# CA IT Client Manager

## Packager and Installer for Linux and UNIX Administration Guide

**Release 12.8**

# CA Technologies Product References

This documentation set references to the following CA products:

- CA Advantage® Data Transport® (CA Data Transport)
- CA Asset Intelligence
- CA Asset Portfolio Management (CA APM)
- CA Common Services™
- CA Desktop Migration Manager (CA DMM)
- CA Embedded Entitlements Manager (CA EEM)
- CA Network and Systems Management (CA NSM)
- CA Patch Manager
- CA Process Automation
- CA Business Intelligence
- CA Service Desk Manager
- CA WorldView™
- CleverPath™ Reporter

# Contact CA Technologies

**Contact CA Support**

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

**Providing Feedback About Product Documentation**

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

# Contents


## Chapter 1: Introduction 11

About Software Management for Linux and UNIX ..... 11
Packager and Installer Environment ..... 12
Packager User Interface ..... 13
Multi-User Support (smgui) ..... 16
Remote Package Registration ..... 16

## Chapter 2: Getting Started with the Packager and Installer 17

Packager Installation Prerequisites ..... 17
Packager Installation ..... 18
Packager Installation Program ..... 19
Packager and Installer Installation Folders ..... 19
Packager Removal ..... 20

## Chapter 3: Using Packager and Installer for Linux and UNIX 21

PIF Products ..... 21
Delta Products ..... 22
Patch Products ..... 22
    Main Patch Package ..... 23
    Conventions for Patch Packages ..... 24
Virtual Packages and Alias Names ..... 24
Components of PIF Products ..... 25
    Standard Components ..... 25
    Platform-Specific Components ..... 25
    Language-Specific Components ..... 26
    Pre-install Components ..... 27
    Optional Components ..... 27
Files, Directories, and Symbolic Links in PIF Products ..... 27
Disk Space Reservation for Files in PIF Products ..... 28
Scripts within the Scope of Product Installation and Removal ..... 29
    Usage of Internal Procedures ..... 29
    Usage of Non-shell Scripts ..... 30
    Usage of Scripts with Parameters ..... 30
    Scripts Timer Modification ..... 31
    Progress of Scripts ..... 31
Dependencies of PIF Products ..... 31

Product Parameters ..... 32
Types of Parameters ..... 33
Ways to Provide Parameter Values ..... 33
Types of Parameter Values ..... 34
Default Values for Parameters ..... 34
Additional Parameter Descriptions ..... 34
Important Notes on Parameters ..... 35
Example: Using Parameters to Control Access Rights ..... 35
Response Files for PIF Products ..... 36
Internal Parameters ..... 36
How the Installer Resolves Parameters ..... 38
Templates for PIF Products ..... 39
Installation Dialogs ..... 40
Defining Installation Dialogs ..... 42
Standard Installation Dialogs ..... 42
Task-oriented Installation Dialogs ..... 42
Dynamically Detected Dialogs ..... 43
Dialog Validation Scripts ..... 43
Action Scripts ..... 45
Splash Screen ..... 45
Check Box Trees ..... 46
Font Specification for Installation Dialogs ..... 47
Specify the Input Order of a Dialog ..... 49
Installation Dialog Editor ..... 50
Workbox ..... 51
Graphics Areas ..... 53
Toolbox and Dialog Elements ..... 54
Properties Box ..... 61
Support of Various Packaging Formats ..... 61
Functions for PKG, RPM, AIX, HP-SD, and OSX Package Integration ..... 62
RPM-specific Integration Functions ..... 63
Restrictions with the Use of RPM Packages ..... 64
Native RPM and PKG Switches ..... 64
Installer Functions Overview ..... 65
No Dependency on the libncurses Library ..... 66
Self-installing PIF Products ..... 66
Integration of JRE in Self-installing PIF Products ..... 67
Product Family Concept ..... 68
Migration of Non-PIF Products ..... 68
Installer Compatibility Considerations ..... 69

## Chapter 4: Working with the Packager User Interface 71


Create a New PIF Package ..... 71
Save a PIF Package ..... 72
Build a PIF Package ..... 73
Register a PIF Package to Library or to Disk ..... 74
Modify a PIF Package ..... 75
Configure the Encryption Method for a PIF Package ..... 76
Unload a PIF Package ..... 76
Delete a PIF Package ..... 77
Import a PIF Package ..... 77
Create a Multilingual PIF Package ..... 78
    Select the Language for the Installation Dialogs ..... 79
    Copy and Change a Dialog Resource ..... 80
Patch Product Creation Wizard ..... 81


## Chapter 5: Building a Sample PIF Product 83


Requirements and Specifications of the UNIX Application MERCHANT ..... 83
Create the Prototype File and Specify Product Information for the PIF Product MERCHANT ..... 84
Define Dependencies for the PIF Product MERCHANT ..... 85
Specify Files for the PIF Product MERCHANT ..... 87
Add a Procedure to the PIF Product MERCHANT ..... 88
Change the Installation Path of the PIF Product MERCHANT ..... 90
Add a Dialog to Query the Installation Path of the PIF Product MERCHANT ..... 91
Add Language-specific Product Files to the PIF Product MERCHANT ..... 93
Localize the Installation Dialogs for the PIF Product MERCHANT ..... 95
    Localize the Installation Dialogs through the Command Line ..... 95
    Localize the Installation Dialogs Through the GUI ..... 95
Build the Final PIF Product MERCHANT ..... 96


## Chapter 6: Installer Tasks 99


Pre-Install Tasks ..... 100
    Validate the Software ..... 100
    Extract Pre-install Components to a Temporary Folder ..... 100
    Execute a Migration Script ..... 101
    Run Installation Dialogs ..... 102
    Run Post-Interview Script ..... 102
    Create a Backup ..... 102
Tasks During Product Installation ..... 102
    Read Parameter Values from the Environment ..... 103
    Execute Pre-Install Scripts ..... 103

Install Components ..... 104
Run Post-Install Scripts ..... 104
Register Product Information ..... 104
Post-Install Task ..... 104

## Chapter 7: Reference 105

File Naming Conventions ..... 105
Prototype File Format and Structure ..... 106
Product Information Section ..... 107
Component Section ..... 108
Default Section (Parameter Section) ..... 109
Dialog Section ..... 109
Fontlist Section ..... 110
Desktop Section ..... 110
Resource Section ..... 111
Keywords and Properties of the Prototype File ..... 111
Keywords ..... 111
Properties ..... 120
Packager and Installer Commands ..... 131
pifproto - Create a Prototype File for a PIF Product ..... 131
pifmk - Generate a PIF Product ..... 134
pifself - Generate a Self-Installing PIF Product ..... 135
pifextract - Extract a PIF Product ..... 137
pifdelta - Create a Delta Product ..... 138
piflib - Generate a PIF Library File ..... 139
pifpatch - Generate a Patch Product Automatically ..... 140
sd_registerproduct - Register a PIF Product in the Software Package Library ..... 141
lsm—Manage Software Installations on Linux and UNIX Systems ..... 143
Install a Self-Installing PIF Product ..... 148
Supported Linux and UNIX Operating Environments ..... 150
Language Identifiers ..... 151
Template Parameters and pifproto Command Switches ..... 152

## Chapter 8: Frequently Asked Questions 153

Log File Collection Tool dsminfo ..... 153
Self-installing PIF Product can be Splitted ..... 154
Register Multiple PIF Files As One Product ..... 154
The Installer Uses Directories on a Customer System ..... 155
The Packager Uses Directories to Store its Files ..... 155
X11 Emulation Needs to be Configured ..... 155
Packager and Installer Require a Certain Type of Java Runtime Environment ..... 156

Product Installation Logged in a File ..... 156
Selective Uninstall Supported ..... 157
Interaction Between a Prepif Script and a Response File During Silent Installations ..... 157
PIF Products Created With the Packager of PIF SDK Version 4.1 Installable on Unicenter Software Delivery 4.0 Agents ..... 158
Packager Installation on a Computer that has no DVD Drive ..... 158

**Index** **159**

# Chapter 1: Introduction

This guide describes the use and the features of the main components of CA Technologies Software Management for Linux and UNIX, the Packager and the Installer.

The Software Management features and their purposes are generally introduced; the main functions are also applied to build a sample PIF product. This example is provided in a separate chapter, where the creation of the sample PIF product is described step by step to make the process easy understandable.

The Software Management features described in this guide and the Packager and Installer software are available with the PIF Software Development Kit (SDK) Version 4.5.

Audience of this guide are administrators and developers who want to create software products to install on Linux and UNIX target computers. You should have extensive knowledge of the Linux and UNIX operating systems, including Java environments and X emulations, and you must be familiar with packaging software products and installing them on various Linux and UNIX operating environments on target computers.

If you want to distribute and install Linux or UNIX products over a network, you must understand the software delivery concepts, usage, and terminology.

## About Software Management for Linux and UNIX

CA Technologies software management solution for Linux and UNIX includes two main components, the Packager and the Installer. CA Technologies software management tools and functions for Linux and UNIX support all relevant Linux and UNIX based operating environments on the target computers in a network.

The Software Management Packager (Packager) is used to define, configure, and package software products. CA Technologies provides a standard for packaging software for Linux and UNIX systems, called the Product Interchange Format (PIF). The result of the packaging process is called a PIF product. PIF products can be created for and installed on all Linux and UNIX operating environments that are supported by the PIF SDK (software development kit). All Packager functions are available through both the graphical user interface (Java GUI) and the command line interface (CLI).

The Installer is located on the target computer and installs products in any operating environment supported by the PIF SDK, whether the installation is unattended or dialog-driven. Installation dialogs can be used for the product installation on all types of target computers.

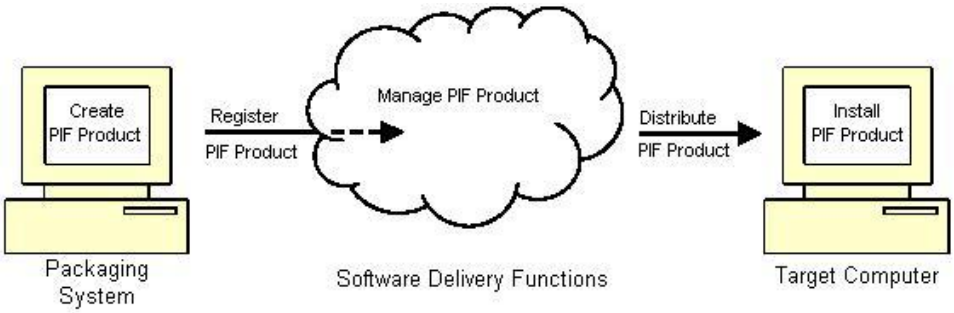
The Installer has features that make it easy to install products in packaging formats other than PIF, and you can manage any installed Linux or UNIX product on target computers.

