocol that provides
    clients with transparent access to remote file services.

1 'NOTICE
=prompt © Digital Equipment Corporation 1992. All rights reserved.
Unpublished rights reserved under the copyright laws of
the United States.

This software is proprietary to and embodies the confidential technology of
Digital Equipment Corporation. Possession, use, or copying of this software
and media is authorized only pursuant to a valid written license from Digital
or an authorized sublicensor.

Restricted Rights:   Use, duplication, or  disclosure by the U.S.
Government is subject to restrictions as set forth in subparagraph (c)(1)(ii)
of DFARS 252.227-7013, or in FAR 52.227-19 or in FAR 52.227-14 Alt. III, as
applicable.
1 'LICENSE
=prompt This product uses the PAKs: UCX and UCX-IP-RT.
This product currently has two Product Authorization Keys (PAKs):

   Producer  PAK Name  Version  Release Date

     DEC     UCX         2.0     6-JUL-1992
     DEC     UCX-IP-RT   2.0     6-JUL-1992

1 'PRODUCER
=prompt Digital Equipment Corporation
This software product is sold by Digital Equipment Corporation.
1 EXAMPLES
=prompt Example files
The example files include client/server programming examples.
1 NFS
=prompt NFS files

The DEC NFS component supports Network File System (NFS) protocol
specifications. NFS is an Application layer protocol that provides
clients with transparent access to remote file services.
```

```
The DEC NFS server promotes data sharing among clients by providing
a central data storage facility for OpenVMS and UNIX files. The DEC NFS
server provides two types of file access for UNIX clients: 1) client
access to OpenVMS files, and 2) client access to files compatible with UNIX
systems.
1 APPLICATIONS
=prompt Applications
The Applications component includes the popular user-oriented protocols
for file transfer, remote processing, remote printing, and mail: File
Transfer Protocol (FTP), Telnet Protocol (Telnet), Berkeley R commands
(rsh, rlogin, rexec), remote printing, and Simple Mail Transfer
Protocol (SMTP).
1 PRE_INSTALL
=prompt Complete preinstallation tasks for DEC TCP/IP Services first.
Before you install DEC VMS UCX, you must complete certain preinstallation
tasks. For more information, refer to the "DEC TCP/IP Services for VMS
Installation and Configuration Guide."
1 POST_INSTALL
=prompt Postinstallation tasks required for DEC TCP/IP Services.
For more information, refer to these associated documents:

- "DEC TCP/IP Services for VMS Installation and Configuration Guide"
- "DEC TCP/IP Services for VMS System Management"
```

4
Packaging the Kit

You use the Package command to create a software product kit. This operation uses a product description file (PDF), an optional product text file (PTF), and product material files as input to produce a software product kit in either sequential or reference format.

When you package your product, it takes one of the following forms:

- **Sequential copy format**. In this form, the PDF, the PTF, and all files that comprise the product are packaged in a single container file. This container file can be placed either on a random-access device, such as a compact disc, or on a sequential access device, such as a magnetic tape. Most layered products are distributed in sequential copy format.
- **Reference copy format**. In this form the PDF, the PTF, and all files that comprise the product are placed in a directory tree on a random-access device. OpenVMS is distributed in reference copy format on CD–ROM.

Note

Although the Package commands described in this section are for use with the DCL interface, the POLYCENTER Software Installation Utility also provides a Motif interface, which is available on a workstation or an X terminal running DECwindows Motif for OpenVMS.

4.1 Packaging with the DCL Interface

You can enter the DIGITAL Command Language (DCL) command PRODUCT PACKAGE in the following format:

```
$ PRODUCT PACKAGE product-name[,...]
      /SOURCE=file-specification
      /DESTINATION=device-name[directory-name]
      /MATERIAL=path-name
```

where *product_name* specifies the name of the software you want to package. Note that you cannot specify hyphens (-) in the *product_name*. Table 4–1 describes the PRODUCT PACKAGE command in detail.

Table 4–1 Product Command Format

Qualifier	Description
/BASE_SYSTEM=base-system-name	Performs the operation only on software products that apply to the named base system. The default value is the platform (that is, the hardware and software combination) on which the POLYCENTER Software Installation utility is executed. The default is AXPVMS when you run the utility on OpenVMS Alpha and VAXVMS when you run the utility on OpenVMS VAX.
/COPY(default) /NOCOPY	Specifies whether you want the product material files and associated directories included in the product kit. The /NOCOPY qualifier can save file processing time when you are debugging a PDF and do not need to produce a complete product kit. The use of /NOCOPY with /FORMAT=SEQUENTIAL is not supported and produces undefined results.
/DESTINATION=device-name:[directory-name]	If /FORMAT=SEQUENTIAL is specified, /DESTINATION specifies the directory where the utility creates the sequential kit. A sequential kit is a container file that includes the PDF, PTF, and all of the images and other material that make up the product. The file type of the sequential kit file is .PCSI. If /FORMAT=REFERENCE is specified (or defaulted), /DESTINATION specifies the directory where the utility creates the output PDF and optional PTF. The file types of the PDF and PTF files are .PCSI$DESCRIPTION and .PCSI$TLB, respectively. The images and other materials that make up the product are placed in a directory tree under this directory. If the device-name is not provided, it defaults to the user's default device. If the directory-name is omitted, it defaults to the user's default directory. **This is a required qualifier for the PRODUCT PACKAGE command. The logical name PCSI$DESTINATION is not used.**
/FORMAT=keyword	Specifies the output format of the product kit. Keywords are: • REFERENCE — Reference format in which product files are placed in a directory tree. The default is /FORMAT=REFERENCE. • SEQUENTIAL — Sequential format in which product files are placed in *full-product-name.PCSI*, a container file.
/LOG /NOLOG (default)	Displays informational messages as the POLYCENTER Software Installation utility performs the operation.

(continued on next page)

Table 4–1 (Cont.) Product Command Format

Qualifier	Description
/MATERIAL=path-name /MATERIAL=(path-name[,...])	Specifies one or more locations in which the utility can search for product material files to include in the software product kit. Material files represent the output of the producer's software engineering process, that is, all files that make up the software product excluding the PDF and PTF. The format for path-name is: device-name:[directory-name] You can specify path-name as: • A specific directory — Only one directory is searched. • A root directory — A period (.) following the directory name denotes a root directory specification. For example, TEST$:[ABC.FT2.] limits the search path to subdirectories of [ABC.FT2]. • A wildcard directory — The directory name includes one or more of the wildcard characters; asterisk (*), percent sign (%), or ellipsis (...). All directories that satisfy the wildcard specification are searched. Note that when you use either a wildcard directory or a list of path names, if files in different directories have the same name, only the first file found in the search path is used. When either a specific directory or a wildcard directory is used, the relative file specification on the *file* statement in the PDF is not used to locate the file. However, when a root directory is used, the utility appends the relative file specification from the *file* statement in the PDF to the root directory in the material search path to locate files. This is a required qualifier for the PRODUCT PACKAGE command. Parentheses (( )) are optional only when you specify a single path name. They are required when you specify multiple path names.
/OWNER_UIC=uic	Specifies the owner user identification code (UIC) for files created during a copy operation. (This operation requires SYSPRV if the UIC is not your own.) By default, the user executing the operation owns the software product files. For example, if you are logged in to your own account, you can use this qualifier during a copy operation to assign ownership of the product files to SYSTEM rather than to your own account. Specify the UIC in alphanumeric format (in the form [name]) or in octal group-member format (in the form [g,m]). UIC formats are described in the *OpenVMS User's Manual*.
/PRODUCER=producer-name	Performs the operation only on software products that are produced by the specified manufacturer. By default, the operation is performed for all producers.

(continued on next page)

Table 4–1 (Cont.) Product Command Format

Qualifier	Description
/SOURCE=file-specification	Specifies the location of the input PDF. If the device-name is omitted, it defaults to the user's default device. If the directory-name is omitted, it defaults to the user's default directory. If the file name and file type components of the file-specification are not provided, they default to *full-product-name.PCSI$DESCRIPTION*. The optional PTF, if used, must be in the same directory and have the same file name as the PDF with a .PCSI$TEXT file type. **This is a required qualifier for the PRODUCT PACKAGE command. The logical name PCSI$SOURCE is not used.**
/VERSION=version-number	Performs the operation only on software products that have the specified version. By default, the operation is performed for all versions.

4.2 Product Command Example

```
$ PRODUCT PACKAGE VIEWER /FORMAT=SEQUENTIAL -
_$ /SOURCE=[JAMES.TEST.PDF] -
_$ /DESTINATION=DKA200:[PCSI_KITS] -
_$ /MATERIAL=BUILD$:[VIEWER0201.RESULT...]
```

This command produces the following sample display, to which you respond to prompts.

```
   1 - ABC AXPVMS VIEWER V2.1
   2 - ABC VAXVMS VIEWER V2.2
   3 - ABC VAXVMS VIEWER V2.1
   4 - All products listed above
   5 - Exit
Desired Product(s): 3

The following product has been selected:
ABC VAXVMS VIEWER V2.1

Do you want to continue [YES] y
%PCSI-I-SUCCESS, operation completed successfully
```

The command in this example creates a sequential kit for product VIEWER named DKA200:[PCSI_KITS]ABC-VAXVMS-VIEWER-0201–1.PCSI. The input PDF named [JAMES.TEST.PDF]ABC-VAXVMS-VIEWER-0201–1.PCSI$DESCRIPTION in the user's default directory and product material files from the BUILD$:[VIEWER0201.RESULT...] directory tree are used to create the kit. Since more than one PDF matching the selection criteria is present in the source directory, the user is asked to select which to use.

Part II

Product Description Language Statements

Part II contains descriptions of individual product description language statements.

account

The *account* statement uses a command procedure to create a system account.

Syntax

account *name* with *(parameters,...)* ;

Parameters

name
Specifies the user name of the account as a 1- to 12-character string. The user name is passed to the command procedure as P1.

with *(parameters,...)*
Specifies the list of parameters that are passed to the command procedure that creates the account. Each parameter must be a single unquoted or quoted string that specifies P2 through P8, in order. Refer to the Description section for the meaning of the parameters.

Description

The *account* statement uses a command procedure (SYS$UPDATE:PCSI$CREATE_ACCOUNT.COM) to create an account. The parameters that you pass to the command procedure that creates the accounts are:

- P1 specifies the user name of the account (using the **name** parameter).
- P2 specifies general AUTHORIZE qualifiers. If there are no qualifiers to pass, specify a null string ‘‘ ’’
- P3 specifies a comma-separated list of rights identifiers to grant to the user name. These identifers must already exist, or be created with a separate *rights identifier* statement.
- P4 through P8 specify other general AUTHORIZE qualifiers.

When you remove a product that created accounts, the POLYCENTER Software Installation utility uses a command procedure (SYS$UPDATE:PCSI$DELETE_ACCOUNT.COM) to delete accounts associated with your product.

Note

In a future version, the POLYCENTER Software Installation utility may create and delete these managed objects directly without the use of command procedures. If this is the case, these statements will continue to function, but the command procedures may not be maintained or shipped with future versions of the utility.

The *account* statement specifies an account managed object that has the following characteristics:

- Its name is the value of the **name** parameter. The name must be unique among all account names.
- It has operating lifetime.

- Managed object conflict is not recoverable.

See Also

rights identifier

Example

```
account test with ("/priv=(tmpmbx, netmbx)",❶
                   "PCSI_TEST",❷
                   "/account=PCSI",❸
                   "/astlm=500/biolm=200/bytlm=96000",
                   "wsdefault=4000",
                   "/flags=(nodisuser,genpwd)",
                   "/pwdminimum=8");
```

In this example, the *account* statement creates the TEST account.

❶ Parameter P2 specifies the TMPMBX and NETMBX privileges to be assigned to the TEST account.

❷ Parameter P3 is a rights identifier. This name must exist on the system prior to executing the *account* statement. It can be created with a *rights identifier* statement.

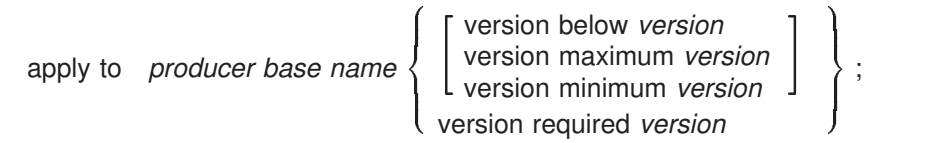
❸ Parameters P4 to P8 assign certain values to the TEST account.

apply to

The *apply to* statement specifies a product that you want to update with a mandatory update or patch product description.

You must include an *apply to* statement in a mandatory update or patch PDF to identify the product that is being patched or updated. This statement is not valid in other types of PDFs.

Syntax



Parameters

producer
Specifies the legal owner of the software product.

base
Specifies the base on which the product executes. The parameter must be a single quoted or unquoted string.

name
Specifies the product name. The combination of producer name and product name must be unique.

Options

version below *version*
Specifies the smallest invalid product version. Use this option to specify that the product version must be less than (but not equal to) the specified version. You cannot use this option with either the version minimum or version required options. By default, there is no smallest invalid version.

version maximum *version*
Specifies a maximum product version that must be available. Use this option to specify that the product version must be less than or equal to the specified version. You cannot use this option with the version below option. By default, there is no maximum version.

version minimum *version*
Specifies a minimum product version that must be available. Use this option to specify that the product version must be greater than or equal to the specified version. By default, there is no minimum version.

version required *version*
Specifies a required product version that must be available. Use this option to specify that a specific product version must be present. You cannot use this option with either the version below, version maximum, or version minimum options. By default, there is no required version.

apply to

Description

The *apply to* statement specifies which versions of another product must be available for a valid installation operation. You can use options to this statement to define below, maximum, minimum, and required versions.

If your product references another product with an *apply to* statement, the referenced product will be installed earlier than, and removed later than, your product. If two products reference each other (creating an infinite loop), the utility issues an error message.

The *apply to* statement is a utility directive and does not specify a managed object.

See Also

product
upgrade
software

Example