# Packager and Installer Environment

In most cases, you will use Software Management integrated in a software delivery environment, but you can also use the Packager and Installer in a stand-alone mode.

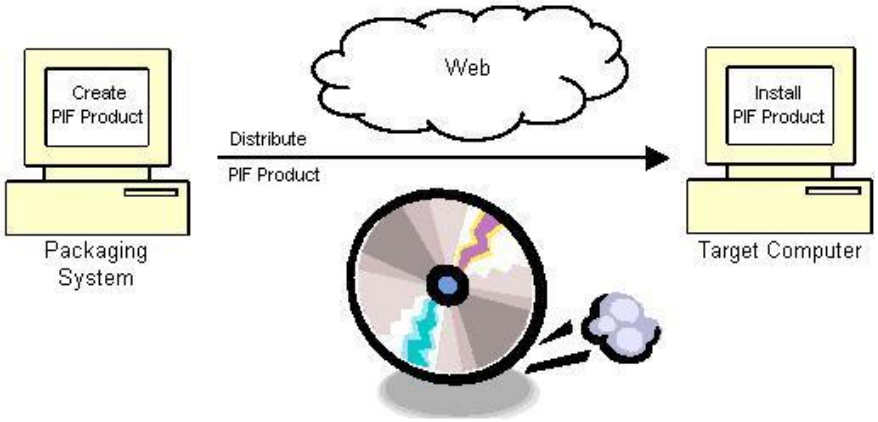
When you use the Packager and Installer in a software delivery environment, products are packaged in the PIF format on a packaging system using the Packager. Then, the PIF products are registered in the software package library on a manager system and distributed through software delivery functions using an installation order. On the target computer, software delivery agent functions receive and evaluate the installation order, and the Installer installs the PIF product.

The following illustration shows the Packager and Installer in a software delivery environment:




When you use the Packager and Installer in the stand-alone mode, the PIF products created on the packaging system are distributed to the target computers using media like CD-ROM or DVD, or through the web. On the target computers, the Installer installs the PIF products from that media.

The following illustration shows the Packager and Installer in the stand-alone mode:




# Packager User Interface

Software Management provides you with an easy-to-use graphical user interface (Java GUI) for the Packager, that helps you define, build, and manage the PIF product.

Additionally, all Packager functions are available through commands at the command line interface (CLI).

To run the Packager Java GUI, the Java Runtime Environment (JRE) Version 1.4 must be installed on the packaging computer, the monitor must be graphics-enabled, and the $DISPLAY variable must be set to the local terminal.

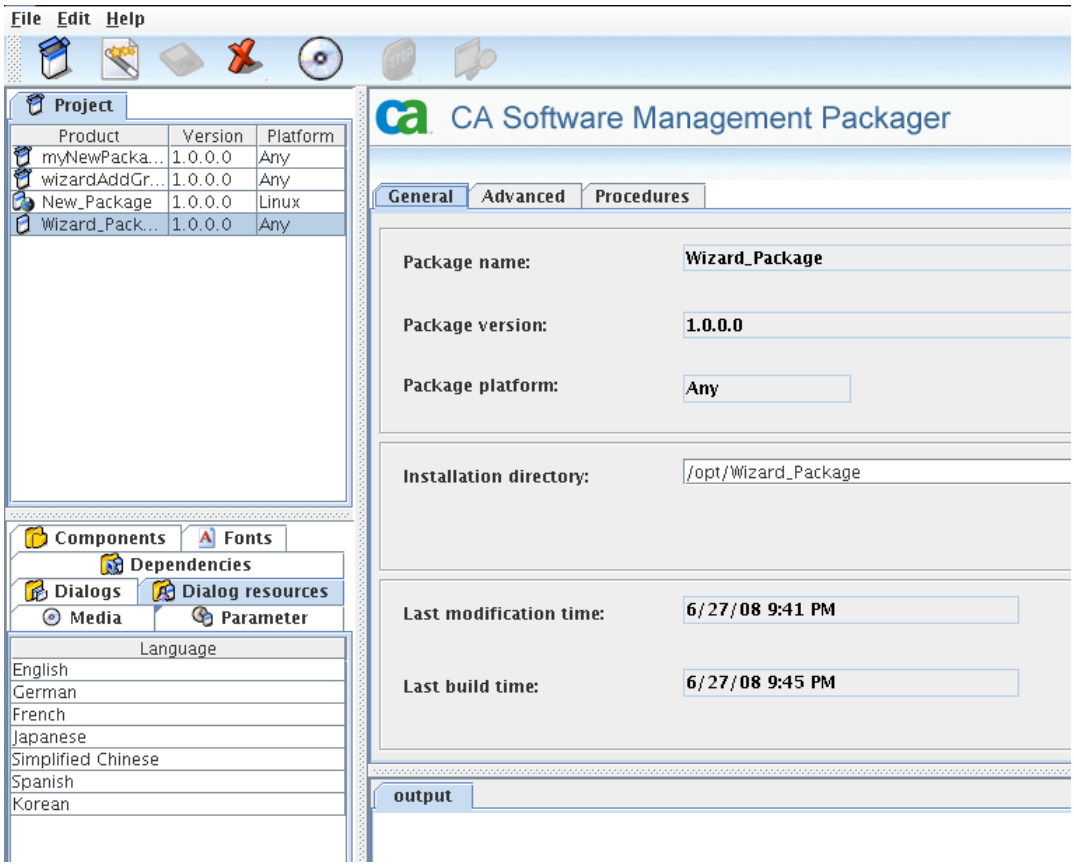
Before calling any Packager command, you must start a new login shell by entering one of the following commands at the command line:

```
su - [user]
```

```
. /etc/profile
```

You start the Packager GUI by executing the smgui command at the Packager's installation directory.

A sample Packager GUI is shown below:




You can perform the main software packaging tasks from the File menu on the menu bar or by clicking the appropriate icon on the toolbar.

The toolbar additionally provides the Cancel and the Test dialog functions. You can access the Packager online help by choosing Help, Help topics on the menu bar or by pressing the F1 key.

The following table includes the icons and associated functions that the toolbar provides:

| Icon | Icon Name | Description |
|---|---|---|
|  | New | Create the prototype file for a new PIF product. |
|  | Save | Save the PIF product. |
|  | Delete | Remove the PIF product. |
|  | Build | Build the PIF product. |
|  | Cancel | Stop a running sub process of the packaging process. |
|  | Test dialog | Preview the installation dialog being designed. |
|  | Package wizard | Launch the package creation wizard. |

The project pane is located in the upper-left side of the window and lists all created and imported products. To manage a product, you must select the appropriate item in the project pane.

The subproject pane is located beneath the project pane and shows relevant modules (subprojects) of the product selected in the project pane. Each tab in the subproject pane provides the current contents of the subproject. You can select an item to view and modify its properties in the work pane.

The work pane is located at the upper-right side of the GUI window and shows the properties of the selected product or subproject item.

The output pane lists status and error messages issued by commands that have been called through the Packager GUI. The error message is displayed in a separate tab of the output pane. This error message tab remains visible till a new build job is started; thus, allowing you to switch between patches without losing the error information.

# Multi-User Support (smgui)

Different users can now log in to the packager GUI (smgui) at the same time. This implies that different users can manage different packages at one time. However, one package cannot be changed by different users.

# Remote Package Registration

On Linux and UNIX computers, install the SD agent to enable remote package registration from the packager GUI or to enable the "registerproduct" command.

# Chapter 2: Getting Started with the Packager and Installer

This chapter describes how to start working with the Packager and Installer for Linux and UNIX. However, at the end of this chapter, there is also information about how to remove the Packager.

This section contains the following topics:

Packager Installation Prerequisites (see page 17)
Packager Installation (see page 18)
Packager Installation Program (see page 19)
Packager and Installer Installation Folders (see page 19)
Packager Removal (see page 20)

## Packager Installation Prerequisites

The Packager for Linux and UNIX can be installed on all 32-bit and 64-bit Linux and UNIX platforms that are supported by the CA IT Client Manager agent.

Depending on the hardware and software configuration of the computer on which you are going to install the Packager, the Packager installation program will start in a graphical or a character-oriented mode.

The Packager installation dialogs run in graphical mode if the following prerequisites are met:

- The Java Runtime Environment (JRE) 1.4 is installed on that computer
- The monitor is graphics-enabled
- The $DISPLAY variable is set to the local terminal
- If JRE is included in the self installer file

The Packager installation dialogs run in a character-oriented VT100 mode, if any of these prerequisites is not met on that computer.

# Packager Installation

To create software packages, you need the Packager installed on a local system. This system is called packaging system.

**To install the Packager on the packaging system**

1. Log on to the packaging computer as user root.
2. Mount a DVD/CD-ROM drive to enable access to the installation media that contains the Packager. Enter the mount command for DVD/CD for your individual operating system. (See your individual operating system documentation for the use of the mount command.)
3. Change to the relevant Packager directory on the installation media.
4. Specify the target location for the Packager's runtime components, if other than default is desired.

   By default, the Packager installation program will use the /tmp (temporary) directory to extract its runtime components. In that directory, it requires about 5 MB of free disk space.

   If you want the runtime components to be extracted at a location other than the default directory, set now the TMPDIR variable to the desired path, as follows.
   `$TMPDIR=`*`newpackagerpath`*

   The extracted components are automatically removed after the Packager installation has successfully completed.
5. Execute the Packager installation program install-sm-packager.sh (see Packager Installation Program (see page 19)).
6. Follow the instructions in the installation wizard to complete the installation.

   **Note:** In the case of character-oriented VT100 installation dialogs, use the Tab key to select a navigation button in the bottom line, and the carriage return (CR) key to check or clear a selection button.

When the Packager installation has successfully completed, all runtime components of the install program are automatically removed from the temporary directory (/tmp or as specified by TMPDIR).

# Packager Installation Program

The Packager installation program install-sm-packager.sh is a self-installing PIF product. You can specify options when executing install-sm-packager.

The command has the following format:

`sh install-sm-packager.sh [-t tracefile] [-p logfile] [-r responsefile] [-s] [-F] [-x]`

**-t *tracefile***

Traces the installation in the specified trace file.