```
product DEC VAXVMS CSCPAT57 V1.0 patch ;
    apply to DEC VAXVMS FORTRAN version required V2.0 ;
    patch image [SYSEXE]FORTRAN.EXE with [000000]CSCPAT57.PAT ;
end product ;
```

This example shows part of the product description for a patch to DEC Fortran. As shown in the *apply to* statement, you must have DEC Fortran Version 2.0 installed to apply this patch.

bootstrap block

The *bootstrap block* statement specifies the file that the bootstrap block references.

Syntax

bootstrap block *name* image *source* ;

Parameters

name
Specifies the bootstrap file specification. You must define the file you specify in the same product description (with a *file* statement). You must also ensure that the file has bootstrap scope and product or assembly lifetime (using the *scope* statement).

image *source*
Specifies the file specification of the file that contains the bootstrap block image. The referenced file must be defined in the same product description (with a *file* statement), and it must also have product scope and product lifetime.

Description

The *bootstrap block* statement specifies the file that the bootstrap block references. You specify the name of the file as the **name** parameter.

The *bootstrap block* statement also specifies a bootstrap block managed object that has the following characteristics:

- It is unnamed and unique within the bootstrap scope.
- It has operating lifetime and bootstrap scope.
- Managed object conflict is not recoverable.

See Also

scope

Example

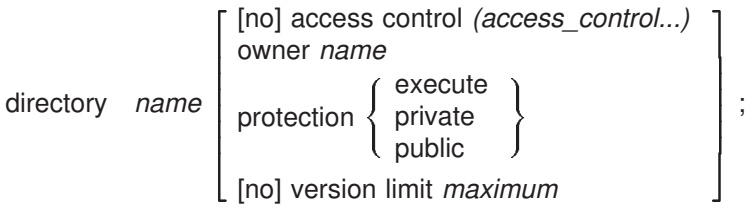
```
bootstrap block [sysexe]vmb.exe image [sysexe]bootblock.exe ;
```

This example uses the *bootstrap block* statement to point the bootstrap block to the bootstrap file ([SYSEXE]VMB.EXE).

directory

The *directory* statement creates the specified directory if it does not already exist.

Syntax



Parameter

name
Specifies the directory name.

Options

[no] access control *(access_control...)*
Specifies the minimum access control entries (ACEs) that the directory will have. You must specify the ACEs as a quoted string. By default, directories have no added ACEs.

owner *name*
Specifies the account name that owns the directory. By default, the directory is owned by the SYSTEM account. If you specify a numeric value for *name*, you must enclose the string in quotation marks; for example `"[11,7]"`.

protection execute
Sets the directory protection so that users have execute access.

protection private
Sets the directory protection so that users have no access.

protection public
Sets the directory protection so that users have read and execute access. This is the default option.

[no] version limit *maximum*
Specifies the maximum number of file versions in the directory as an unsigned integer from 1 through 32767.

The default is no version limit.

Description

The *directory* statement creates the specified directory if it does not already exist.

The *directory* statement specifies the name of a directory managed object. Check the other statements in your PDF to make sure the name you specify is unique among all directory, file, and link managed objects in all scopes.

The scope and lifetime of the directory managed object depend on whether it is lexically contained in a *scope, end scope* pair, as shown in Table PDL–1.

Table PDL–1 Directory Managed Object Scope and Lifetime

Type of Scope Group	Lifetime	Scope
Product	Product	Product
Global	Assembly	Global
Bootstrap	Operating	Bootstrap
Processor	Operating	Processor

If you use the access control option, the *directory* statement specifies one access control entry (ACE) managed object that references the directory managed object for each entry specified with the access control option. The ACE managed object has the following characteristics:

- It is unnamed.
- It has operating lifetime.
- It has the same scope as the directory.

See Also

scope

Example

```
directory [SYSHLP.EXAMPLES.FMS.MESSAGE] protection private
     access control ("(IDENTIFIER=[FMS], ACCESS=READ)");
```

This example specifies the directory [SYSHLP.EXAMPLES.FMS.MESSAGE]. The protection private option specifies that no users have access to this directory. The access control option grants the user FMS read access to the directory.

end

Ends a group of statements. For more information, see the corresponding opening statement.

Syntax

```
end if ;
end option ;
end part ;
end product ;
end remove ;
end scope ;
```

error

The *error* statement displays an error message during an installation or reconfiguration operation. The text is from a PTF text module.

Syntax

error *name ;*

Parameter

name
Specifies, as a quoted or unquoted string, the name of the associated PTF text module. The name you specify can be from 1 to 31 characters in length and must be unique among all names in the same product description.

Description

The *error* statement specifies a text module you want to display during an installation or reconfiguration operation. The *error* statement must be immediately contained within an *if* group.

The POLYCENTER Software Installation utility processes *error* statements in lexical order. The utility displays prompt and help text during the validation phase (which follows the configuration phase).

After receiving an error, the utility prompts the user to continue or terminate the operation. If the operation is not executed interactively, it terminates. Each *error* statement results in a separate prompt.

The *error* statement is a utility directive and does not specify a managed object.

You must supply text in the associated product text module. The module must contain a =prompt directive line.

See Also

if

Example

Suppose the product description file contains the following lines:

```
if (<hardware processor model 7>) ;
    error UNSPROC ;
end if ;
```

The corresponding module in the PTF could contain the following lines:

```
1 UNSPROC
=prompt Not supported on MicroVAX I.
This product is not supported on the MicroVAX I processor.
```

error

If the processor model is 7, the system displays the following message:

```
Not supported on MicroVAX I.

This product is not supported on the MicroVAX I processor.
Terminating is strongly recommended.  Do you want to terminate? [YES]
```

execute install...remove

The *execute install* statement specifies commands that you want to execute when the product is installed. The *remove* part of the statement indicates commands you want to execute when the product is removed from the execution environment.

Syntax

execute install *(command,...)* remove (command,...) [uses *(file,...)*] ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Option

uses *(file,...)*
Specifies the files required to execute the commands you specified in the **command** parameter. Specify the list of required files with the **uses** option. By default, this statement does not require files.

Description

The *execute install...remove* statement specifies commands that you want to execute when the product is installed. The *remove* part of the command indicates commands that execute when the product is removed from the execution environment. You specify actions by entering a command line, which the utility passes to the DIGITAL Command Language (DCL) interpreter running in a subprocess.

The *scope* statement controls the execution of the commands; the commands execute once in each scope.

If you need files for the *execute install...remove* statement, specify the **uses** option. Each file you specify with the uses option must be present in the product material.

The *execute install...remove* statement causes the utility to define logical names for use by the subprocess that executes the specified commands. The commands should use these logical names to reference files, as follows:

- PCSI$SOURCE is a directory specification that points to the location of a sequential kit or to the root directory of a kit in reference copy format.
- PCSI$DESTINATION is a root directory specification that points to root directory for the current scope where product material will be placed.
- PCSI$SCRATCH is a subdirectory under the user's login directory that can be used by commands for temporary working space.

The *execute install...remove* statement is a utility directive and does not specify a managed object.

execute install...remove

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute
    install "@pcsi$source:[sysupd]load_loadable_image.com"
    remove "@pcsi$source:[sysupd]unload_loadable_image.com"
    uses ([sysupd]load_loadable_image.com,
          [sysupd]unload_loadable_image.com) ;
```

In this example, the *execute install...remove* statement sets up command procedures to run when the product is installed and removed. The **uses** option specifies the file names of the command procedures for installation and removal.

execute login

The *execute login* statement displays a message to users that they should execute the specified commands when the product is made available to a process.

Syntax

execute login *(command,...)* ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility displays in a message to the user.

Description

The *execute login* statement displays a message to users that they should execute the specified commands when the product is made available to a process. You can use this statement to advise users of commands they should add to their login files.

The *execute login* statement does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute login "$ @USER_START" ;
```

In this example, the *execute login* statement displays a message to users about adding a line to their LOGIN.COM file.

execute postinstall

The *execute postinstall* statement specifies commands that execute after the product is made available to the system.

Syntax

execute postinstall *(command,...)* [uses *(file,...)*] ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Option

uses *(file,...)*
Specifies the files required to execute the commands you specified in the **command** parameter. Specify the list of required files with the **uses** option. By default, this statement does not require files.

Description

The *execute postinstall* statement specifies commands that execute after the product is made available to the system. You specify actions by entering a command line, which the utility passes to the DIGITAL Command Language (DCL) interpreter running in a subprocess.

The *scope* statement controls the execution of the commands; the commands execute once in each scope.

If you need files for the *execute postinstall* statement, specify the **uses** option. Each file you specify with the uses option must be present in the product material.

The *execute postinstall* statement causes the utility to define logical names for use by the subprocess that executes the specified commands. The commands should use these logical names to reference files, as follows:

- PCSI$SOURCE is a directory specification that points to the location of a sequential kit or to the root directory of a kit in reference copy format.
- PCSI$DESTINATION is a root directory specification that points to root directory for the current scope where product material will be placed.
- PCSI$SCRATCH is a subdirectory under the user's login directory that can be used by commands for temporary working space.

The *execute postinstall* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute
    postinstall "@pcsi$source:[sysupd]product_cleanup.com"
    uses [sysupd]product_cleanup ;
```

In this example, the *execute postinstall* statement sets up a command procedure to run after the product is installed. The **uses** option specifies the file name of the command procedure.

execute release

The *execute release* statement specifies commands to execute when the existing product version is replaced with a later version.

Syntax

execute release *(command,...)* [uses *(file,...)*] ;

Parameter

(command,...)
Specifies the commands the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Option

uses *(file,...)*
Specifies the files required to execute the commands you specified in the **command** parameter. Specify the list of required files with the **uses** option. By default, this statement does not require files.

Description

The *execute release* statement specifies commands that execute when the existing product version is replaced with a later version. You specify actions by entering a command line, which the utility passes to the DIGITAL Command Language (DCL) interpreter running in a subprocess.

If you need files for the *execute release* statement, specify the **uses** option. Each file you specify with the uses option must be present in the product material.

The *execute release* statement causes the utility to define logical names for use by the subprocess that executes the specified commands. The commands should use these logical names to reference files, as follows:

- PCSI$SOURCE is a directory specification that points to the location of a sequential kit or to the root directory of a kit in reference copy format.
- PCSI$DESTINATION is a root directory specification that points to root directory for the current scope where product material will be placed.
- PCSI$SCRATCH is a subdirectory under the user's login directory that can be used by commands for temporary working space.

The *execute release* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute release "@pcsi$source:[sysupd]config" uses [sysupd]config.com ;
```

In this example, the *execute release* statement sets up a command procedure to run when the product is upgraded. The **uses** option specifies the file name of the command procedure.

execute start...stop

The *execute start* statement displays a message to users indicating commands they should execute when the product is made available on the processor to which it is bound. On installation or upgrade of the product, these commands are also executed. The *stop* part of the statement indicates commands they should execute when the product is made unavailable on the processor to which it is bound. On removal of the product, or when it is upgraded, these commands are also executed.

Syntax

execute start *(command,...)* stop *(command,...)* ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility displays in a message to the user and also passes to the command interpreter in the execution environment.

Description

The *execute start* statement displays a message to users indicating commands they should execute when the product is made available on the processor to which it is bound. On installation or upgrade of the product, these commands are also executed. The *stop* part of the statement indicates commands they should execute when the product is made unavailable on the processor to which it is bound. On removal of the product, or when it is upgraded, these commands are also executed.

(Note that the *stop* part of the command is required even if there are no commands that you want to execute when the product is made unavailable.)

If you need files for the *execute start...stop* statement, you must provide them with the *file* statement.

The *execute start...stop* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Examples

1.
```
execute
    start "@sys$startup:product_startup.com"
    stop "@sys$startup:product_shutdown.com" ;
```

In this example, the *execute start...stop* statement displays a message to users about command procedures they should run to start and stop the product.

2.
```
execute
    start "@sys$startup:abs_startup.com"
    stop "" ;
```

In this example, the *execute start...stop* statement displays a message to users about command procedures they should run to start the product. Note that there are no commands executed when the product is stopped.

execute test

The *execute test* statement specifies commands that execute during the functional test of the product.

Syntax

execute test *(command,...)* ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Description

The *execute test* statement specifies commands that execute during the functional test of the product. You specify actions by entering a command line, which the utility passes to the DIGITAL Command Language (DCL) interpreter running in a subprocess.

If you need files for the *execute test* statement, you must provide them with the *file* statement.

The *execute test* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute
    test "@sys$test:prod$ivp.com" ;
```

In this example, the *execute test* statement runs a command procedure to perform a functional test of the product.

file

The *file* statement creates a file in the execution environment. If a file of the same name already exists, the POLYCENTER Software Installation utility might replace the file, depending on the options specified.

Syntax

file *name*

```
[ [no] access control (access_control...)
  [no] archive
  assemble { copy
           { execute (command,...) [assemble uses (file,...)]
           { requires file (file,...)
  [no] generation generation
  image library
  image module module_name
  owner owner
  protection { execute }
             { private }
             { public  }
  release execute (command,...) [release uses (file,...)]
  { release merge   }
  { release replace }
  release notes
  size size
  source source
  [no] write ]
```

;

Parameter

name
Specifies the relative file specification of the file.

Options

[no] access control *(access_control...)*
Specifies the minimum access control entries (ACEs) that the file will have. By default, files have no added ACEs (no access control).

[no] archive
Allows you to preserve existing files during an upgrade. The utility appends *_OLD* to the end of the file type. For example, if you archived an existing file named STARTUP_TEMPLATE.SYS, the utility would rename it STARTUP_TEMPLATE.SYS_OLD. If there are several versions of the existing file, the utility purges the files before renaming the file type. By default, the POLYCENTER Software Installation utility does not preserve existing file versions (no archive). You cannot use this option with the release merge or write options.

assemble copy
Establishes the contents of the file by duplicating a file in the reference copy. This is the default.

assemble execute *(command,...)*
Establishes the contents of the file by executing the specified commands. Specify the command lines as quoted or unquoted strings.

assemble uses *(file,...)*
Specifies a list of additional files required by the assemble execute option. Specify the relative file specification. By default, the assemble execute option does not require additional files.

assemble requires file *(file,...)*
Specifies a list of files that you must assemble before the commands specified by the assemble execute option can execute. Specify the relative file specification. By default, the assemble execute option does not require additional files.

Note that required files are assembled before the file that requires them. For example, you can use this option to specify a main image that requires a shareable image.

[no] generation *generation*
Specifies that the file has an explicit generation number. Specify the number as an unsigned integer in the range 0 through 4294967295. Refer to the Description section for the meaning of this value. By default, the file does not have an explicit generation number (no generation), which is equivalent to 0.

image library
Specifies that the file's symbols are inserted into the system shareable image symbol table library. The file must be a shareable image.

image module *module_name*
Specifies the name of the module being inserted into the system shareable image symbol table library. If you do not specify this option, the module name is the same as the file name. This option has no effect unless specified with the image library option.

owner *owner*
Specifies the account name that owns the file. By default, the file is owned by the SYSTEM account. If you specify a numeric value for *name*, you must enclose the string in quotation marks; for example `"[11,7]"`.

protection execute
Sets the file protection, giving general users execute access.

protection private
Sets the file protection, giving general users no access.

protection public
Sets the file protection, giving general users read and execute access. This is the default.

release execute *(command,...)*
Specifies command lines (as quoted or unquoted strings) that execute to convert the file during a version upgrade. You cannot use this option with the release merge or release replace options.

release uses *(file,...)*
Specifies a list of additional files required by the release execute option. Specify the file specifications of the files. By default, the release execute option does not require additional files.

release merge
Specifies that library modules propagate during a version upgrade. If modules are present in the existing library but not in the new library, they are propagated to the new library. The file you specify with the **name** parameter must be a library. You cannot use this option with the archive, release execute, release replace, or write options.

release replace
Specifies that previous versions of the file are replaced during a version upgrade. You cannot use this option with the release execute or release merge options.

release notes
Specifies that the file is a release notes file. Users can extract the release notes to a file using the DCL command PRODUCT EXTRACT RELEASE_NOTES. The release notes are created in the file DEFAULT.PCSI$RELEASE_NOTES in the current directory.

size *size*
Do not specify this option in your PDF. When you package your product, the utility calculates the size (in blocks) of the files you specify and provides this option in the output PDF.

source *source*
Specifies the relative file specification of the file in the reference copy as a single quoted or unquoted string. Use this option when you want to give a file a different name in the execution environment. When users install your product, the utility uses this source file in the reference copy to create the file you specify with the **name** parameter. By default, the file in the reference copy and the file created in the execution environment have the same file specification.

[no] write
Specifies that you expect that users will modify the file during system operation. If you specify this option, during a version upgrade if the file already exists, it remains the active version. The default is no write. You cannot use this option with the archive or release merge options.

Description

The *file* statement creates a file in the execution environment. If a file of the same name already exists, the utility determines which file to preserve by comparing the generation numbers of the files.

If both files have generation numbers, the utility preserves the file with the higher generation number. If one file does not have a generation number or has a generation number of zero, the utility preserves the file that has a nonzero generation number. If both files have the same nonzero generation number, they are considered to be equivalent and either one may be used. Finally, if neither file has a nonzero generation number, a file conflict error is reported to the user.

The *file* statement specifies a file managed object. You specify the name of the file managed object using the **name** parameter. The name is a multicomponent name qualified by the relative directory path.

The *bootstrap block*, *link*, and *loadable image* statements can also specify references to a file managed object.

The file specified by the source option, if present, or otherwise by the **name** parameter, must be supplied as product material if the assemble copy option is in effect.

The assemble execute option causes the utility to define logical names for use by the subprocess that executes the specified commands. The commands should use these logical names to reference files, as follows:

- PCSI$SOURCE is a directory specification that points to the location of a sequential kit or to the root directory of a kit in reference copy format.
- PCSI$DESTINATION is a root directory specification under the user's login directory used as a staging area. The commands specified in the assemble execute option are responsible for creating a file in this directory tree whose name matches the one specified in the file name parameter. After the commands are executed, the utility moves the file to the product's destination directory for the current scope.
- PCSI$SCRATCH is a subdirectory under the user's login directory that can be used by commands for temporary working space.

Each file specified by the assemble uses option must be supplied as product material if the assemble execute option is in effect.

The scope and lifetime of the file managed object depend on whether it is contained within a *scope, end scope* pair as shown in Table PDL–2.

Table PDL–2 File Managed Object Scope and Lifetime

Type of Scope Group	Lifetime	Scope
Product[1]	Product	Product
Global	Assembly	Global
Bootstrap	Operating	Bootstrap
Processor	Operating	Processor

[1]If the assemble option is not assemble copy, the file managed object has assembly lifetime and product scope.

If you specify the access control option, the *file* statement specifies one access control entry (ACE) managed object that references the file managed object for each entry specified with the access control option. The ACE managed object has the following characteristics:

- It is unnamed.
- It has operating lifetime. It has the same scope as the file managed object.
- The system resolves managed object conflict by managed object collection.

If the image library option is in effect, it specifies an image library module managed object that references the file managed object. The image library module managed object has the following characteristics:

- It must be unique within the global scope.

- It has assembly lifetime and global scope.
- Managed object conflict is not recoverable.

See Also

bootstrap block
directory
link
loadable image
scope

Examples

1.
```
file [SYSLIB]FDVSHR.EXE image library ;
```

   The *file* statement in this example specifies that the symbols for the shareable image [SYSLIB]FDVSHR.EXE are inserted into the system shareable image symbol table library.

2.
```
file [SYSMGR]DECW$STARTUP.COM protection public ;
```

   The *file* statement in this example creates the file [SYSMGR]DECW$STARTUP.COM, giving users read and execute access.

3.
```
file [SYSMGR]DECW$SYLOGIN.COM protection public
    source [SYSMGR]DECW$SYLOGIN.TEMPLATE ;
```

   The *file* statement in this example creates the file [SYSMGR]DECW$SYLOGIN.COM in the execution environment using the contents of the file [SYSMGR]DECW$SYLOGIN.TEMPLATE from the reference copy.

4.
```
file [SYSMGR]DECW$SYSTARTUP.COM generation 56 archive write ;
```

   The *file* statement in this example creates the file [SYSMGR]DECW$SYSTARTUP.COM, preserving the existing copy. It also assigns a generation number to the file for conflict resolution. For example, if a version of the file already exists with a generation number of 60, the utility will preserve the copy with generation number 60 and will not create a new one. The write option indicates that users can modify this file and that, during a version upgrade, the existing version should remain the active one.

5.
```
file [SYSTEM]CALIBRATE.EXE
      assemble execute "@PCSI$SOURCE:[TEMP]CALIBRATE_LINK.COM"
      assemble uses ("[TEMP]CALIBRATE.OBJ",
                     "[TEMP]CALIBRATE_LINK.COM") ;
```

   The *file* statement in this example creates the file [SYSTEM]CALIBRATE.EXE in the execution environment by executing a command procedure to link the image. The link command procedure and object file are obtained from product material packaged in the kit. The link command in CALIBRATE_LINK.COM uses the link qualifier /EXECUTABLE=PCSI$DESTINATION:[SYSTEM]CALIBRATE.EXE to create the image file.

hardware device

The *hardware device* statement specifies that a required hardware device must be present in the execution environment. If the device is not available, the POLYCENTER Software Installation utility prompts the user to continue or to terminate the operation.

Statement Syntax

hardware device *name* ;

Function Syntax

< hardware device *name* >

Parameter

name
Specifies the device name of the hardware device. You must include the colon (:) at the end of the device name.

Description

Statement

The *hardware device* statement specifies a required hardware device. If the device is not available, the utility prompts interactive users to continue or to terminate the operation. Each failed *hardware device* statement results in a separate prompt.

If the operation is not executed in interactive mode, it is terminated.

Function

The *hardware device* function tests whether a specified device is present. The value is true if the device is present; otherwise, the value is false. If the function value is true, a reference to the device is created. Otherwise, no reference is created.

The name of the referenced device is the value of the **name** parameter.

While the reference exists, the utility does not permit an operation that makes the specified conditions false.

Examples

1.
```
hardware device LPA0: ;
```

   The *hardware device* statement in this example specifies that if the device named LPA0: is not present in the execution environment, the utility displays a message prompting the user to continue or to terminate the operation.

2.
```
if (<hardware device GAA0:>) ;
    file [SYSEXE]SMFDRIVER.EXE ;
end if ;
```

The *hardware device* function in this example provides the file [SYSEXE]SMFDRIVER.EXE if the device GAA0: is present.

hardware processor

The *hardware processor* statement indicates specific system processor models that must be present. If none of the specified system processor models is present, the POLYCENTER Software Installation utility prompts the user to continue or to terminate the operation.

Statement Syntax

hardware processor model *(model,...)* ;

Function Syntax

< hardware processor model *(model,...)* >

Parameter

model *(model,...)*
Specifies processor model identifiers.

Description

Statement

The *hardware processor* statement indicates specific system processor models. If none of the specified models is present, the utility prompts interactive users to continue or to terminate the operation. Each failed *hardware processor* statement results in a separate prompt.

If the operation is not executed in interactive mode, it is terminated.

Function

The *hardware processor* function tests whether a specified system processor model is present. The value is true if the model is present; otherwise, the value is false. If the function value is true, a reference to the system processor model is created. Otherwise, no reference is created.

The referenced system processor model is unnamed. While the reference exists, the utility does not permit an operation that makes the specified conditions false.

Example

Suppose the PDF contains the following lines:

```
if (<hardware processor model 7>) ;
    error UNSPROC ;
end if ;
```

You would have an UNSPROC module in the PTF similar to the following:

```
1 UNSPROC
=prompt Not supported on MicroVAX I.
This product is not supported on the MicroVAX I processor.
```

If the processor model is 7, the system displays a message indicating that the product is not supported on the MicroVAX I computer and prompts the user to continue or terminate the operation.

if

The *if* statement allows you to conditionally process a group of statements. The *if*, *else*, *else if*, and *end if* statements are used together to form an *if* group.

Syntax

if *expression* ;

PDL-statements

end if ;

if *expression* ;

PDL-statements

else ;

PDL-statements

end if ;

if *expression* ;

PDL-statements

else if *expression* ;

PDL-statements

.
.
.

else if *expression* ;

PDL-statements

else ;

PDL-statements

end if ;

Parameter

expression
Specifies the condition you want to test. See Section 2.4.3 for the definition of an expression.