**-p *logfile***

Logs the installation in the specified log file.

**-r *responsefile***

Installs the product using the specified response file.

**-s**

Installs the product in unattended (silent) mode using default parameter values.

**-F**

Forces the product installation. Backup errors are ignored.

**-x**

Only extracts the product to the /tmp directory. The file name used for the extracted product is internally defined and echoed to the command line.

**-V *vt100 mode***

Installs the product in a character-oriented VT100 mode.

**-a *ask mode***

Installs the product in the Ask mode.

**-o *overwrite (response file)***

Overwrites existing response during response file generation.

# Packager and Installer Installation Folders

CA Technologies installer standard allows customers to define their individual product installation folder for the Packager.

The home directory of the Installer can be defined through the parameter $INSTALLER_DIR. If this parameter is set prior to the first installation of a PIF product, the Installer will choose the value of this parameter as its installation folder.

If the parameter $INSTALLER_DIR is not set, but other CA products have been installed into the Shared components folder and have exported the referring parameter $CASHCOMP, the Installer uses $CASHCOMP/installer as its home directory.

The Installer does not set $CASHCOMP for global use, that is, the user explicitly needs to set this parameter before installing the first product.

To allow installations of products containing previous Installer versions, following additional files or links are necessary on the system:

**/usr/bin/lsm**

(Script) Anchor to the command line interface (lsm), used by previous installations

**/opt/CA/installer**

(Link) Redirects previous installations to the new installer home directory

If a previous Installer version is being upgraded, the existing Installer home directory is used. There is no migration to new folders.

# Packager Removal

**To remove the Packager from the packaging system**

1. Sign on as root or using an ID with root rights. For example issue the following command.
   ```
   su - root
   ```
2. Issue the following lsm command to remove the Packager.
   ```
   lsm -e SMPackager
   ```

# Chapter 3: Using Packager and Installer for Linux and UNIX

This chapter provides an overview of how to use the features and functions of the Packager and the Installer. For a better understand how to specify these features, see Building a Sample PIF Product (see page 83). The sample PIF product contains many of the features.

The detailed descriptions of all keywords and properties that are used to describe the features in the prototype file can be found in the Keywords and Properties (see page 111) section.

This section contains the following topics:

PIF Products (see page 21)
Delta Products (see page 22)
Patch Products (see page 22)
Virtual Packages and Alias Names (see page 24)
Components of PIF Products (see page 25)
Files, Directories, and Symbolic Links in PIF Products (see page 27)
Disk Space Reservation for Files in PIF Products (see page 28)
Scripts within the Scope of Product Installation and Removal (see page 29)
Dependencies of PIF Products (see page 31)
Product Parameters (see page 32)
Templates for PIF Products (see page 39)
Installation Dialogs (see page 40)
Installation Dialog Editor (see page 50)
Support of Various Packaging Formats (see page 61)
Installer Functions Overview (see page 65)
No Dependency on the libncurses Library (see page 66)
Self-installing PIF Products (see page 66)
Integration of JRE in Self-installing PIF Products (see page 67)
Product Family Concept (see page 68)
Migration of Non-PIF Products (see page 68)
Installer Compatibility Considerations (see page 69)

## PIF Products

The result of the packaging process using the Packager for Linux and UNIX is called a PIF product.

On the Packager GUI, several tools and wizards support you to create new products or copy and modify existing products.

You can also easily create or modify PIF products from the Software Management command line interface (CLI). Once you have created a prototype file, you can modify it using an editor, or duplicate the prototype file, modify the duplicate, and run the pifmk command to build a new PIF product from the modified prototype file.

PIF products can be created for and installed on all Linux and UNIX operating environments that are supported by the PIF SDK.

# Delta Products

A PIF delta product is a small installable unit in the PIF packaging format, which contains only the differences between two versions of the same PIF product. Instead of installing a complete new PIF product on all target computers, you can install only the small delta product on all target computers where the previous version of the PIF product is already installed. For example, a PIF delta product contains all changed and new files of the new PIF product and additional uninstall instructions for files in the previous product version that are no longer used in the new PIF product.

A PIF delta product is a standard PIF product.

You create PIF delta products using the pifdelta command, as shown in the following example.

### Example: Create a delta product from two product versions

The following command compares the versions of the PIF product MERCHANT located in the subdirectories called 2100 and 3000 of the product installation directory and creates the PIF delta product containing only the differences between the product versions 2.1.0.0 and 3.0.0.0 in the subdirectory called deltas.

```
pifdelta -p $PATHmerchant\2100\MERCHANT.Any.@pif
    -v $PATHmerchant\3000\MERCHANT.Any.@pif
    -d $PATHmerchant\deltas\MERCHANT.Any.2.1.0.0-3.0.0.0.@pif
```

# Patch Products

A PIF patch product is a standard PIF product containing patches to update a specific PIF product. Patch products let you fix applications without republishing or distributing the entire product. The patches can be applied using software delivery functions, or they can be applied manually.

Most of the existing functions for defining PIF products, such as dependency check and installation dialogs, can be used to define the patches. When the PIF patch product is installed, all replaced or removed files of the product are saved in the product's home directory. Changes are reversed if the patch installation is rolled back. Updating or removing a product removes all installed patches for the specific product.

The Packager provides a wizard to create PIF patch products. To launch the Create a new patch product wizard, right-click the product to be updated and select New patch from the context menu.

You can also create PIF patch products using the pifpatch command. The PIF product the patch should be applied to is specified using the #patchproduct keyword in the prototype file. The pifpatch command generates a patch automatically, based on the PIF delta product creation mechanism, by comparing two sets of packages and querying the dependency structure of the main patch product.

When you are going to apply cumulative patches, you can specify in the current patch product a list of patch packages that are substituted by the current patch product. In the prototype file, you specify the list of substituted patch packages using the #replacepatches keyword.

With the lsm command you can list all patch packages or all patches applied for a specific product.

## Main Patch Package

Consider the following scenario:

At the time the original PIF product, consisting of several packages, was installed, the customer decided to configure the installation of a certain subset of packages, that is, not all packages of the original product are installed at the customer side.

When a PIF patch product has to be installed in this scenario, it has to evaluate automatically which of its patch packages must be applied.

Therefore you must define a "main patch product" which serves as anchor for the individual patch packages, with appropriate dependencies to the individual patch packages.

The main patch package can be compiled as self-installing package, which ensures that the Installer will always be updated to the required version. Required patch packages are installed only if the original package for the patch exists. If an original package has not been installed on the target computer, the related patch is also not installed and no error is returned. The application of this patch package is ignored.

If the main patch package is removed (rolled back) all required individual patch packages are also removed (rolled back).

## Conventions for Patch Packages

The following conventions exist for patch packages

- The name of a patch package must be different from the name of the package it is going to patch. The issue number or the support fix number may be used within the patch package name for easy recognition.
- Configuration dialogs and removal dialogs are not supported in a patch package.
- Versioning for patch packages is not supported. A patch is applied only once. It is not possible to upgrade the same patch. In this case, the patch must be replaced by another patch.
- A patch package is applied only to a single instance of an original package. If several instances of the original package are installed, the patch must be applied to each instance separately.

# Virtual Packages and Alias Names

Within a PIF product, for example, PROD1, you can specify alias names of other PIF or RPM packages (virtual packages), whose functionality or services are covered by the PROD1 product. The alias names can include, for example, the previous name of PROD1, if it has been renamed recently, or the names of packages that were merged in the product PROD1.

The use of alias names separates names of physical packages and names of logical services. This allows renaming packages without losing dependency information.

During installation of PROD1 the dependency check evaluates the list of alias names and resolves dependencies to these packages. So the Installer prevents a service or package being removed from the system if any product still requires this service or the service the package provides.

You specify alias names in the @PRODUCT section of the prototype file using the #vname keyword, or on the Advanced tab of the Packager GUI.

**Example:**

Three different product packages provide the internationalization service: "CAWIN", "ca-cs-cawin", and "CAI18N". When you define the alias name "CAWIN" inside each of the three packages, none of those packages will be removed from the system as long as there are products that require the "CAWIN" internationalization service.

# Components of PIF Products

A PIF product is structured into one or more components. According to the number of components, the prototype file for the PIF product consists of one or more component sections. The contents of the components are defined in the component sections of the prototype file.

Depending on their content and purpose, components can be of the following types:

**Standard component**

Contains directories, product files, symbolic links, and disk space reservations for files that are created after installation during product usage.

**Platform-specific component**

Contains directories, product files, and symbolic links that are installed only in the operating environment defined in the component.

**Language-specific component**

Contains directories, product files, and symbolic links that are installed only if the language specified in the component matches the language defined on the target computer.

**Pre-install component**

Bundles all files to be installed before installation of the main PIF product.

**Configuration component**

Bundles all files required for configuration dialogs and product removal dialogs.

## Standard Components

Standard components contain all definitions of resources for the PIF product, such as directories, product files, symbolic links, and reservations of disk space for files that are not installed during product installation. These items are processed during the main part of the product installation. Standard components are installed on the target computer.

## Platform-Specific Components

Typically, a component of a PIF product can be installed in all UNIX and Linux operating environments. If you want a component to be installed only on a specific operating environment, you define the component as platform-specific. You specify the desired operating environment through the #sys: keyword in the prototype file. The component is installed only on that target computer whose operating environment matches the specified platform identification (PlatformID parameter). If no platform-specific component is found during installation, the default component with the PlatformID = Any is installed on the target computer.

A suitable example for the use of platform-specific components is the installation of a shared library. Shared library names differ in different UNIX operating environments. On Sun Solaris and Linux systems, the names of shared libraries end with .so; on HP-UX, they end with .sl; and on IBM-AIX, they end with .o or .a. To cover these different naming conventions, for example, for HP-UX and Solaris systems in your network, add two platform-specific components to your PIF product. The first component stores the HP-UX library with the extension .sl, and the second component stores the Solaris library with the extension .so.

The Packager distinguishes between products that can be installed on any UNIX and Linux operating environment (PlatformID = Any) and products for a specified operating environment. If a product can be installed on any operating environment, all platform-specific components are added to the PIF product. If the product can be installed only on a specified operating environment, all components that do not match the PlatformID parameter are removed from the PIF product.