Required Terminator

end if ;

if

Description

The *if* group allows you to conditionally process a group of statements. The utility executes the statements contained in the *if* group up to the first occurrence of an *else* statement, *else if* statement (if present), or *end if* statement if the expression evaluates to true.

else if

The *else if* statement is valid only if it is immediately contained in an *if* group and is not lexically preceded by an *else* statement.

The utility executes the statements lexically contained in the if group between the *else if* statement and the next occurrence of an *else*, *else if*, or *end if* statement if all of the following conditions exist:

- The result of evaluating the expression in the *if* statement is false.
- The result of evaluating the expression in all lexically preceding *else if* statements in the same *if* group (if present) is false.
- The result of evaluating the *else if* expression is true.

If any of these conditions are not satisfied, the utility also does not execute statements lexically contained in the *if* group between the *else if* statement and the next occurrence of an *else*, *else if*, or *end if* statement.

else

The *else* statement is valid only if it is immediately contained in an *if* group and is the only *else* statement in the *if* group. The utility executes the statements following the *else* statement (in the same if group) if both of the following conditions exist:

- The result of evaluating the expression in the *if* statement is false.
- The result of evaluating the expression in all lexically preceding *else if* statements in the same *if* group (if present) is false.

If either of these conditions is not satisfied, the utility does not execute statements lexically contained in the *if* group between the *else* statement and the *end if* statement.

Example

```
if (<software DEC VAXVMS DECWINDOWS>) ;
    file [SYSEXE]PRO$DW_SUPPORT.EXE ;
else if (<software DEC VAXVMS MOTIF>) ;
    file [SYSEXE]PRO$MOTIF_SUPPORT.EXE ;
else ;
    file [SYSEXE]PRO$CC_SUPPORT.EXE ;
end if ;
```

This example uses the *if* statement in conjunction with the *software* function to determine which file to provide, as follows:

1. If DECwindows is present, the utility provides the file [SYSEXE]PRO$DW_SUPPORT.EXE.
2. If DECwindows is not present and DECwindows Motif is present, the utility provides the file [SYSEXE]PRO$MOTIF_SUPPORT.EXE.
3. If neither DECwindows nor DECwindows Motif is present, the utility provides the file [SYSEXE]PRO$CC_SUPPORT.EXE.

infer

The *infer* statement tests the target system to determine if a product or product version is available.

Note

The *infer* statement is valid only in a transition PDF.

Syntax

infer available from install *file* ;

infer available from logical name *logical_name* ;

infer version from *file* ;

Parameters

file
Specifies the relative file specification of the file you want to test.

logical_name
Specifies the logical name you want to test.

Description

The *infer* statement tests the target system to determine if a product or product version is available. This statement is valid only in a transition PDF.

There are several types of *infer* statements:

- The *infer available* statement tests the target system to determine if the product named in the =product directive of the transition PDF is available. If no *infer available* statement is present, the product is inferred to be available.
  - The *infer available from install* statement tests whether the product is available only if the specified file is installed as a known image. The *scope* statement controls execution of this statement; the test executes in the specified scope.
  - The *infer available from logical name* statement tests whether the product is available only if the logical name you specify has a translation.
- The *infer version* statement tests the target system to determine the presence and active version of the product named in the =product directive of the transition PDF. The product is inferred to be present if the specified file is present on the system and absent otherwise. If the product is present, the active version is inferred to be the internal version number of the specified file. The *scope* statement controls execution of this statement; the test executes in the specified scope.

infer

See Also

scope

Examples

1. `infer available from logical name DOC$ROOT ;`

   The *infer available* statement in this example determines if the product is available by checking to see if there is a translation for the logical name DOC$ROOT. The name of the product that the statement is testing for is contained in the =product directive in the transition PDF.

2. `infer version from [SYSEXE]FORTRAN.EXE`

   The *infer version* statement in this example determines the active version of the product by checking to see if the file [SYSEXE]FORTRAN.EXE is present.

information

The *information* statement displays text from a specified text module in the PTF either before or after the execution of a POLYCENTER Software Installation utility operation.

Syntax

information *name* [[no] confirm | phase { after | before }] ;

Parameter

name
Specifies, as a quoted or unquoted string, the name of the associated PTF text module. The name you specify can be from 1 to 31 characters in length and must be unique among all names in the same product description.

Options

[no] confirm
Displays the contents of the text module and prompts the user for a response. The user can continue or terminate the operation. The confirm option does not have any effect in batch mode. The default is no confirm.

phase after
Displays the contents of the text module after the POLYCENTER Software Installation utility operation finishes. This option cannot be used with the phase before option.

phase before
Displays the contents of the text module during the configuration phase. This option is the default and cannot be used with the phase after option.

Description

The *information* statement displays text from a specified text module in the PTF either before or after the execution of a POLYCENTER Software Installation utility operation.

The POLYCENTER Software Installation utility processes *information* statements in lexical order. The utility displays prompt text and help text (if the user requests it).

You must supply product text in the associated product text module. The module must contain a =prompt directive.

Information statements that specify the phase after option do not display text if they are lexically contained in an option group that is not selected.

If you have *information* statements that specify the phase before option and they are lexically contained in a group with configuration choices, they are processed in lexical order and may be nested.

The *information* statement declares a name; it is not a variable.

information

The confirm option to the *information* statement causes the utility to prompt the user to continue or terminate the operation.

Example

Suppose the product text file for DEC Rdb for OpenVMS software contains the following lines:

```
1 RELEASE_NOTES
=prompt Release notes for Rdb/VMS available.
The release notes for Rdb/VMS are available in the file
SYS$HELP:RDBVMSV4.RELEASE_NOTES.
1 STOP_RDB_VMS_MONITOR
=prompt The Rdb/VMS monitor must be stopped before installation
The Rdb/VMS monitor must be stopped before Rdb/VMS may be installed.
Perform the following operation:
$ @SYS$MANAGER:RMONSTOP
```

The product description file could contain the following information statements:

```
information RELEASE_NOTES phase after ;
information STOP_RDB_VMS_MONITOR phase before confirm ;
```

If the user requests help, the first *information* statement displays the following text after the operation finishes:

```
Release notes for Rdb/VMS available.
The release notes for Rdb/VMS are available in the file
SYS$HELP:RDBVMSV4.RELEASE_NOTES.
```

If the user requests help, the first *information* statement displays the prompt text after the operation finishes:

```
Release notes for Rdb/VMS available.
```

If the user requests help, the second *information* statement displays the following text for the user during the configuration phase:

```
The Rdb/VMS monitor must be stopped before installation
The Rdb/VMS monitor must be stopped before Rdb/VMS may be installed.
Perform the following operation:
$ @SYS$MANAGER:RMONSTOP
Do you want to continue [YES]?
```

If the user does not request help, the utility displays the prompt text during the configuration phase:

```
The Rdb/VMS monitor must be stopped before installation
Do you want to continue [YES]?
```

Regardless of whether the help display option is set, the confirm option in the second statement forces the user to respond to the prompt before continuing.

link

The *link* statement specifies a second directory entry for a file or directory.

Syntax

link *name* from *source* [type hard] ;

Parameters

name
Specifies the file specification of the second directory entry.

from *source*
Specifies the file specification of an existing directory entry for the file or directory. The parameter string must be a single quoted or unquoted string. The referenced file or directory must be defined by a directory or *file* statement in the same product description.

Options

type hard
Specifies that a hard link be created. This is the default.

Description

The *link* statement specifies a second directory entry for a file or directory.

The scope and lifetime of the link managed object depend on whether it is contained in a scope group, as shown in Table PDL–3.

Table PDL–3 Link Managed Object Scope and Lifetime

Type of Scope Group	Lifetime	Scope
Product	Product	Product
Global	Assembly	Global
Bootstrap	Operating	Bootstrap
Processor	Operating	Processor

If the *link* statement is not contained in a *scope, end scope* pair or is contained in a scope product group, the link managed object has product lifetime and product scope.

Managed object conflict is unrecoverable.

See Also

directory
file
scope

Example

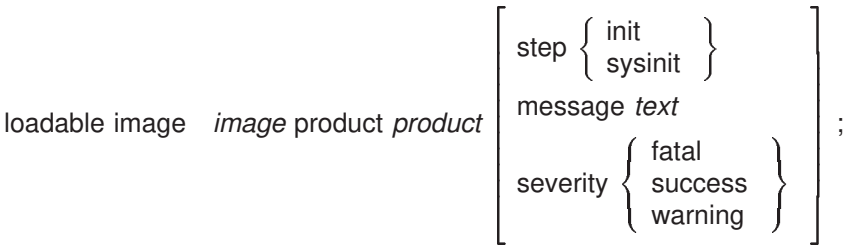
```
link [SYSEXE]FMS.EXE from [SYS$EXE]FMS.EXE ;
```

The statement in this example specifies that the file [SYSEXE]FMS.EXE is linked to the file [SYS$EXE]FMS.EXE.

loadable image

The *loadable image* statement places an image into the system loadable images table, SYS$LOADABLE_IMAGES:VMS$SYSTEM_IMAGES.DATA, and also into SYS$UPDATE:VMS$SYSTEM_IMAGES.IDX for compatibility with the System Management utility (SYSMAN).

Syntax



Parameters

image
Specifies the file name of the system loadable image. The name you specify must be defined in the same product description and must have bootstrap scope and product or assembly lifetime.

product *product*
Specifies the product mnemonic (as a single quoted or unquoted string of 1 to 8 characters) that uniquely identifies the loadable image. For user-written images, this should typically contain the string _LOCAL_.

Options

step init
Specifies that the system load the image during the INIT step of the booting process.

step sysinit
Specifies that the system load the image during the SYSINIT step of the booting process. This is the default.

message *text*
Specifies the message you want displayed using the severity option. The message must be a single quoted or unquoted string. Case is significant. By default, the severity option displays the message ''system image load failed.''

severity fatal
Specifies that if an error occurs while the image is being loaded, the system displays the message and bugchecks; if no error occurs, processing continues.

severity success
Specifies that the system continue processing and not display a message regardless of whether an error occurs while the image is being loaded.

severity warning
Specifies that if an error occurs while the image is being loaded, the system displays the message and continues; if no error occurs, the system continues and does not display the message. This is the default.

Description

The *loadable image* statement places an image into the system loadable images table, SYS$LOADABLE_IMAGES:VMS$SYSTEM_IMAGES.DATA, and also into SYS$UPDATE:VMS$SYSTEM_IMAGES.IDX for compatibility with the System Management utility (SYSMAN).

The *loadable image* statement specifies a loadable image module managed object that has the following characteristics:

- It must be unique within the global scope.
- It has assembly lifetime and global scope.
- Managed object conflict is not recoverable.

The *loadable image* statement also refers to a file managed object specified using the image parameter.

See Also

file

Example

```
loadable image DDIF$RMS_EXTENSION product _LOCAL_
    message "DDIF Extension not loaded"
    severity warning ;
```

The statement in this example places the user-written image DDIF$RMS_ EXTENSION in the system loadable images table. If an error occurs while loading this image, the system displays the error message ‘‘DDIF Extension not loaded.’’

module

The *module* statement creates one or more modules in a command, help, macro, object, or text library.

Syntax

module *file* type *type* module *module_name* [[no] generation *generation* | [no] globals | library *library* | [no] selective search | size *size*] ;

Parameters

file
Specifies the relative file specification of the file that contains the modules.

type *type*
The library type. Table PDL–4 lists the keywords you can specify with this parameter.

Table PDL–4 Library Types for Module Statement

Keyword	Library Type	Default
Command	Command definition library	[SYSLIB]DCLTABLES.EXE
Help	Help library	[SYSHLP]HELPLIB.HLB
Macro	Macro library	[SYSLIB]STARLET.MLB
Object	Object library	[SYSLIB]STARLET.OLB
Text	Text library	[SYSLIB]STARLETSD.TLB

module *module_name*
Specifies the list of module names you are specifying.

Options

[no] generation *generation*
Specifies that the file has an explicit generation number. Specify the number as an unsigned integer in the range of 0 through 4294967295. Refer to the Description section for the meaning of this value. By default, the file does not have an explicit generation number (no generation), which is equivalent to zero.

[no] globals
Specifies whether the global symbol names of the modules you are inserting into an object library are included in the global symbol table. You can use this option with object libraries only. By default, the global symbols of the module are inserted into the global symbol table.

library *library*
Specifies the relative file specification of the library. The file you specify must be a library of the type you specified with the **type** parameter.

module

[no] selective search

Specifies whether the input modules being inserted into the library are available for selective searches by the linker (by default they are not). For more information about selective searches, see the *OpenVMS Linker Utility Manual*.

size *size*

Specifies the size of the file as an unquoted string that specifies an unsigned integer. The utility supplies this option during a package operation; the utility ignores it if it is supplied as input to a package operation.

Description

The *module* statement creates one or more modules in a command, help, macro, object, or text library.

The name of a module managed object is specified using the module option.

The module managed object has assembly lifetime, and its scope is the same as the library.

If a module of the same name already exists, the utility determines which module to preserve using the generation numbers. If both modules have generation numbers, the utility preserves the module with the higher generation number. If one module does not have a generation number or has a generation number of zero, the utility preserves the module that has a nonzero generation number. If the generation numbers are equal or not present, an error is returned.

Examples

1.
```
module [SYSUPD]CDD.CLD type command module CDD ;
```

   The statement in this example creates the command module CDD in the default command library [SYSLIB]DCLTABLES.EXE using the file [SYSUPD]CDD.CLD.

2.
```
module [SYSUPD]HELP.HLP type help module FOO_HELP ;
```

   The statement in this example creates the help module FOO_HELP in the default help library [SYSHLP]HELPLIB.HLB using the file [SYSUPD]HELP.HLP.

3.
```
module [SYSUPD]SPI$CONNECT.MAR type macro
    library [SYSLIB]LIB.MLB module TEST ;
```

   The statement in this example creates the macro module TEST in the macro library [SYSLIB]LIB.MLB using the file [SYSUPD]SPI$CONNECT.MAR.

4.
```
module [SYSUPD]COBRTL.OBJ type object module TEST2;
```

   The statement in this example creates the object module TEST2 in the default object library [SYSLIB]STARLET.OLB using the file [SYSUPD]COBRTL.OBS.

5.
```
module [SYSUPD]PROTOTYPE_BOOK.TXT type text
    library [SYSLIB]LPS$FONT_METRICS.TLB module FONT;
```

   The statement in this example creates the text module FONT in the text library [SYSLIB]LPS$FONT_METRICS.TLB using the file [SYSUPD]PROTOTYPE_BOOK.TXT.

network object

The *network object* statement uses a command procedure to create a DECnet network object.

Syntax

network object *name* with *(parameters,...)* ;

Parameters

name
Specifies the name of the network object. The network object name is passed to the command procedure as P1.

with *(parameters,...)*
Specifies the list of parameters that are passed to the command procedure that creates the network object. Each parameter must be a single unquoted or quoted string that specifies P2 through P5, in order. Refer to the Description section for the meaning of the parameters.

Description

The *network object* statement uses a command procedure (SYS$UPDATE:PCSI$CREATE_NETWORK_OBJECT.COM) to create network objects. The command procedure determines whether DECnet Phase IV or DECnet Phase V is running on the system. If Phase IV is being used, the command procedure runs the Network Control Program (NCP) utility to create the network object. Otherwise, it runs the Network Control Language (NCL) utility. PCSI passes the following parameters to the command procedure:

- P1 specifies the name of the network object (using the **name** parameter).
- P2 specifies the object number (for DECnet Phase IV systems only).
- P3 specifies the user name associated with the object (for DECnet Phase IV systems only). If you specify a user name, it must already exist.
- P4 specifies optional parameters to use with the NCP command DEFINE OBJECT for DECnet Phase IV objects.
- P5 specifies optional parameters to use with the NCL command CREATE SESSION CONTROL APPLICATION for DECnet Phase V objects.

When you remove a product that created network objects, the POLYCENTER Software Installation utility uses a command procedure (SYS$UPDATE:PCSI$DELETE_NETWORK_OBJECT.COM) to delete network objects associated with your product.

Note

In a future version, the POLYCENTER Software Installation utility may create and delete these managed objects directly without the use of command procedures. If this is the case, these statements will continue to function, but the command procedures may not be maintained or shipped with future versions of the utility.

network object

The *network object* statement specifies a network object managed object that has the following characteristics:

- Its name is the value of the **name** parameter. The name must be unique with respect to all network object names in the processor scope.
- It has operating lifetime and processor scope.
- Managed object conflict is not recoverable.

Example

```
network object k$test with ("number 107", "user KRYPTON") ;
```

In this example, the *network object* statement creates a network DECnet Phase IV object named k$test. Its object number is 107 and it will execute as user [KRYPTON].

option

The *option* statement is a utility directive that specifies a group of elements that the user can choose to make either all present or all not present. This statement allows you to present options to the user.

Statement Syntax

option *name* [default *value*] ;

PDL-statements

end option ;

Function Syntax

< option *name* [default *value*]>

Parameter

name
Specifies, as a quoted or unquoted string, the name of the associated PTF text module. The name you specify can be from 1 to 31 characters in length and must be unique among all names in the same product description.

Option

default *value*
Specifies the default value for the variable. The value must be either 1(true), 0(false), yes, no, true, or false; the default is 1 (true).

If you specify an *option* statement with the default value 0, and it contains other *option* statements, any defaults in the contained *option* statements apply only when the top-level *option* statement is selected.

Required Terminator

end option ;

Description

Statement

The *option* statement specifies a group of elements that the user can choose to make either all present or all not present.

You can nest option groups. The POLYCENTER Software Installation utility processes nested options in lexical order. An option group containing *option* statements must be selected and processed before you can select any of the options it contains.

The prompt text is a string that describes the option. The utility displays this text to the user when prompting for a value. If the user sets the session's help display option, the utility displays the accompanying text, which should describe each option in as much detail as necessary.

option

The utility gets the default value for an option from either:

- The *option* statement in the PDF (if you use the default option).
- The product database (if you use the /PRODUCER=CURRENT qualifier to the PRODUCT command). This qualifier causes the utility to use the default values for the current version of the product.
- The product configuration file (PCF), if you created one.

You must supply text in the associated product text module. The module must contain a =prompt directive.

Function

The option function returns true if the user selects the option and false if the user does not select the option. The user can select options during the configuration phase. All option functions default to 1 (true).

See Also

part

Examples

1.
```
option NET ;
    file [SYSEXE]NETSERVER.COM ;
    file [SYSEXE]NETSERVER.EXE ;
    file [SYSHLP]NCPHELP.HLB ;
    option NET_A default 0 ;
        file [SYSEXE]FAL.COM ;
        file [SYSEXE]FAL.EXE ;
        end option ;
    option NET_B ;
        file [SYSEXE]REMACP.EXE ;
        file [SYSMGR]RTTLOAD.COM ;
        file [SYS$LDR]CTDRIVER.EXE ;
        file [SYS$LDR]RTTDRIVER.EXE ;
        end option ;
    end option ;
```

If the product description file contains the lines above, the product text file contains the corresponding text:

```
1 NET
=prompt network support
This option allows you to participate in a DECnet network.
1 NET_A
=prompt incoming remote file access
This option allows file access from other nodes in a DECnet network.
1 NET_B
=prompt incoming remote terminal access
This option allows users on other nodes in a DECnet network to log
in.
```

The user must select option NET before NET_A or NET_B are available for selection. Therefore, NET is processed before NET_A or NET_B.

2.
```
if (<option A>) ;
        file [SYSEXE]A.EXE ;
else ;
        file [SYSEXE]B.EXE ;
end if ;
```

The product text file contains the corresponding text:

```
1 A
=prompt the X capability
This feature provides the A capability, but you will not get the B
capability.
```

In this example, if the user selected the A option, the utility provides the file [SYSEXE]A.EXE. Otherwise, the utility provides the file [SYSEXE]B.EXE.

part

The *part* statement allows you to specify a message about a group of elements that you want to display to users.

Syntax

part *name* ;

PDL-statements

end part ;

Parameter

name
Specifies, as a quoted or unquoted string, the name of the associated PTF text module. The name you specify can be from 1 to 31 characters in length and must be unique among all names in the same product description.

Required Terminator

end part ;

Description

The *part* statement allows you to specify a message about a group of elements that you want to display to users. For example, you can display a message about a group of software products that you are providing as part of a platform.

You can nest part groups. The POLYCENTER Software Installation utility processes nested parts in lexical order.

If the user requests help, the utility displays the accompanying text, which describes each part in detail. The prompt text is a string that describes the part. The utility displays this text to the user.

You must supply text in the associated product text module. The module must contain a =prompt directive.

See Also

option

Example

Suppose the product description file contains the following lines:

```
part DESKTOP ;
software DEC AXPVMS DECPRINT
    version required V4.0 component ;
software DEC AXPVMS DOCUMENT
    version required V2.0 component ;
software DEC AXPVMS LSE
    version required V3.0 component ;
end part;
```

The product text file contains the corresponding text:

```
1 DESKTOP
=prompt Desktop Publishing Tools
This product provides the following desktop publishing products:
DECprint Printing Services software, VAX DOCUMENT software, and the VAX
Language Sensitive Editor (LSE) software are the required products that
comprise this platform.
```

This example shows how to use the *part* statement to display a message about the required software products that the desktop platform provides.

patch image (VAX only)

The *patch image* statement updates an executable image.

Syntax

patch image *name* with *source* ;

Parameters

name
Specifies the relative file specification of the executable image you want to update.

with *source*
Specifies the file specification of the file containing the update commands. The file that contains the update commands should contain OpenVMS Image File Patch Utility (PATCH) commands.

Description

The *patch image* statement updates an executable image. Use this statement when it is inconvenient to provide a new image.

You must supply the file containing the update commands as part of the product material.

The *patch image* statement specifies a maintenance managed object that has the following characteristics:

- Its name is the same as the **name** parameter of the product group in which the statement is lexically contained; it is a multicomponent name qualified by the relative file specification of the file that is being updated. It must be unique with respect to all maintenance managed objects in all scopes.
- It has assembly lifetime, and its scope is the same as that of the file being updated.
- Managed object conflict is unrecoverable.

Example

```
patch image [SYS$LDR]SYS.EXE with [SYSUPD]VERSION_PATCH.PAT ;
```

This statement provides a file, [SYSUPD]VERSION_PATCH.PAT, to patch the image [SYS$LDR]SYS.EXE.

patch text

The *patch text* statement updates a text file.

Syntax

patch text *name* with *source* ;

Parameters

name
Specifies the relative file specification of the text file you want to update.

with *source*
Specifies the file specification of the file containing the update commands (as a single quoted or unquoted string). The file that contains the update commands should contain SUMSLP commands.

Description

The *patch text* statement updates a text file. Use this statement when it is inconvenient to provide a new file.

You must supply the file containing the update commands as part of the product material. You must also supply the file that you want to update, but this file is not propagated to the reference copy. The POLYCENTER Software Installation utility uses it to calculate the input and output checksum values.

The *patch text* statement creates a temporary directory, identified by the logical name PCSI$SCRATCH, to compute a checksum value. The PCSI$SCRATCH directory is created as a subdirectory of SYS$SCRATCH.

The *patch text* statement specifies a maintenance managed object that has the following characteristics:

- Its name is the same as the **name** parameter of the product group in which the statement is lexically contained; it is a multicomponent name qualified by the relative file specification of the file that is being updated. It must be unique with respect to all maintenance managed objects in all scopes.
- It has assembly lifetime, and its scope is the same as that of the file being updated.
- Managed object conflict is unrecoverable.

Example

```
patch text [SYSUPD]VMSINSTAL.COM with [SYSUPD]VMSINSTAL.SLP ;
```

This statement provides a file, [SYSUPD]VMSINSTAL.SLP, to patch the text file [SYSUPD]VMSINSTAL.COM.

process parameter

The *process parameter* statement displays a message to users about process parameter requirements. Note that the POLYCENTER Software Installation utility does not adjust process parameters.

Syntax

process parameter *name* { consume *value* | require *value* | [maximum *value*] [minimum *value*] } ;

Parameter

name
Specifies the process parameter name. The name you specify must be valid on the system where the product executes.

Options

consume *value*
Specifies that the process parameter must be increased by the specified value. The value must be a single unquoted string that specifies an unsigned integer value. This option is valid if the data type of the value is signed integer or unsigned integer. Use this option when the product consumes a resource that is controlled by the process parameter.

maximum *value*
Specifies that the process parameter must have a value less than or equal to the specified value. The value must be a single unquoted string that specifies an integer value. This option is valid if the data type of the value is signed integer or unsigned integer.

minimum *value*
Specifies that the process parameter must have a value greater than or equal to the specified value. The value must be a single unquoted string that specifies an integer value. This option is valid if the data type of the value is signed integer or unsigned integer.

require *value*
Specifies that the process parameter must have the specified value. The value must be a single string that specifies a value of the parameter's type. This option is valid for any parameter data type.

Description

The *process parameter* statement displays a message to users after the installation about process parameter requirements. Note that the utility does not adjust process parameters.

Example

```
process parameter ASTLM minimum 6;
process parameter BYTLM require 32768;
process parameter PRCLM maximum 2;
process parameter FILLM maximum 40;
```

These statements display a message to users that a process that executes the product must have the following process parameters:

ASTLM greater than or equal to 6
BYTLM set to 32768
PRCLM less than or equal to 2
FILLM less than or equal to 40

process privilege

The *process privilege* statement displays a message to users about process privilege requirements. Note that the POLYCENTER Software Installation utility does not adjust process privileges.

Syntax

process privilege *(name[,...])* ;

Parameter

name
Specifies the process privilege names as a list. The privileges you specify must be valid on the system where the product executes.