## Language-Specific Components

Language-specific components enable multilingual installations. Language-specific components contain files that are installed on the target computer only if the language defined in the component matches the language specified by the $LANG parameter on the target computer. Typically, language-specific components contain language resource files, help files, manual pages, and graphic files that contain language-specific texts. You specify a language-specific component through the #locale keyword in the component section of the prototype file.

Language-specific components can be stored inside the main PIF product or in an external language-specific PIF file. The external language-specific PIF file must be stored in the same folder as the main PIF product.

To translate language-specific components, extract them using the pifextract command. The translated language-specific component can be added to the PIF product as a new language-specific component.

Language-specific components are not platform-specific.

The component with the default product language is installed anytime. Additionally, the language-specific component that matches the target computer language ($LANG) is installed.

## Pre-install Components

Preinstall components contain all files used prior to the actual installation of the PIF product, that is, before installation dialogs and scripts are executed. The preinstall components are extracted to a temporary folder, specified through the parameter PIF_PREINSTALL_DIR.

Typically, preinstall components contain items such as the following:

- Graphic files used by Java dialogs
- Scripts used inside installation dialogs, verification scripts, and dynamic dialog detection scripts
- A script to migrate non-PIF products
- Product and component prenatally scripts
- Text resource files used by install scripts

You may have several pre-install components with the same name, but different languages.

If the locale of the preinstall component does not match the system locale, an external language-specific PIF product will be used instead, if you have provided such a product.

## Optional Components

For all types of components you can, if appropriate, specify that the component should not be installed with the product in any case, but only on demand (that is, by selecting this component from an installation dialog or depending on a parameter value in a response file.) Components that are installed only on demand are called optional components. For example, a set of documentation can be defined as an optional component.

Optional components can be selected or unselected with the use of a PIF parameter. If the PIF parameter is set to 1, the component is installed; however, if it is set to 0, the component is not installed. This parameter is assigned to the PIF package through the GUI with “components/advanced/installation behavior” or manually with the keyword “#cinstall:”. You can add a check box to provide the parameter with the according value. You must add this check box to a dialog.

# Files, Directories, and Symbolic Links in PIF Products

A PIF product contains files, directories, and symbolic links. Normally, all these items are installed relative to the product installation path. To install them at a specific location on the target computer, specify their fully qualified destination path names.

The Packager and Installer support the following file types:

**Standard file**

Standard files are unchangeable product files that are mandatory installed on the target computer. The Installer checks standard files for consistency. In case of reinstall, standard files are replaced during reinstallation.

**Configuration file**

In configuration files, all settings are preserved, that is, configuration files are not updated during an update installation. The file size of configuration files changes often and does not reflect the product consistency. Therefore, the PIF product consistency check ignores changed configuration files. Adding new parameters to a configuration file must be done inside a pre- or post-install script. Configuration files are removed when the complete PIF product is removed, unless removal is suppressed by the lsm option -R or the internal parameter PIF_RETAIN_CONFIGURATION. In case of reinstall, configuration files are not modified by the Installer.

**Temporary file**

Temporary files are copied to the target computer at the time of the product installation, and are removed from the target computer when the product installation has finished. In case of reinstall, the Installer does not check temporary files.

**Permanent file**

A permanent file is not deleted from the system if a software product is removed. It remains on the system after the product removal. If the permanent file exists before the product installation and the modification date of the existing file is newer than the file that is shipped with the product, the permanent file remains unchanged on the system.

Directories (#dir) and symbolic links (#slink) are assigned to users and groups. These users and groups must exist prior to the product installation.

All directories of a path must have their own entry in the PIF product; for example, /usr/local/bin/ requires three entries in the product for the following directories: /usr, /usr/local, and /usr/local/bin.

# Disk Space Reservation for Files in PIF Products

For files that are not installed during product installation (virtual files), you can reserve disk space using the #reservedspace: keyword in the prototype file when creating the product. The required disk space of the virtual files will then be considered by the disk space check program.

# Scripts within the Scope of Product Installation and Removal

You can write scripts and executables for the Installer to run prior to or after the installation or removal of a PIF product on the target computer.

By default, Bourne-compatible shell scripts are assumed for execution. You can run internal procedures as well as nonstandard shell scripts executed by another interpreter, and you can use parameters to customize and control the scripts.

## Usage of Internal Procedures

Internal procedures are script files used to configure and interchange information with the product installation process. Internal procedures are part of the product; each of the script files must be defined in the file list of a component. Execute rights must have been specified for these script files.

You can create a new script file from the Packager GUI using the internal editor (which opens when you choose the Add New Files function.) Saving the new script file automatically adds its name to the file list of the component.

An existing script file is added to the file list of a component by using the Insert files function for the component of the PIF product.

If you want to add the internal procedure to a specific component, the procedure must be defined in that specific component (through the GUI or as an entry in the appropriate component section of the prototype file.) If you want to add the procedure to the product in general, the internal procedure can be defined for the product (through the GUI or as an entry in the product section in the prototype file.)

You can specify the following properties for internal procedures:

**Execution Time**

Execution time of the script. Possible values are as follows:

- pre installation—The script is executed before the package installation.
- post installation—The script is executed after the package installation.
- pre remove—The script is executed before the package is removed. If the script exits with a nonzero exit value, the component is not removed from the system, but the uninstallation process continues.
- post remove—The script is executed after the package has been removed.
- pre start—The script is executed before the product installation and any installation dialog starts.

- pre PIF install—A special script that is executed only the first time a product is installed on the target computer. Only this script is able to modify the product installation path.
- post interview—The script is executed after the interview phase (of the installation dialogs) has completed.
- post check—The script runs after the installer checks have been executed.

**Options**

If your script allows using command switches to configure the script execution, you can specify these switches when adding the internal procedure to the product. These command switches are applied to the script during script execution.

**Interpreter**

Optionally, you can specify an interpreter if the default interpreter, the shell, is not used.

## Usage of Non-shell Scripts

You can run scripts that are executed by another interpreter than the standard shell, for example, IPS/DMS scripts and Perl scripts. In these cases, the name of the required interpreter must be specified as an Installer property.

## Usage of Scripts with Parameters

You can use parameters with your script to execute during product installation, for example, to set values for the script or to control script execution. The parameter values you pass to the script may determine which part of the script to execute. Parameter values can be passed to the script as shell parameters, script arguments, or a response file.

To update parameters in shell scripts, use a response file to pass the parameter values to the script. The response file must contain all modified parameters. The Installer evaluates the response file and uses the modified parameter values during the remainder of the installation process. The name of the corresponding response file is referred to in the script through the internal parameter $PIF_RESPONSE_FILE. Updated parameters must be appended to this response file.

Changes made to shell parameters are ignored.

**Important!** When a product is updated, the value of the installation path parameter cannot be changed. The pre-install script can change only the component install flag.

## Scripts Timer Modification

If you have long running scripts that may exceed the default PIF timer of ten minutes, you can define a product timer that fits the requirements of the installation script (using the #timer keyword in the prototype file).

For example, a PIF product depends on a compressed non-PIF product that is installed using uncompress, and the installation time of the non-PIF product exceeds the default time limit of ten minutes. In this case, it must be ensured that the non-PIF product is completely installed prior to the main PIF product.

**Important!** Setting timer values too high may circumvent timeout mechanisms for unattended installations.

## Progress of Scripts

The progress of script execution can be displayed in the progress area of Progress dialogs, using output of the script. This feature is useful to show the progress of long running scripts during product installation. To show the output of the script, use the dialog element Progress Area. In the prototype file, this feature is supported through the #progressarea keyword.

In the Progress dialog the name of the script currently running is shown in the Action field. The Product progress is on hold when a script is executed.

This feature is not available for RPM packages.

# Dependencies of PIF Products

The installation of a PIF product can depend on the presence or absence of another PIF product or product version on the target computer. Such a dependency can exist between the PIF product to install and another required product in the PIF, PKG, or RPM packaging format.

Dependencies of products can be of the following types:

**Mandatory**

A product prod1 is mandatory dependent on product prod2, when prod2 must be installed on the target computer before product prod1 is installed. If prod2 is not installed, the Installer automatically searches for it, using a specific algorithm, and installs it if found. If the Installer cannot find prod2, the installation is aborted. Prod2 is called required product.

**Incompatible**

A product prod1 is incompatible dependent on product prod2, when prod2 must not be installed on the target computer when product prod1 is installed. If prod2 is installed, the installation of prod1 is aborted.

**Optional**

A product prod1 is optional dependent on a product prod2, when prod2 is not mandatory required before prod1 is installed. Prod2 is installed on the target computer only on user demand. If a previous version of prod2 already exists on the target computer, it is updated to the new version.

Dependencies of a product are defined in the prototype file, using the keywords #pdep (mandatory and optional) or #dis (incompatible), or through the Packager GUI.

When installing a product, the Installer automatically resolves all dependencies to other PIF products defined in the product and its components, if needed over several layers. The Installer ensures that all required products are installed on the target computer before the actual product is installed. You must only ensure that all required products are provided on the distribution media.

When a product is being removed, the Installer automatically removes required products, over several layers if needed, provided that other products do not require these products. The Installer checks only those required products for removal that were delivered and installed with the product.

Incompatible products are not automatically removed.

# Product Parameters

Product parameters configure the installation of the PIF product. They exchange information between scripts and the install process. The Installer resolves parameters at installation time of the PIF product. Using parameters, you can define installation paths and include or exclude components from the installation process. Product-specific parameters can also be passed to the installation. Parameters are uniquely specified by their names. Parameter names are always lead by the dollar sign ($).

Parameters are defined by the parameter name, type, and value. The value of a parameter can be an absolute (static) value, the result of a command execution, the outcome of another parameter, or the combination of an absolute value and another parameter.

## Types of Parameters

We distinguish the following kinds of parameters, depending on their product installation impact and visibility within the PIF product:

**Private**

Specifies that the parameter is valid only for the product itself. Private parameters can be modified by product-specific installation scripts only.

**Published**

Specifies that the parameter is valid for the product and all its required products. In the required products the access is restricted to "read-only." Published parameters can be set in the main product and in subsequent PIF packages if the parameter has not been already set in a package at a higher level.

**Global**

Specifies that the parameter is valid for all product installations. Once set a global parameter cannot be changed anymore unless all products using this parameter are removed. A global parameter can be removed only if no other user references it.

**Important!** Take care when using parameters to configure the PIF product itself. In this case, you should use private parameters only, because these cannot be changed by other product installations.

## Ways to Provide Parameter Values

Values for PIF product parameters can be provided in different ways, as follows:

**As default parameter values**

The default value of a parameter is defined in the Default section of the prototype file. The default value ensures that a value always exists for that specific parameter.

**From the environment**

During product installation, the Installer examines the environment of the running shell. If a parameter value has been assigned on the shell, it will overwrite the default parameter value defined in the product.

**From a response file**

A response file, which is provided by the user to configure an unattended product installation, contains parameter value assignments that overwrite the default parameter values defined in the product.

**From a script**

You can define parameter value assignments in pre-install scripts (scripting parameters). These scripts insert the parameter values in the product when they are run during the product installation process.

## Types of Parameter Values

Parameter values used in PIF products can be of the following types:

**Static**

A static value is entered and directly assigned to the parameter.

**Combined**

The value assigned to the parameter is a combination of static values and previously defined parameters, for example:

```
#parameter: $installdir , /opt/CA/installpath ;
#parameter: $my_install_path , $installdir/myproduct ;
```

The end of a static parameter within a combined parameter is determined by one of the following characters:

```
.  ;  :  \t  \n  \\  \"  /  ,  $  (  )  -  {  }  white space
```

**Dynamic**

The value assigned to the parameter is the result of the execution of a command. Therefore, a command is provided instead of a static value, for example:

```
#parameter: $my_date , `date` ;
```

## Default Values for Parameters

Each parameter used in the PIF product has to be assigned a default value; otherwise, the product creation will fail. These default values are used for the following purposes:

- As the initial values when tasks on the installation dialogs are performed.
- For an unattended installation of the PIF product, if no response file is provided.

In the prototype file, you define default values for parameters in the @DEFAULT section using the keyword #parameter.

**Important!** When you create a PIF product using one of the standard product templates, default values are already assigned automatically to the parameters.

## Additional Parameter Descriptions

To provide a short description of an individual product parameter, you can add a comment as a parameter property.

You add this description

- In the Comment field of the Add/change parameter dialog (This description is added to a response file that you create with the lsm -a command.)
- As a property paramcomment of the #parameter keyword within the prototype file

## Important Notes on Parameters

- Global parameters defined in optional required products are added to the common set of parameters only if the optional required product has been selected.
- For global parameters set through the environment, the following applies:

  The Installer performs a check whether a global parameter previously unknown to the Installer is already defined in the environment. If this is the case, the Installer takes that value as a set value for the parameter, which automatically disables dialogs changing the value of this parameter.
- Parameter values that consist of other parameters or combination of parameters and absolute values are automatically evaluated when the user navigates between dialogs.
- The following characters are not allowed within parameter names:
  `. ; : \t \n \\ \" / , $ ( ) - { } white space`

## Example: Using Parameters to Control Access Rights

When defining files (#file) or directories (#dir) in the prototype file, you can use parameters to control the user, group, and file access rights, and assign appropriate read, write, or execute permissions. This can be useful if you want to install the PIF product for different users on different operating environments.

The following example shows the use of parameters for a #dir entry in the prototype file:

```
@DEFAULT:
#sys: Any ;
#parameter: $OWNER , root ;
#parameter: $GROUP , sys ;
#parameter: $ACCESS , drwxrwxrwx ;
@ENDDEFAULT:

@PRODUCT:
#phead: exampleOwnerParam , 1000 ;
#ppath: /tmp/exampleOwnerParam ;
#sys: Any ;
@ENDPROD:

@COMPONENT:
#chead: comp1 ;
#dir: mydir , $OWNER , $GROUP , $ACCESS ;
@ENDCOMP:
```

## Response Files for PIF Products

The response file for a PIF product contains records that assign values to parameters. On the target computer, you can create a response file interactively, using the Installer command lsm with the -a option. This command runs the installation dialogs and creates the response file using the values entered; the PIF product is not installed.

The parameter value assignments may have the following syntax formats:

***parameter=value* ;**

The value on the right side of the equation is directly assigned to $parameter (static assignment).

Example:

```
INSTALLDIR=/opt/CA/installpath ;
```

***parameter=`command`***

The command is executed, and the output of the command is assigned to $parameter (dynamic assignment). The command must be enclosed in backticks (`). (Note that these are not single-quotes.)

Example:

```
SYSTEMNAME=`uname -n`
```

Comment lines in a response file must begin with the hash sign (#).

If parameters have a description assigned in the Default section of the prototype file (#parameter keyword), this description is automatically added to the response file.

## Internal Parameters

Internal parameters are provided during installation of the PIF product. These parameters can also be used to present information within the installation dialogs. The prefix PIF_ is reserved for internal use and cannot be used for user-defined parameters.

The following internal parameters are used:

**PIF_BACKUP**

Controls backup creation prior to updating or reinstalling an existing product version.

**PIF_CHANGECONTROL_*dialogelement***

Controls whether a dialog element is grayed out if it is not used.

**PIF_CHANGEPARAM_*parameter***

Controls whether the specified parameter can be edited using dialog elements. If PIF_CHANGEPARAM_*parameter* has the value 1 (True), dialog elements that alter the specified parameter are enabled. If the value is 0 (False), the specified parameter is not accessible.

**PIF_CONTROL_NAME**

Specifies the name of the field that was active (had the focus) before a script was called. Based on this information the script can behave differently depending on the field from where it was called. This internal parameter is passed to any validation script within a PIF dialog, that is, field validation script, action script, dialog validation script, and dynamic dialog detection script.

**PIF_DIALOG_LANGUAGE**

Specifies the language to use for the product installation. The value of this parameter is a 2 or 3 bytes language identifier, for example, ENU.

**PIF_DIALOG_MODE**

Indicates the installation dialog type. Possible values are:

JAVA, VT100, and NONE.

**PIF_DIALOG_NAME**

Specifies the name of the dialog corresponding to the selected button.

**PIF_INSTALL_JOB**

Identifies the install job. Possible values are:

INSTALL, REINSTALL, UPDATE, and REMOVE.

**PIF_INSTALL_MODE**

Specifies the install mode. Possible values are:

ATTENDED, UNATTENDED, and ASK.

**PIF_INSTALL_PATH**

Specifies the installation path of the PIF package.

**PIF_LOCALE**

Indicates the locale where the PIF package is to be installed.

**PIF_LOG_FILE**

Specifies the name of a log file.

**PIF_MAINPRODUCT_DIR**

Specifies the installation path of the main package.

**PIF_PREINSTALL_DIR**

Specifies the name of the temporary folder where the pre-installation components are extracted.

**PIF_PRODUCT_NAME**

Specifies the name of the PIF package.

**PIF_PRODUCT_PLATFORM**

Indicates the operating environment of the PIF package.

**PIF_PRODUCT_VERSION**

Specifies the version of the PIF package.

**PIF_RESPONSE_FILE**

Specifies the name of the response file used to provide parameter values.

**PIF_RETAIN_CONFIGURATION**

Controls the removal of configuration files.

**PIF_SUPPLIER**

Specifies the name of the vendor of the PIF package.

**PIF_INSTANCE_NAME**

Specifies the name of a new instance. This parameter is used during installation.

**PIF_CREATION_DATE**

Specifies the date of the PIF creation. The format is DD/MM/YY.

**PIF_CREATION_TIME**

Specifies the time of the PIF creation. The format: hh:mm:ss.

## How the Installer Resolves Parameters

Parameters used in a PIF product are resolved by the Installer during installation of the product on the target computer. Following are the steps the Installer performs, depending on the type of installation.

**New installation**

1. Reads default parameter values defined in the PIF product.
2. Reads parameter values from a response file.

3. Reads parameter values from an installation dialog (installation path can be modified).
4. Reads parameter values from the existing environment (unattended mode).
5. Reads parameter values changed after execution of the product-specific pre-install script.

**Reinstallation**

Reads parameter values from the previous installation (installation path cannot be updated).

**Update installation**

1. Reads default parameter values defined in the PIF product.
2. Reads parameter values from a response file.
3. Reads parameter values from the previous installation.
4. Reads parameter values from an installation dialog (installation path cannot be updated).
5. Reads parameter values changed after execution of the product-specific pre-install script.

# Templates for PIF Products

Product templates unify installations of PIF products and define a PIF product in detail. Product templates are standard prototype files that contain a set of predefined parameters. These parameters are replaced during generation of the PIF product. The pifproto command supports template management.

The following standard templates for PIF products are provided through the Packager installation in the $SDPcker/templates folder:

**standard.Any.@enu**

Creates a package with a set of predefined basic installation dialogs. This template is used as the default template when no specific template has been chosen.

**advanced.Any.@enu**

Creates a package with an enhanced set of predefined installation dialogs.

**unattended.Any.@enu**

Creates a package for unattended installation without any installation dialog.

**multi_language.Any.@enu**

Creates a package with specific dialogs that let you choose from multiple languages for the installation.

**multi_instance.Any.@enu**

Creates a package with multiple instance support that allows the installation of the multiple instances of the package. The package must be marked as "multi installable" (#multiinst:1).

In the $SDPcker/templates directory, you can copy and change the standard templates to define your own company-specific installation of PIF products. The Packager recognizes each new file in this directory.

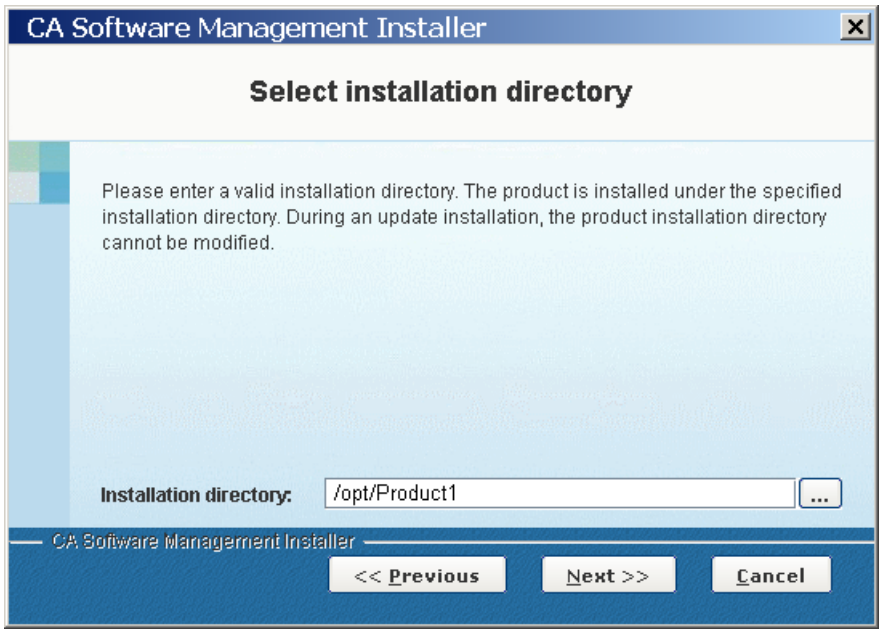
# Installation Dialogs

Installation dialogs allow user-friendly and enhanced setups on the target computers. Installation dialogs serve to retrieve parameters and configure the installation of the PIF product on the target computer. The Installer supports installation dialogs for all types of target computers.