Description

The *process privilege* statement displays a message to users after the installation about process privilege requirements. Note that the utility does not adjust process privileges.

Example

```
process privilege (group, oper, tmpmbx, sysnam) ;
```

The statement in this example displays a message to the user that processes using the product must have the GROUP, OPER, TMPMBX, and SYSNAM privileges.

product

The *product* group statement specifies the product identification and other descriptive information about the product.

Syntax

product producer base name version type ;

PDL-statements

end product ;

Parameters

producer
Specifies the legal owner of the software product.

base
Specifies the base on which the product executes. Possible values are *AXPVMS* and *VAXVMS*.

name
Specifies the product name or in the case of a patch or mandatory update kit, the name of the kit. Specify the product name with an *apply to* statement for patch and mandatory update kits. The combination of product name, producer, and base must be unique.

version
Specifies the product version.

type
Specifies the description type as one of the following keywords:

- **full**. A complete description that specifies application software that can be made available on a system (in addition to the operating system) to produce fully functional software.
- **operating system**. A complete description that can be made available on a system to produce fully functional software. Exactly one product that specifies this type must be available on a system.
- **partial**. A partial description that can update an existing version of the same product on a system to produce functionally updated software. The active version number of the product changes.
- **patch**. A partial description that can update an existing version of the same product on a system to produce functionally updated software. The active version number of the product does not change.
- **platform**. A complete description of a software system containing multiple separate products.
- **transition**. A partial description that supplies information about product versions that predate conversion to the POLYCENTER Software Installation utility or to another technology that can interoperate with the utility.

- **mandatory update**. A partial description that can update an existing version of the same product on a system to produce functionally updated software. The active version number of the product does not change. A product description of the mandatory update description type must contain an *apply to* statement to identify the product to which the update applies.

Option

operating system
Specifies a transition kit for an operating system. This option is valid only for transition kit types.

Required Terminator

end product ;

Description

The *product* group statement is a utility directive that specifies the product identification and other descriptive information about the product. It does not specify a managed object.

Examples

1.
```
product DEC AXPVMS FMS V2.4 full ;
    file [sysmsg]fdvshr.exe image library ;
    file [sysmsg]fmsmsg.exe ;
    file [sysexe]fmsfed.exe ;
    file [sysexe]fmsfaa.exe ;
    file [sysexe]fmsfte.exe ;
    directory [systest.fms] ;
    file [systest.fms]ivp.exe ;
    file [systest.fms]samp.flb ;
end product ;
```

The *product* statement in this example specifies information about the FMS product.

2.
```
product DEC VAXVMS VMS V6.2 transition operating system ;
```

The *product* statement in this example specifies information about the OpenVMS VAX operating system. The operating system option identifies the software as an operating system transition kit.

register module

The *register module* statement registers in the product database one or more existing modules in a command, help, macro, object, or text library.

Syntax

register module type *type* module (*module_name*,...)
[[no] generation *generation* | library *library*];

Parameters

type *type*

Specifies the library type. Table PDL–5 lists the keywords you can specify with this parameter.

Table PDL–5 Library Types for Register Module Statement

Keyword	Library Type	Default
Command	Command definition library	[SYSLIB]DCLTABLES.EXE
Help	Help library	[SYSHLP]HELPLIB.HLB
Macro	Macro library	[SYSLIB]STARLET.MLB
Object	Object library	[SYSLIB]STARLET.OLB
Text	Text library	[SYSLIB]STARLETSD.TLB

module *module_name*

Specifies the names of the modules contained within the library. If you are registering a text module, you can specify only one module name.

Options

[no] generation *generation*

Specifies that the module has an explicit generation number. Specify the number as an unsigned integer in the range of 0 through 4294967295. Refer to the Description section of the *module* statement for the meaning of this value. By default, the module does not have an explicit generation number (no generation), which is equivalent to zero.

library *library*

Specifies the file specification of the library. The file you specify must be a library of the type you specified with the **type** parameter.

Description

The *register module* statement registers in the product database one or more existing modules in a command, help, macro, object, or text library. Registering these modules in the product database allows the utility to detect conflicts with other modules.

Examples

1.
```
register module type help generation 5
          module (":=","=","@",ACCOUNTING,ALLOCATE,ANALYZE,APPEND,...) ;
```

   In this example, the *register module* statement registers several help modules. The generation option allows the utility to perform conflict resolution with these help modules.

2.
```
register module type object generation  1
        module (BAS$$CB,BAS$$COPY_FD,BAS$$DISPATCH_T,...) ;
```

   In this example, the *register module* statement registers several object modules. The generation option allows the utility to perform conflict resolution with these object modules.

remove

The *remove* statement removes managed objects from the product database and the system.

Note

You cannot use the *remove* statement in a transition PDF.

Syntax

remove ;

PDL-statements

end remove ;

Required Terminator

end remove ;

Description

The *remove* statement allows you to remove managed objects from the product database and from the system. Statements that normally provide managed objects (for example, the *file* statement) remove managed objects when contained within *remove, end remove*.

By using the *remove* statement in a partial, patch, or mandatory update kit, you can eliminate obsolete files from a previous version of your product. By using the *remove* statement in a full kit, you can eliminate objects provided by a previous installation mechanism (for example, VMSINSTAL).

Statements that do not provide managed objects function normally within *remove, end remove*.

You can nest *remove, end remove* within *scope, end scope*, if necessary.

Examples

1.
```
remove ;
  directory [SYSHLP.EXAMPLES.FOO] ;
  file [SYSHLP.EXAMPLES.FOO]SMLUS.COM ;
  file [SYSHLP.EXAMPLES.FOO]SMLUT.COM ;
  file [SYSHLP.EXAMPLES.FOO]SMLUU.COM ;
end remove ;
```

The statements in this example remove some files and a directory (if they exist) from the product database and the running system.

2.
```
scope bootstrap ;
  remove ;
  file [SYSEXE]PROD_PROC.EXE ;
  end remove ;
  file [SYSEXE]PROD_PROC_V2.EXE ;
end scope ;
```

The statements in this example remove a file in the bootstrap scope and then provide a new file.

rights identifier

The *rights identifier* statement uses a command procedure to create a rights identifier.

Syntax

rights identifier *name* with *(parameters,...)* ;

Parameters

name
Specifies the name of the rights identifier. The rights identifier name is passed to the command procedure as P1.

with *(parameters,...)*
Specifies the list of parameters that are passed to the command procedure that creates the rights identifier. Each parameter must be a single unquoted or quoted string that specifies P2 and P3, in order. Refer to the Description section for the meaning of the parameters.

Description

The *rights identifier* statement invokes a command procedure (SYS$UPDATE:PCSI$CREATE_RIGHTS_IDENTIFIER.COM) to create rights identifiers. This command procedure runs the Authorize utility to perform the function. PCSI passes the following parameters to the command procedure:

- P1 specifies the name of the rights identifier (using the **name** parameter).
- P2 specifies the optional qualifiers to use with the Authorize command ADD /IDENTIFIER.
- P3 specifies the /VALUE qualifier to use with AUTHORIZE command ADD /IDENTIFIER. You can specify this parameter only if the identifier does not already exist on the system.

When you remove a product that created rights identifiers, the POLYCENTER Software Installation utility uses a command procedure (SYS$UPDATE:PCSI$DELETE_RIGHTS_IDENTIFIER.COM) to delete rights identifiers associated with your product.

Note

In a future version, the POLYCENTER Software Installation utility may create and delete these managed objects directly without the use of command procedures. If this is the case, these statements will continue to function, but the command procedures may not be maintained or shipped with future versions of the utility.

The *rights identifier* statement specifies a rights identifier managed object that has the following characteristics:

- Its name is the value of the **name** parameter. The name must be unique with respect to all rights identifier names in the operating scope.

- It has operating lifetime.
- Managed object conflict is not recoverable.

Example

```
rights identifier PCSI_TEST
       with ("/attributes=DYNAMIC",
             "/value=IDENTIFIER:14600926") ;
```

In this example, the *rights identifier* statement creates a rights identifier named PCSI_TEST with a value of 14600926.

scope

The *scope* statement specifies for certain managed objects a scope and lifetime other than its defaults.

Syntax

scope { bootstrap | global | processor | product } ;

PDL-statements

end scope ;

Required Terminator

end scope ;

Description

The *scope* statement specifies for a managed object a scope and lifetime other than its defaults. You can nest *scope* statements. For more information about the scope and lifetime of managed objects, see Appendix B.

See Also

directory
execute install...remove
execute release
file
infer
link

Example

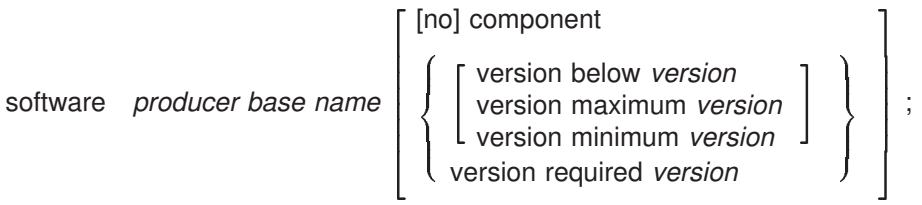
```
scope bootstrap ;
    file [SYSEXE]SYSBOOT.EXE ;
    file [SYSEXE]VMB.EXE ;
    bootstrap block [SYSEXE]VMB.EXE image [SYSEXE]BOOTBLOCK.EXE ;
end scope;
```

The statements in this example specify that the files VMB.EXE and SYSBOOT.EXE must be placed on every bootstrap disk.

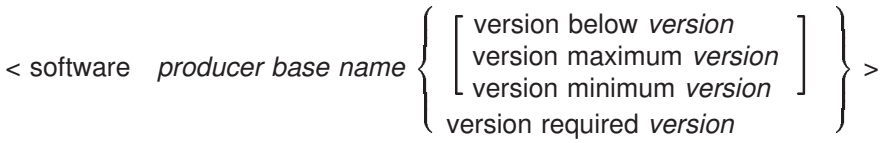
software

The *software* statement specifies a software product that must be available. The *software* function tests for the presence of a second product. You can also specify the version of the product that must be present.

Statement Syntax



Function Syntax



Parameters

producer
Specifies the legal owner of the software product.

base
Specifies the base on which the product executes. Possible values are *AXPVMS* and *VAXVMS*.

name
Specifies the product name. The combination of producer name, base, and product name must be unique.

Options

[no] component
Specifies that if the product is copied (using a copy operation) or packaged (using a package operation), the component products will be copied or packaged along with the product. The default is no component (the product does not need to be present during a copy or package operation).

version below *version*
Specifies the smallest invalid product version. Use this option to specify that the product version must be less than (but not equal to) the specified version. You cannot use this option with either the version minimum or version required options. By default, there is no smallest invalid version.

version maximum *version*
Specifies a maximum product version. The version must be a single quoted or unquoted string that specifies a version identifier. The version of the product that is available must be less than or equal to the specified version. You cannot

use this option with the version below option. By default, there is no maximum version.

version minimum *version*
Specifies a minimum product version. The version must be a single quoted or unquoted string that specifies a version identifier. The version of the product that is available must be greater than or equal to the specified version. By default, there is no minimum version.

version required *version*
Specifies a required product version. The version must be a single quoted or unquoted string that specifies a version identifier. The version of the product that is available must be equal to the specified version. You cannot use this option with either the version below, version maximum, or version minimum options. By default, there is no required version.

Description

Statement

The *software* statement specifies a software product that must be available in the execution environment. You can also specify a specific version of a product.

If you use the component option, the POLYCENTER Software Installation utility creates a copy of the referenced product when you copy your product.

If you reference a product that is not available or one that does not fit the constraints you specified, the utility prompts the user to continue or to terminate the operation.

If the operation executes in batch mode and a referenced product is not available, the operation terminates.

Note

If a referenced product is not available, Digital recommends that users accept the default prompt and terminate the operation.

If your product references another product with a *software* statement, the referenced product will be installed earlier than, and removed later than, your product. If two products reference each other (creating an infinite loop), the utility issues an error message.

Function

The *software* function tests for the presence of a second product. You can also specify a specific version of a product. The software function tests the state the system will be in when the operation finishes, not when the operation begins.

The function value is true if the following conditions exist; otherwise, the value is false:

- The product specified by the **producer**, **base**, and **name** parameters is available.
- The **version** option is omitted, or the available version satisfies the specified constraints.

If the function value is true, the utility creates a reference to the product. While the reference exists, the utility does not permit an operation that makes the specified conditions false. If the function value is false, the utility does not create a reference.

See Also

apply to
product
software

Examples

1.
```
software DEC VAXVMS FORTRAN
    version minimum V3.0 version maximum V5.0 ;
```

   The *software* statement in this example specifies that this product requires DEC Fortran software. The version must be between 3.0 and 5.0.