When the target computer has the Java Runtime Environment (JRE) 1.4 installed, and the monitor is graphics-enabled, the installation dialogs appear in the graphical mode (Java GUI). If these prerequisites are not met on the target computer, the installation dialogs appear in a VT100 compatible mode.

You can include a JRE in a self-installing PIF product, that will then install and use the included JRE to run the Java wizard of the Installer on the target computer.

The following graphic shows a sample installation dialog in graphical mode.




In the VT100 compatible mode the same dialog displays as follows:

```
+------------------------------------------------------------------------------+
|  Select installation directory                                               |
+------------------------------------------------------------------------------+

Please enter a valid installation directory. The product is installed under
the specified installation directory. During an update installation the
product installation directory cannot be modified.

Installation directory: /opt/Product1









[ << Previous ]  [ Next >> ]  [ Cancel ]
```

## Defining Installation Dialogs

In the prototype file, you define the dialog properties, buttons, and texts in the Dialog and Resource sections, using appropriate keywords.

Each installation dialog is identified by a unique name, which lets you navigate between dialogs and define sequences for dialogs. Scripts can be specified for a dialog to verify every single value or all values entered before the subsequent dialog is started or to determine the name of the next dialog to start.

Installation dialogs can contain different dialog elements such as text input fields, text output areas, installation directory input fields, check boxes, navigation buttons, and so on. A dialog element can be determined that has the focus when the installation dialog is started, for example, the Next button.

All text messages presented in the installation dialogs are stored in separate sections in the prototype file, so they can be localized to provide different languages within the same PIF product.

**Note:** The VT100 compatible mode can be forced by using the Installer (lsm command) option -V.

## Standard Installation Dialogs

When you create a PIF product using one of the standard templates, except unattended.Any.@enu, the Packager automatically adds a set of standard installation dialogs to the PIF product.

On the Packager GUI, these dialogs are listed on the Dialogs tab in the subproject pane. In the prototype file, each of these installation dialogs is described in an individual Dialog section.

The standard installation dialogs contain a number of preset dialog elements which can be changed, enhanced, or copied using the Packager's installation dialog editor.

## Task-oriented Installation Dialogs

The Installer distinguishes between the dialog for a new installation (installation dialog) and the dialog for an update installation, reinstall, or removal of a PIF product (update dialog).

To start individual dialogs for update, reinstall, and removal of a product, the initial update dialog should present a menu with the different installation modes available to the user.

In the prototype file, the keyword #dlgpreinit specifies the installation dialog that is called first when you are initially installing a PIF product. The keyword #dlgpreupd specifies the installation dialog that is called first when you are going to update, reinstall, or remove a PIF product.

On the Packager GUI, use the Dialog properties dialog to determine whether the created dialog is for a new or an update installation.

You must define at least one Cancel dialog for each navigation button that has the Cancel action associated.

## Dynamically Detected Dialogs

To determine the next installation dialog to be shown, you can specify a shell script, which provides the name of the next installation dialog, instead of specifying a fix dialog name. Using a shell script instead of a fix dialog name enables sequences of dynamically specified dialogs.

The shell script must print the name of the next installation dialog to standard output. The name of the shell script is entered as a dialog property. The script is linked to the navigation button control.

## Dialog Validation Scripts

To validate the combination of values entered in an installation dialog, you can provide a dialog validation script. The validation script file must be added to a pre-install component. Within a dialog you can validate user data from the complete dialog or from specific input fields.

**Validation of all user data entered in the dialog**

The dialog verification script is executed when you click Next on the installation dialog. All parameter values entered in the installation dialog are then passed through the shell environment. The script must return zero (0) if the verification has succeeded. In case of an error, the script must return the number of a matching error text in the Resource section of the prototype file. The referred error text number must be in the range of 1 – 255. If a corresponding error text is missing, the text number is displayed in the error message.

The following example checks two entered values, $LOCALE and $DATE. If an English locale is entered, the date field may not contain any date (or in German date representation).

```
#!/bin/sh

# If one of the parameters $LOCALE or $DATE is not set,
# error message 1 is printed
[ ! "$LOCALE" -o ! "$DATE" ] && exit 1

# If an English locale has been entered, the date field
# must not contain any dot
[ "$LOCALE" = "en_US" -a `echo "$DATE" | grep "." 2>/dev/null 1>&2; echo $?` -eq
0 ] && exit 2

# exit without error message
exit 0
```

**Validation of user data entered in specific input fields**

You can specify validation scripts for text fields (#textfield), installation directories (#instdir), and password fields (#passwordfield). These validation scripts are executed when the user leaves the input field. These scripts are also called, when the Next button is clicked before the dialog verification script is invoked. The entered parameter value is passed to the script as a shell parameter. The script must return zero (0) if the verification has succeeded. In case of an error, the script must return the number of a matching error text in the Resource section of the prototype file. The referred error text number must be in the range of 1 – 255. If a corresponding error text is missing, the text number is displayed in the error message.

The following example checks if a directory name is fully qualified (parameter $INSTDIR):

```
#!/bin/sh

# If the parameters $INSTDIR is not set, error message 1 is printed
[ ! "$INSTDIR" ] && exit 1

# If the parameters $INSTDIR does not start with a slash,
# error message 2 is printed
[ `expr "$INSTDIR" : '\(.\)'` != "/" ] && exit 2

# exit without error message
exit 0
```

**Providing logging information from the validation script**

Logging information can be provided by using the parameter PIF_LOG_FILE in the dialog validation scripts, as follows:

```
if [ $VERBOSE -eq 1 -a -w "$PIF_LOG_FILE" ]
then
  echo "LOG INFORMATION TEXT" >> "$PIF_LOG_FILE"
fi
```

## Action Scripts

Action scripts can be defined for the use with check boxes, password fields, installation directories, file selection fields, combo boxes, and radio buttons. These scripts let you change attributes and contents of dialog elements (fields) in the installation dialog immediately, for example, to activate or deactivate fields.

You can find examples for action scripts under the directory $SDPcker/examples/advanced/actionscript, on the system where the Packager is installed.

## Splash Screen

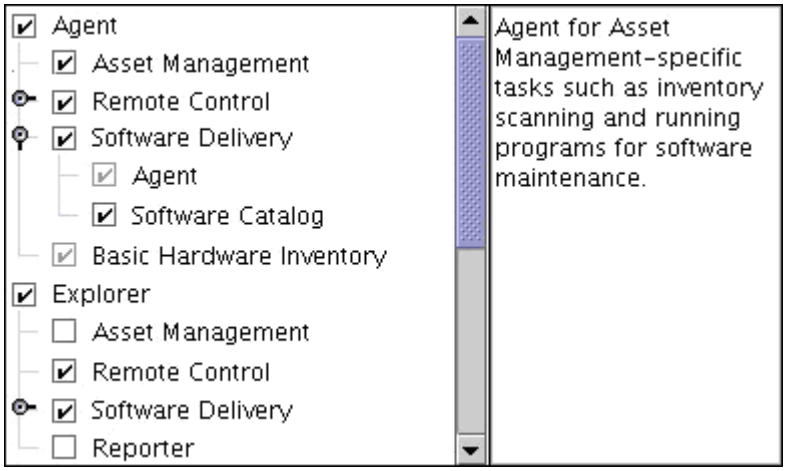
A splash screen is typically presented at the beginning of an installation and pops up a graphic informing about the product, its version, and copyrights. The splash screen is shown in interactive installations prior to any dialog of the interview phase. When the interview is started, the splash screen disappears automatically.

You can specify a graphic in the GIF format, which is used as background graphic in the splash screen, and an optional text string that is displayed in the left bottom edge of the graphic, 10 pixels from each border.

In the prototype file, you use the keyword #splashscreen to define the splash screen; the text string is specified in the Resource section and can be localized. You can define line breaks in the splash screen text by using the tag <br> within the text string.

## Check Box Trees

A check box tree lists product components in a hierarchical structure. A sample check box tree is shown in the following screenshot:




When a component is selected, all sub-components are automatically selected. Required components are also selected, if the referring component is selected.

The component list can be parsed using a scroll bar.

Optionally you can define a text area that shows information about the selected component.

Two pairs of keywords let you define a check box tree control:

**#tree_begin, #tree_end**

A tree contains a list of check boxes.

**#scrollarea_begin, #scrollarea_end**

A scroll area contains a list of check box trees or check boxes.

### Example: Add and Modify a Check Box Tree

This example provides the information on how to add a check box tree to a selected installation dialog and then modify it.

This example covers the following topics:

- Add a check box tree
- Add a node to the existing check box tree
- Remove a node from the existing check box tree

**To add a check box tree**

1. Select a package from the Project pane.

   The general package options appear.

2. Select the appropriate dialog from the Dialog tab (subproject pane).

3. Drag and drop the check box tree icon from the toolbox to the desired location in the workbox.

4. Click the check box tree icon in the workbox.

   An empty area opens where you create and edit your check box tree.

5. Right-click in this area and select Add Checkbox from the context menu.

   The Input dialog appears.

6. Enter the name of the check box and click OK.

   The check box appears in the editing area; the name of the check box appears as descriptive text next to the check box.

**To add a node to the check box tree**

1. Right-click the check box and select Add Child Checkbox from the context menu.

   The Input dialog appears.

2. Enter the name of the child check box and click OK.

   The check box appears below the selected check box at one level down.

**To remove a node from the check box tree**

1. Right-click the node in the check box tree.

   The context menu options appear.

2. Select Delete from the context menu.

   The selected node is deleted from the tree.

## Font Specification for Installation Dialogs

The PIF SDK lets you adapt font sizes and font representation in Java installation dialogs for all dialog elements. The following font properties can be modified:

- Size
- Bold

- Italic
- Color (RGB format)
- Java logical fonts: Serif, SansSerif, Monospaced, Dialog, and DialogInput

  Logical fonts are mapped by the Java runtime system to the best fitting physical font.