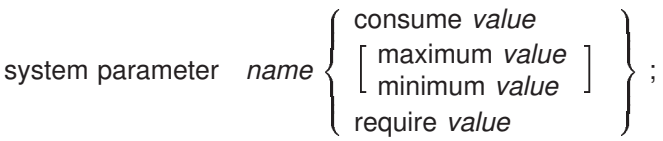
2.
```
software DEC VAXVMS FORTRAN version below V5.0 ;
```

   The *software* statement in this example specifies that this product requires DEC Fortran software. The version must be less than (but not equal to) 5.0.

system parameter

The *system parameter* statement allows you to display a message to users that expresses system parameter requirements for your product. Note that the POLYCENTER Software Installation utility does not change system parameters.

Syntax



Parameter

name
Specifies the name of the system parameter. The parameter you specify must be valid on the system where the product executes.

Options

consume *value*
Specifies that the value of the system parameter must be increased by the specified value. Use this option when the product consumes a resource that is controlled by the system parameter. The value must be a single unquoted string that specifies an unsigned integer value. This option is valid if the data type of the value is signed integer or unsigned integer. If you specify this value, you cannot specify the minimum, maximum, or require option.

maximum *value*
Specifies that the system parameter must have a value less than or equal to the specified value. The value must be a single unquoted string that specifies an integer value. This option is valid if the data type of the value is signed integer or unsigned integer.

minimum *value*
Specifies that the system parameter must have a value greater than or equal to the specified value. The value must be a single unquoted string that specifies an integer value. This option is valid if the data type of the value is signed integer or unsigned integer.

require *value*
Specifies that the system parameter must have the specified value. The value must be a single string that specifies a value of the parameter's type.

Description

The *system parameter* statement displays a message to users after the installation about system parameter requirements for your product. Note that the POLYCENTER Software Installation utility does not adjust system parameters.

Example

```
system parameter vaxcluster require 1 ;
system parameter tty_classname require "TT" ;
system parameter pagedyn consume 200 ;
```

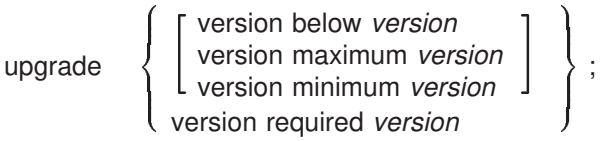
The statements in this example display the following messages:

```
This product requires the following system parameters
parameter VAXCLUSTER require 1

This product requires the following system parameters
parameter TTY_CLASSNAME require TT

This product requires the following system parameters
parameter PAGEDYN consume 200
```

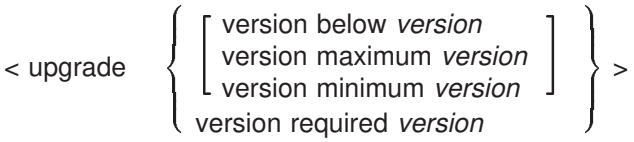
upgrade

The *upgrade* statement specifies product versions that must be available for a successful product upgrade. The *upgrade* function tests whether a version of the product is available.

Syntax



Function Syntax



Options

version below *version*
Specifies the smallest invalid product version. Use this option to specify that the product version must be less than (but not equal to) the specified version. You cannot use this option with either the version minimum or version required options. By default, there is no smallest invalid version.

version maximum *version*
Specifies a maximum product version that must be available. Use this option to specify that the product version must be less than or equal to the version you specify. You cannot use this option with the version below option. By default, there is no maximum version.

version minimum *version*
Specifies a minimum product version that must be available. Use this option to specify that the product version must be greater than or equal to the version you specify. By default, there is no minimum version.

version required *version*
Specifies a required product version that must be available. Use this option to specify that a specific product version must be present. You cannot use this option with either the version below, version maximum, or version minimum options. By default, there is no required version.

upgrade

Description

Statement

In a full, operating system, or platform PDF, the *upgrade* statement specifies which product versions must be available for a successful product upgrade. If there is no available version of the product, the POLYCENTER Software Installation utility ignores the *upgrade* statement and performs a new installation.

In a partial PDF, the *upgrade* statement is required to specify which product versions must be available in order for the partial kit to be applied successfully.

If the *upgrade* statement is not present, there is no constraint on the available version of the product.

Function

The *upgrade* function tests whether a version of the product is available. If a version of the product is available, the function returns true. If a version of the product is not available, the function returns false.

See Also

apply to
product
software

Examples

1.
```
product DEC VAXVMS VMS V5.5-2 partial ;
upgrade version minimum V5.5 version maximum V5.5-2 ;
   .
   .
   .
end product;
```

The *upgrade* statement in this example specifies that to install VMS Version 5.5–2, your system must be running at least VMS Version 5.5 and no higher than VMS Version 5.5–2.

A

Migrating from VMSINSTAL to the POLYCENTER Software Installation Utility

VMSINSTAL is an installation mechanism supplied by Digital. This appendix contains information about VMSINSTAL options and callbacks and their POLYCENTER Software Installation utility equivalents.

A.1 VMSINSTAL Options and Equivalents

Table A–1 lists some tasks that you may need to perform, the corresponding VMSINSTAL option, and the POLYCENTER Software Installation utility equivalent. Note that some VMSINSTAL options do not have an equivalent. In many cases, this is because the design of the POLYCENTER Software Installation utility eliminates the need for an equivalent.

Table A–1 VMSINSTAL Options and Equivalents

Task	VMSINSTAL Option	POLYCENTER Software Installation Utility Equivalent
Creating a file that specifies answers to installation questions	OPTIONS A	Create a product configuration file (PCF). This is similar to an auto-answer file in VMSINSTAL.
Specifying a temporary work directory	OPTIONS AWD	Specify the /WORK qualifier to the PRODUCT command.
Startup	OPTIONS B[1]	No equivalent.
Tracing callbacks during installation	OPTIONS C[2]	Use the /LOG and /TRACE qualifiers to the PRODUCT command. You can also use the /NOCOPY qualifier when debugging a product description file (PDF) to prevent the product material from being copied into the reference copy.
Manipulating product kits	OPTIONS G	Use the COPY/FORMAT=REFERENCE and COPY/FORMAT=SEQUENTIAL commands to manipulate product kits (see Chapter 4).
Suppressing VMSINSTAL prompts	OPTIONS I[2]	No equivalent.
Debugging a kit	OPTIONS K[2]	Use the /LOG and /TRACE qualifiers to assist in debugging a PDF.
Providing a log of installation operations	OPTIONS L	Use the /LOG and /TRACE qualifiers. This provides more information than OPTIONS L with VMSINSTAL.

[1]OpenVMS startup use only

[2]Developer's use only

(continued on next page)

Table A–1 (Cont.) VMSINSTAL Options and Equivalents

Task	VMSINSTAL Option	POLYCENTER Software Installation Utility Equivalent
Displaying or printing release notes	OPTIONS N	Use the release notes option to the *file* statement and the PRODUCT EXTRACT RELEASE_NOTES command. The release notes are created in the file DEFAULT.PCSI$RELEASE_NOTES in the current directory.
Performing an installation in test mode	OPTIONS Q[2]	No equivalent.
Installing a product in an alternate root	OPTIONS R	Use the /DESTINATION qualifier.
Pausing the installation at various points	OPTIONS RSP[2]	No equivalent.
Compiling information about the installation	OPTIONS S[2]	Use the /LOG and /TRACE qualifiers to the PRODUCT command.

[2]Developer's use only

A.2 VMSINSTAL Callbacks and Equivalents

To install a product using VMSINSTAL, you create a command procedure named KITINSTAL.COM that makes callbacks to VMSINSTAL. If you are migrating from VMSINSTAL to the POLYCENTER Software Installation utility, refer to Table A–2, which lists the VMSINSTAL callbacks and their equivalents.

Table A–2 VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Adding an identifier to the rights database	ADD_IDENTIFIER		Use the *rights identifier* statement.
Prompting the installer for information	ASK		To confirm the completion of preinstallation tasks, use the confirm option to the *information* statement. The product text file (PTF) contains the prompt and help text.
Not recording responses to installation questions		A	No equivalent.
Forcing a Boolean answer		B	No equivalent.
Preceding prompt with blank line		D	No equivalent.
Disabling terminal echo		E	No equivalent.
Displaying help text before the prompt		H	The *information* statement.
Answer must be an integer		I	No equivalent.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Returning input in lowercase		L	No equivalent.
Returning input in the same case		M	No equivalent.
Indicating a null response is acceptable		N	No equivalent.
Ringing the terminal bell before the prompt		R	No equivalent.
Indicating the response can be a string		S	No equivalent.
Returning input in uppercase		U	No equivalent.
Indicating the response can be Ctrl/Z		A	No equivalent.
Determining whether a license for the product is installed on the system	CHECK_LICENSE		No equivalent. License management is outside the domain of the utility.
Determining whether the network is running	CHECK_NETWORK		No equivalent. If you use a statement that references the DECnet network, the utility ensures that the network is available.
Determining whether there is sufficient disk space on the target device	CHECK_NET_ UTILIZATION		No equivalent. The utility ensures that sufficient disk space is available.
Determining whether a minimum version of software is present in the execution environment	CHECK_PRODUCT_ VERSION		Use the version minimum option to the *software* statement.
Limiting an installation to specified versions of the OpenVMS operating system	CHECK_VMS_ VERSION		Use the version minimum and version maximum options to the *software* statement, specifying DEC as the producer name, VAXVMS or AXPVMS as the base, and VMS as the product name.
Determining which is the most recent version of an image	COMPARE_IMAGE		You can manage file versions using the generation option to the *file* statement.
Determining whether the user has loaded the license for the product being installed on the system	CONFIRM_LICENSE		No equivalent. License management is outside the domain of the utility.
Providing for orderly exit from an installation	CONTROL_Y		No equivalent necessary; the utility provides this automatically.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Creating an account on the system	CREATE_ACCOUNT		Use the *account* statement.
Deleting obsolete files from a previous installation	DELETE_FILE		In full and operating system kits, the utility deletes files that are replaced during an upgrade. However, in a partial kit, you can remove obsolete files using the *remove* statement.
Locating files	FIND_FILE		If you want to determine whether an optional software product is available, use the *software* function. You do not need to determine whether a file is present before performing an operation that references it; the utility does this automatically.
Generating structure definition language (SDL) definition files	GENERATE_SDL		No equivalent.
Extracting the image file identification string for a file	GET_IMAGE_ID		If you want to determine the available version of a software product, use the *software* statement or function.
Obtaining a password for an account	GET_PASSWORD		No equivalent necessary; the utility provides this function.
Placing requirements on system parameters	GET_SYSTEM_ PARAMETER		Use the *system parameter* statement.
Displaying messages to the user	MESSAGE		Use the *information* statement to display information about pre- and postinstallation tasks. You do not need to provide error messages and progress information; the utility does this automatically.
Patching an image as part of the installation	PATCH_IMAGE		Use the *patch image* statement.
Moving a shareable image's symbol table to the system shareable image library when the patch is complete		I	No equivalent necessary. The image library option to the *file* statement controls its replacement in the image library.
Creating a journal file of patches		J	No equivalent.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Saving old versions of the image file		K	No equivalent necessary. The utility deletes existing versions.
Moving the file to the SYS$SPECIFIC directory		O	No equivalent necessary. The placement of the *file* statement that originally described the image within a scope group determines its placement.
Reinstalling the image when the patch is complete		R	No equivalent necessary; the utility does this automatically.
Queuing a print job to SYS$PRINT	PRINT_FILE		No equivalent.
Invoking a command procedure of product-specific callbacks	PRODUCT		No equivalent.
Adding a command to the system DCL table	PROVIDE_DCL_ COMMAND		Use the *module* statement with the **type command** parameter. You do not need to reinstall the system command table as a known image; the utility does this automatically.
Adding help to the DCL help library	PROVIDE_DCL_HELP		Use the *module* statement with the **type help** parameter.
Adding a new file to the system	PROVIDE_FILE		Use the *file* statement.
Placing the file in more than one location		C	No equivalent necessary.
Preserving old versions		K	No equivalent necessary. The utility deletes existing versions.
Adding the file to the SYS$SPECIFIC directory		O	Enclose the *file* statement in a scope processor group.
Specifying an input file that contains a list of logical names for the source files and their respective destinations		T	No equivalent necessary. Use one *file* statement for each file.
Adding a new image to the system	PROVIDE_IMAGE		Use the *file* statement. The utility can distinguish whether a file is a valid executable image.
Placing the file in more than one location		C	No equivalent necessary.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Dynamically patching ECOs into the new image file		E	No equivalent necessary. You should package the file with the correct ECO numbers already set.
Moving a shareable image's symbol table to the system shareable image library		I	Use the image library option to the *file* statement.
Preserving old versions		K	No equivalent necessary. The utility deletes existing versions.
Moving the file to the SYS$SPECIFIC directory		O	Enclose the *file* statement in a *scope processor* group.
Specifying an input file that contains a list of logical names for the source image files and their respective destinations		T	No equivalent necessary. Use one *file* statement for each file.
Changing the file name and file type of all versions of a file	RENAME_FILE		Use the archive option of the *file* statement to preserve an existing version of a file during an upgrade.
Restoring save sets of a product that is divided among several save sets	RESTORE_SAVESET		No equivalent necessary.
Running an image during installation	RUN_IMAGE		Use the *execute* statement or the assemble execute or release execute option to the *file* statement.
Specifying a UIC or protection code for product files	SECURE_FILE		Use the owner and protection options to the *directory* and *file* statements.
Modifying the access control list (ACL) of a device, directory, or file	SET	ACL	Use the access control option of the *file*, *hardware device* and *directory* statements.
Determining the default case (upper or lower) in which text from the installer is returned to the installation procedure	SET	ASK_CASE	No equivalent.
Running an installation verification procedure (IVP)	SET	IVP	No equivalent necessary. You can specify the *execute test* statement and invoke the functional test for a product with the /TEST qualifier to the PRODUCT command.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Calling a product's installation procedure after files have been moved to their target directories	SET	POSTINSTALL	Depending on your application, you can use the *execute postinstall* statement.
Purging files replaced by an installation	SET	PURGE	No equivalent necessary. The utility deletes existing versions.
Rebooting the system after the installation	SET	REBOOT	No equivalent.
Ensuring a high level of installation success	SET	SAFETY	No equivalent necessary. The utility provides the necessary disk management and reliability features.
Rebooting the system after the installation	SET	SHUTDOWN	No equivalent.
Specifying a product-specific startup command procedure	SET	STARTUP	Use the *execute start* statement.
Editing text files	SUMSLP_TEXT		Use the *patch text* statement.
Identifying installation peculiarities	TELL_QA		No equivalent necessary.
Exiting the installation procedure	UNWIND		No equivalent necessary. The utility controls the flow of the installation.
Updating an existing user account	UPDATE_ACCOUNT		Use the *account* statement to modify existing user accounts.
Making a file available for updating by copying it to a working directory	UPDATE_FILE		No equivalent necessary.
Modifying an identifier in the rights database	UPDATE_IDENTIFIER		Use the *rights identifier* statement to modify an existing rights identifier.
Updating a library	UPDATE_LIBRARY		Use the *module* statement with the appropriate parameter for the type of library you are updating. To update the shareable image library, use the image library option to the *file* statement. No equivalent exists to update RSX libraries.