The font properties are stored separate from the dialog elements. So, once they are defined, you can assign the font properties to any dialog element of an installation dialog.

The fonts and their properties are specified using the Font Chooser dialog or directly in the @FONTLIST section in the prototype file.

All relevant dialog elements (that is, the keywords) contain the optional *fontname* property, which refers to the font specifications in the @FONTLIST section.

## Example: Modify the Font of a Dialog Element

This example provides the information on how you can modify the font of a selected dialog element. Each dialog element that displays text includes the Font name drop-down list in its properties dialog.

**To modify the font of a dialog element**

1. Select an element in the dialog.

   The properties pane appears for the selected dialog element.

2. Click the browse button (...) next to the Font name drop-down list.

   The Font Chooser dialog appears.

3. Enter the values for the following parameters:

   **Font name**

   Indicates the name for the font you are specifying.

   **Font type, Font size**

   Specifies font type and size.

   **Effects**

   Specifies Bold, Italic, or both.

   **Color**

   Specifies the font color, available from the drop-down list.

   **Note:** You can see the preview of the specified font in the Preview area.

4. Click Add to add the new font.

   The new font is added for the selected dialog element.

**Note:** You can also choose a font from the Font name drop-down list if you do not want to define a new font for the dialog element.

## Specify the Input Order of a Dialog

You can specify the order in which input fields of an installation dialog should be processed during the installation of the PIF package. When this installation dialog is called during a later installation of the PIF package, the input fields in this dialog are processed in the specified order.

The input order is specified by using the Input order check box in the toolbar of the installation dialog editor or directly in the @Dialog section in the prototype file. The #inputorder dialog property stores an ordered list of (input) dialog elements.

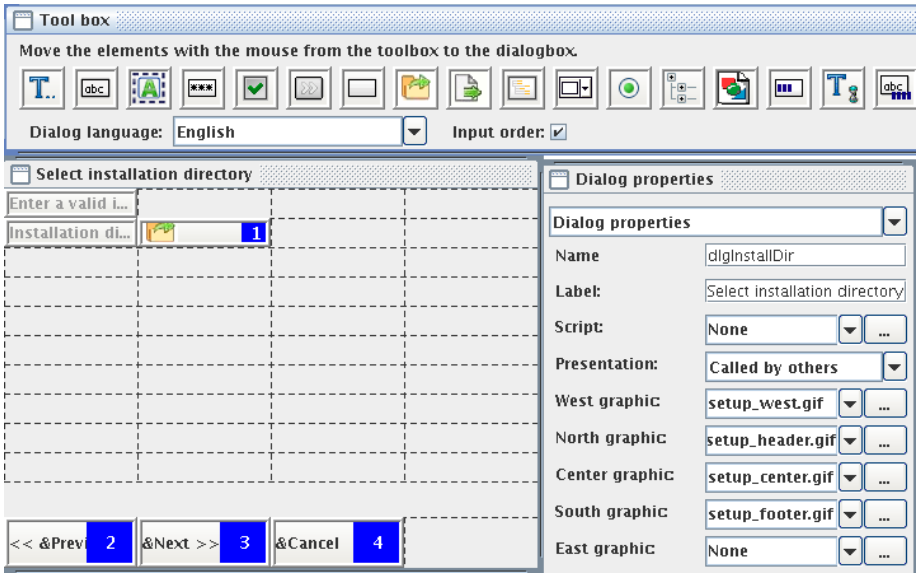
**To specify the input order of a dialog**

1. Select the dialog in which you want to specify the input order for the dialog elements in the Dialog tab (subproject pane).

   The selected dialog appears in the right pane.

2. Select the Input order check box in the installation dialog editor's toolbox.

   Blue boxes appear for the input fields in the dialog creation area, as shown in the following illustration:




3. Click the input fields (blue boxes) one-by-one to specify the input sequence.

   When this installation dialog is called during a later installation of the PIF package, the input fields in this dialog are processed in the specified order.

# Installation Dialog Editor

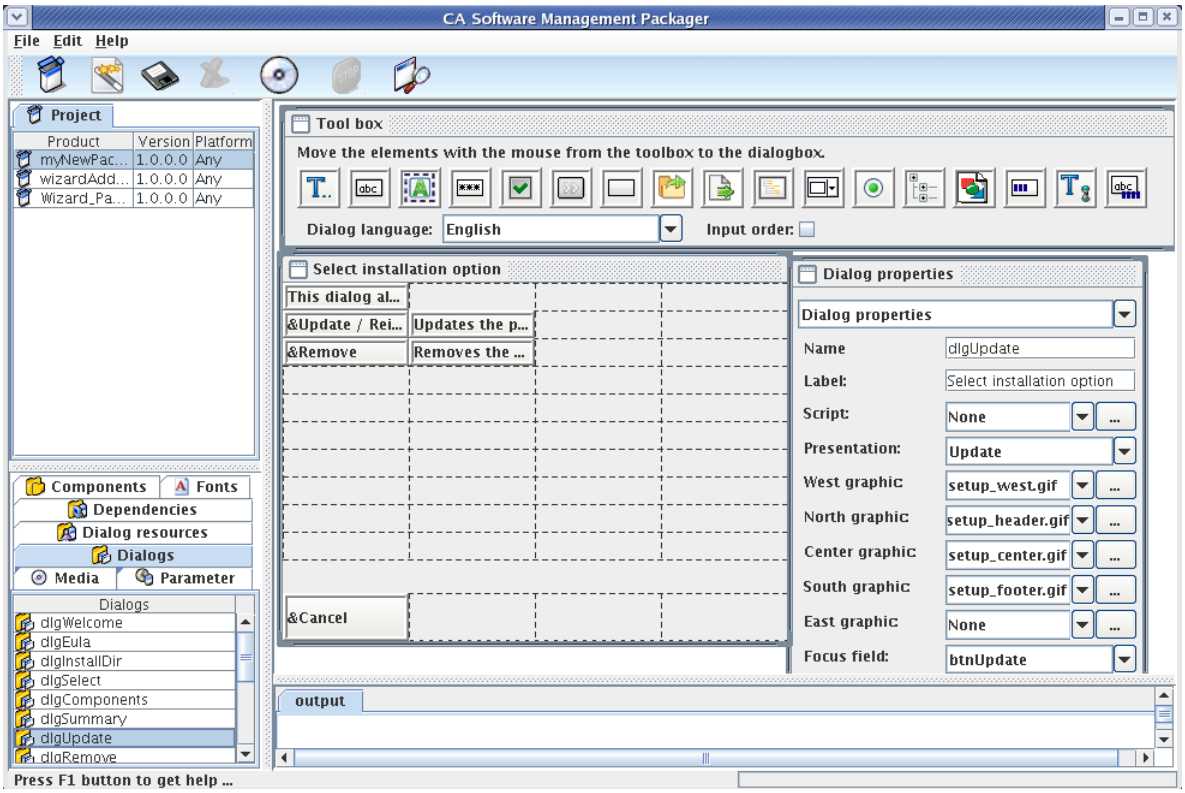
The Packager GUI provides an installation dialog editor and several tools and features to modify standard installation dialogs to fit your needs or create individual new installation dialogs. The installation dialog editor lets you drag and drop dialog elements from a toolbox into a work area and customize the dialog elements through properties boxes.

**Note:** The drag and drop task involves the following steps:

1. Select an element in the toolbox.
2. Move the element into the dialog editor panel (work area) while pressing the right mouse button.
3. Release the mouse button at the place where you want to put the element.

The installation dialog editor opens when you select a dialog on the Dialogs tabs in the subproject pane or right-click on the Dialogs tab and choose to add a new dialog.

The following screenshot shows a sample installation dialog editor window:




Whenever you want to see the current layout of selected dialog, for example, during dialog creation, click the Test dialog icon on the toolbar of the Packager GUI.

## Workbox

The workbox is the work surface of the installation dialog editor. It represents the general layout of an installation dialog.

**Header area**

Contains the header (title) of the installation dialog. The header area is not shown in the work box. The header is defined in the Label field in the Dialog properties box.

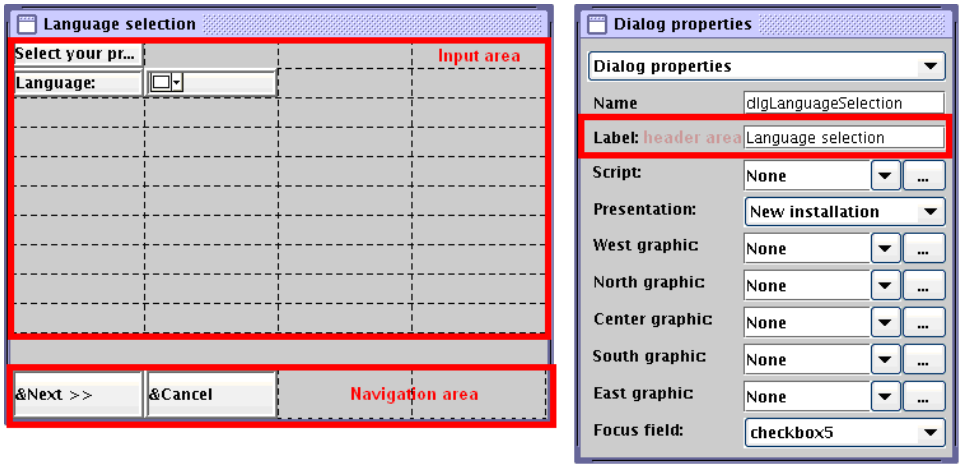
**Input area**

Indicates the main area of the installation dialog. Input area is represented by the 10x4 grid. All dialog elements except for the navigation button can be moved from the toolbox into this area using the drag-and-drop feature. Within the input area, all dialog elements can also be moved using the drag-and-drop feature. A virtual grid with dotted lines, four columns by ten rows, is layered over the input area to position the dialog elements. The grid layout is controlled by Java; the position and content of some dialog elements may influence the size and position of another dialog element.

**Navigation area**

Lets you define navigation buttons only. Navigation buttons are used to navigate between installation dialogs. Navigation area is the last line in the work box and is represented by the 1x4 grid.