B

Advanced PDF Concepts

This appendix contains information about some advanced PDF concepts such as managed object scope and lifetime.

B.1 Defining the Scope of a Managed Object

The **scope** of a managed object defines the degree of sharing that the managed object permits. For example, some objects are available only to certain processes, and some can be shared by all processes. The POLYCENTER Software Installation utility usually ensures that managed objects have the correct scope.

You might need to give a managed object a scope other than its default. Using the *scope* statement, you can ensure that the managed object is placed in the correct area on the system and that processes that need to can access it.

This section describes the different scopes that managed objects have:

- **Global scope** is the largest scope in which a single POLYCENTER Software Installation utility operation can have an effect. A single file that must be shared by every process in the computing facility must exist in global scope. Modules in system object libraries are examples of managed objects that must be in global scope. Writable databases might be in global scope.

  Global scope managed objects have the following characteristics:

  - They exist in every process in the computing facility (as established by an **initial system load**).
  - They are always stored together.
  - They are usually used by operating system products.

- **Bootstrap scope** managed objects function during system bootstrap when operating system facilities are unable to locate and use larger scopes. Drivers and loadable images that must be present before startup executes are examples of files that should be in the bootstrap scope.

  Bootstrap scope should be used for products that use device drivers, especially those drivers that must be read by the primitive file system. Because files in bootstrap scope are read by the primitive file system, they are read when not synchronized with the file system on other cluster members that might access the same disk.

  Therefore, those files must retain stable positions as long as the disk is in use from any system and must not be manipulated by online disk defragmentation operations, including those that use the MOVEFILE primitive.

  Bootstrap scope managed objects have the following characteristics:

  - They support specific computers and usually exist on the same disk volume as processor scope managed objects.

  - A computing facility can have several bootstrap scopes, and multiple computers can share the same bootstrap scope.
  - They are usually used by operating system products.

- **Product scope** managed objects are product specific. Most managed objects for a product reside in product scope. Product scope is the default scope for most objects; therefore, it is generally unnecessary to specify product scope. Product scope managed objects for different products can be stored together or separately.
- **Processor scope** managed objects exist in all processes executing on a single computer. For example, a logical name might exist in processor scope.

B.2 Updating Files

When you update your product with a partial, patch, or mandatory update kit, you can either explicitly state the scope of the file managed objects you are updating or let the utility determine the scope of the file managed objects:

- You can use the *scope* statement to ensure that the utility looks in a specific scope for the file managed object you want to update.
- If you do not use the *scope* statement, the utility searches the execution environment for a file managed object with the same name. If the utility finds the object, it replaces the object; if the utility does not find the file managed object, it provides a new file in product scope.

If you use the *patch* statement, the object you are updating must have been provided by your product. If you use the *module* statement, the object you are updating either must have been provided by your product or must be in global or bootstrap scope.

B.3 Managed Object Lifetimes

The **lifetime** of a managed object defines the duration or time period in which the managed object exists. For example, some managed objects are created each time a product is started; others are created before the product starts.

This section lists the lifetimes that a managed object receives as a result of its scope:

- **Product lifetime** managed objects are part of the product material. You create them when you package the product, and they exist in all reference and sequential copies. The POLYCENTER Software Installation utility does not create these managed objects, but it may copy or delete them. For example, an executable image is usually a product lifetime managed object.
- **Assembly lifetime** managed objects are created before the product is started. They do not need to be created each time the product is started and are identical in every execution environment. For example, a module in a system object library is an assembly lifetime managed object.
- **Operating lifetime** managed objects are created in the execution environment, before the product is started. They do not need to be created each time the product is started. For example, a user account is an operating lifetime managed object.

- **Dynamic lifetime** managed objects are created each time the product is started. For example, a server process that runs continuously to handle inbound network connections is a dynamic lifetime managed object.

Glossary

This glossary lists and defines the terms used in this guide.

integrated platform

A strategic combination of software products that is targeted toward a specific market or a set of applications that a company has standardized for internal use.

managed object

An entity that exists to support the proper functioning of a product. Files, directories, and accounts are all examples of types of managed objects.

package operation

A POLYCENTER Software Installation utility operation that uses the PDF, PTF, and product material to create a reference or sequential copy.

patch

A minor update to a software product that does not change the version level of the product.

PCF

See *product configuration file*.

PDB

See *product database*.

PDF

See *product description file*.

PDL

See *product description language*.

POLYCENTER Software Installation utility

A software product that allows you to create software kits and manage software (for example, installation, removal, configuration).

product configuration file (PCF)

A text file that specifies configuration choices for the POLYCENTER Software Installation utility to use in subsequent operations. For example, you can use a PCF to avoid specifying the same answers to installation questions when you have multiple installations to perform.

product database (PDB)

The repository in which the POLYCENTER Software Installation utility records information about events such as product installation and removal. Users can query the PDB to find out information about their environment.

product description file (PDF)

A text file that specifies the execution environment for your product.

product description language (PDL)

The set of statements that you use to write a PDF. See also *product description file*.

product material

The files associated with the product, excluding the PDF and PTF. Product material files are the output of the software engineering process.

product text file (PTF)

A text file that contains all the product-specific text that the POLYCENTER Software Installation utility can display during product manipulation (for example, description of options, informational text, copyright notice, and so forth).

PTF

See *product text file*.

reference copy

Format of a software product kit. In this form, the PDF, PTF, and all files that comprise the product are placed in a directory tree on a random-access device. OpenVMS is distributed in reference copy format on CD–ROM.

removal

An operation opposite to installation that reverses the effect of an installation. Product files are deleted and the PDB is updated.

sequential copy

Format of a software product kit. In this form, the PDF, PTF, and all files that comprise the product are packaged in a single container file. This container file can be placed either on a random-access device, such as a compact disc, or on a sequential access device, such as a magnetic tape. Most layered products are distributed in sequential copy format.

transition product description file (PDF)

A type of PDF that allows you to reference products not converted to the POLYCENTER Software Installation utility and to migrate products to the POLYCENTER Software Installation utility.

utility directive

A PDL statement that does not specify managed objects. Utility directives affect the operation of the POLYCENTER Software Installation utility but do not affect the execution environment.

Index

A

Account statement, 2–3, PDL–3
Apply to statement, PDL–5

B

Bootstrap block statement, PDL–7

D

Databases
  of software products, 1–3
Data types
  Base data types and values, 2–9
  Boolean, 2–9
  Signed integer, 2–9
  String, 2–9
  String data type constraints, 2–9
  Text module name, 2–9
  Unsigned integer, 2–9
  Version identifier, 2–9
DCL (DIGITAL Command Language) interface
  using to package software, 4–1
Directory statement, 2–3, 2–4, PDL–8

E

End statement, PDL–10
Error statement, PDL–11
Execute install statement, PDL–13
Execute login statement, PDL–15
Execute postinstall statement, PDL–16
Execute release statement, PDL–18
Execute remove statement, PDL–13
Execute start statement, PDL–20
Execute statement, 2–4
Execute stop statement, PDL–20
Execute test statement, 2–4, PDL–22

F

File statement, PDL–23
  uses, 2–3, 2–4, 2–5

H

Hardware device statement, 2–2, PDL–28
Hardware processor statement, 2–3, PDL–30
Help text, displaying for users, 3–3

I

If statement, PDL–31
Infer statement, PDL–33
Information statement, 2–4, PDL–35
Integrated platforms, 1–7
  packaging, 1–8

L

Link statement, PDL–37
Loadable image statement, 2–3, PDL–39
Logical name
  PCSI$DESTINATION, PDL–13, PDL–16, PDL–18, PDL–26
  PCSI$SCRATCH, PDL–13, PDL–16, PDL–18, PDL–26
  PCSI$SOURCE, PDL–13, PDL–16, PDL–18, PDL–26
Logical names, 1–7

M

Managed objects
  definition, 1–8
  scopes, B–1, B–2
Module statement, 2–5, PDL–41

N

Network object statement, 2–3, PDL–43

O

Option statement, 2–3, PDL–45

P

Package operations, 1–2
Part statement, PDL–48
Patch image statement, 2–3, PDL–50
Patch text statement, 2–3, PDL–51
PCSI$DESTINATION logical name, 1–7, PDL–13, PDL–16, PDL–18, PDL–26
PCSI$SCRATCH logical name, PDL–13, PDL–16, PDL–18, PDL–26
PCSI$SOURCE logical name, 1–7, PDL–13, PDL–16, PDL–18, PDL–26
PDF (product description file), 1–3
  creating, 2–1
  example, 2–10, 2–11
  file name format, 2–5
  guidelines for creating, 2–2
  product requirements checklist, 2–2
Platform PDF, 1–8, 2–14
Platforms
  See Integrated platforms
POLYCENTER Software Installation utility
  benefits of using, 1–1
  compared to VMSINSTAL, A–1
  creating the PDF, 2–1
  databases, 1–3
  guidelines for using, 2–2
  package operations, 1–2
  product configuration files, 1–3
Process parameter statement, 2–4, PDL–52
Process privilege statement, PDL–54
Product database
  definition of, 1–3
Product description file
  See PDF, Platform PDF, and Transition PDF
=Product directive, 3–2
  syntax, 3–2
Product formats, 4–1
Product name
  specifying in the PTF, 3–2
PRODUCT PACKAGE command
  format, 1–3, 4–1
Product statement, PDL–55
Product text file
  See PTF
=Prompt directive, 3–3
Prompt text
  including in the PTF, 3–3
PTF (product text file), 1–3, 3–1
  example, 3–4
  file format, 3–2
  file name format, 3–1
  including prompt text, 3–3
  sample file names, 3–1
  specifying the product name, 3–2

R

Reference copy, 1–4, 4–1
Register module statement, PDL–57
Remove statement, 2–5, PDL–59
Rights identifier statement, 2–3, PDL–61

S

Scope statement, PDL–63
Sequential copy, 1–4, 4–1
Software statement, 1–8, 2–2, PDL–64
String data type constraints, 2–9
  Access control entry, 2–9
  Command, 2–9
  Device name, 2–9
  File name, 2–9
  Identifier name, 2–9
  Module name, 2–9
  Processor model name, 2–9
  Relative directory specification, 2–9
  Relative file specification, 2–9
  Root directory specification, 2–9
  Unconstrained, 2–9
System parameter statement, 2–4, PDL–67

T

Transition PDF, 2–14
  definition, 2–14
  example, 2–15

U

Upgrade statement, PDL–69
Utility directives
  definition, 1–8
  example, 1–8

V

VMSINSTAL, migrating from, A–1