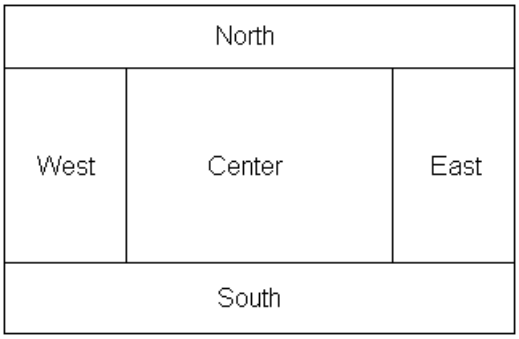
The following screenshot displays the Header, Input, and Navigation areas:




To view the current layout of the installation dialog being built, click the Test dialog icon on the toolbar.

## Graphics Areas

You can define background graphics within a Java installation dialog. You can position the graphics in different graphics areas using the points of a compass, as follows:




Additional areas for the background graphics are the header and footer areas of the dialog. The south area is used as the navigation area.

The only supported file format for graphics and background colors is GIF.

In the properties panel, you can select an existing graphic or assign a new graphic for the Dialog properties. The Dialog properties are presented if you click an empty field in the workbox. To insert a graphic, the GIF file must have been inserted in the pre-install component. The graphic can then be selected from the drop-down list in the Dialog properties box.

As the sizes of the graphics influence the overall size of the dialog, we recommend that graphics do not exceed 400x200 pixels (approximately) to support screen resolutions of 800x600 pixels (SVGA).

Adding graphics is supported only within Java installation dialogs. For VT100 mode dialogs, these features are ignored.

To view the current layout of the installation dialog being built, click the Test dialog icon (highlighted in the screenshot) on the tool bar, as shown in the following illustration:

## Toolbox and Dialog Elements

The toolbox contains all types of dialog elements that can be presented in an installation dialog.

The dialog elements that can be used in installation dialogs are represented by icons in the installation dialog editor's toolbox on the Packager GUI. You can drag and drop the dialog elements from the toolbox to the desired position in the workbox area.

In the prototype file, the dialog elements and their positions in the installation dialogs are specified in the Dialog section through appropriate keywords and properties.

Specific dialog elements can be grayed out and input fields blocked if they are not used within the current context. Therefore, use the internal parameter PIF_CHANGEPARAM_*parameter*.

### Button

A button includes a specification of an action that is performed when the button is clicked, for example, a new dialog is started, a script is executed, or the installation mode (Typical, Minimal, or User-specific) can be selected.

In the prototype file, the keyword #button lets you define the button within the input area of a dialog and specify text or icon to display on the button, parameter value assignments, the action, or the script that are executed when the button is clicked. For specification details see the sections Keywords (see page 111) and Properties (see page 120).

The text to display in the button is stored at a separate location, the Resource section, in the prototype file. This separation makes it easy to localize the texts, and to keep different localized versions within the same prototype file.

In the button text string, you can use the ampersand (&) character to define a shortcut. The ampersand character precedes that character, which can be used as shortcut, for example, if you write Ne&xt as the button text, the small letter x is the defined shortcut. The button text string can contain a maximum of 1024 characters, but it must not contain any control character, such as CR (carriage return) or LF (line feed).

**Important!** Shortcut keys in installation dialogs do not work, if you execute the installation dialogs in a Common Desktop Environment (CDE) session of HP-UX systems of Versions 11.0 and 11.i.

## Check Box

A check box can have the logical values True (the check box is checked) or False (the check box is cleared) and can be used to select an item on the installation dialog (True) or unselect an item (False). In installation dialogs, check boxes are used, for example, to select optional components to install or to configure product properties. It is possible to define sets of parameter=value assignments that are activated when the check box is selected or when the check box is cleared.

For the support of check box trees, a descriptive text, a list of dependent check box dialogs, and an action script can be specified. The action script changes attributes or contents of fields in the installation dialog immediately.

## Check Box Tree

A check box tree is a hierarchical tree structure of check boxes, for example, to list product components for selection. Use the Check Box Tree icon from the tool bar to define the structure of a check box tree in an installation dialog.

See also under "Check Box" for further information.

## Combo Box/List Box

This control works as a combo box or a list box depending on whether you select the Input Enabled property. If Input Enabled is selected, the control works as a combo box; otherwise, it works as a list box.

**Combo Box**

A combo box presents a list of possible values and allows selecting one of them. Alternatively, the user can enter another value. A shell script can be specified that validates the input, and another script that changes attributes or contents of fields in the installation dialog immediately. In VT100 dialogs, navigation is possible with the arrow keys or + and –. If the + or – key is pressed the first time, the list of possible values pops up. Further pressing + or – navigates up and down the list. Carriage Return confirms the selected value and closes the list.

**List Box**

A list box presents a list of possible values and allows selecting one of them. In VT100 dialogs, navigation is possible with the arrow keys or + and -. If the + or - key is pressed the first time, the list of possible values pops up. Further pressing + or - navigates up and down the list. Carriage Return confirms the selected value and closes the list.

## File Area

A file area displays the contents of a file, which must be provided by the pre-install component. A file area is used to display large text files like the end-user license agreement, a help topic, or the readme. Additionally, the contents of some internal administrative files can be displayed using a file area, for example, a summary of all parameter values used in the product. Navigation within the texts and scrolling are possible.

Parameters can be used to dynamically set the file name. Parameters of special use are: PIF_RESPONSE_FILE, PIF_ERROR_FILE, and PIF_LOG_FILE. For example, when you specify the internal variable $PIF_RESPONSE_FILE as file name, the contents of the internally used response file can be displayed.

The file area displays files with the following extensions:

- On Java user interface: .htm, .html, .rtf, and .txt
- On VT100 user interface: .txt files

The file area feature lets you provide different file formats for one file area.

For example, the following file area specification uses readme.html for the Java user interface and readme.txt for the VT100 user interface:

```
#filearea: 1 , 1 , MyFileArea , readme ;
```

To navigate within the file area in VT100 mode installation dialogs, use the following keys:

- Plus sign (+) or Ctrl + F

  Forwards you to the next page.

- Minus sign (–) or Ctrl + B

  Moves you back to the previous page.

- Tab

  Exits from the file area.

## File Selection

The file selection button lets you select a file for the PIF package.

## Image

An image is a picture that can be added to an installation dialog. Images can be shown in their original size or can be scaled to a well-defined size.

## Installation Directory Field

The installation directory field specifies the installation path for the PIF product or one of its components. When a Java installation dialog is being created, a browse button (...) to start a directory selection is automatically added. The text string entered in the installation directory field is assigned to a parameter, which is then passed through the install process. That is, a parameter has to be defined that stores the directory. The parameter must match the product installation directory parameter or the component installation directory parameter. You can specify a script that validates the content of the input field.

## Label

A label describes another dialog element in an installation dialog. The label usually leads input fields, such as installation directory specifications or check boxes. On the installation dialog, the label text cannot exceed a single line, that is, it is not wrapped within the dialog window.

## Navigation Button

A navigation button specifies the next installation dialog to appear and lets you initiate different actions. A shell script can be specified that determines the next installation dialog. You can drag and drop navigation buttons from the toolbox only into the navigation area at the bottom of the workbox.

The following table includes the actions that can be associated with a navigation button:

| Action | Description | Action Number |
|---|---|---|
| Cancel | Aborts the installation | 0 |
| Install | Installs the product | 1 |
| Next dialog | Starts the next dialog<br>**Note:** The name of the next dialog must be defined in the "Next dialog" field. | 2 |
| Remove | Removes the installed product | 3 |
| Previous dialog | Returns to the previous dialog | 4 |
| Dynamic dialog | Determines the name of the next dialog by executing a shell script, and starts that dialog<br>**Note:** The name of the script must be defined in the "script" field. | 5 |
| Start script | End the Installer GUI and start a user script<br>**Note:** The name of the script must be defined in the "script" field. | 6 |
| Goto | Jumps to another dialog and takes over the origin of that dialog, that is, the Back button (previous dialog) will lead to the predecessor dialog of that new dialog.<br>**Note:** The name of the target dialog must be defined in the “Next dialog” field. | 7 |
| Start script and continue | Closes the current dialog, starts the specified script, and starts the initial dialog again.<br>**Note:** The name of the script must be defined in the “script” field. | 8 |
| Execute script (Button only) | Executes the specified script and shows the result in the specified result pane. The result pane can be a progress area or a text area. | 9 |

When creating installation dialogs, you can define four navigation buttons in the navigation area. To visualize their function, you can specify text or an icon that appears on the button. When you choose the Dynamic dialog action, you must specify the script to use.

In the prototype file, use the #navbutton keyword in the Dialog section to define a navigation button. On the Packager GUI, you specify navigation buttons in the Navigation button properties dialog.

In the button text string, you can use the ampersand (&) character to define a shortcut. The ampersand character precedes that character, which can be used as shortcut, for example, if you write Ne&xt as the button text, the small letter x is the defined shortcut. By default, the first character of the text string serves as the shortcut. The button text string must not contain any control character, such as CR (carriage return) or LF (line feed).

You must define at least one Cancel dialog for each navigation button that has the Cancel action associated.

**Important!** Shortcut keys in installation dialogs do not work, if you execute the installation dialogs in a Common Desktop Environment (CDE) session of HP-UX systems of Versions 11.0 and 11.i.

## Password Field


A password field lets you enter a password. The password information is not visible and is masked with the asterisk (*). A shell script can be specified that validates the input data.

## Progress Area


A progress area shows the output of a script. This feature is useful to show the progress of long running scripts during the installation of PIF products. The progress area is reset whenever a new script is invoked. The name of the script currently processed is shown in the Action field of the Progress dialog. This feature is not available for packages in the RPM packaging format.

## Progress Bar

A progress bar shows the progress of the installation of a product or a component, or the overall installation progress. You can drag and drop the progress bar icon from the toolbox to the work area only if you have specified that you are creating a progress dialog, that is, the Presentation field in the dialog properties is set to Progress.

The installation progress of the current product or component is described through the following environment variables:

- PIF_PROGRESSBAR_PRODUCT
- PIF_PROGRESSBAR_COMPONENT
- PIF_PROGRESSBAR_COUNT

## Progress Label

A progress label shows information about the installation progress that changes during the installation process. The progress label usually leads a progress bar and informs the user about the current install action (for example, install, backup, remove) and optionally the installation phase, the name of the file or component currently being installed, or the elapsed installation time. You can drag and drop the progress label icon from the toolbox to the work area only if you have specified that you are creating a progress dialog, that is, the Presentation field in the Dialog properties is set to Progress.

The currently running installation action, the file being installed, the component name, and so on, are described through the following environment variables:

- PIF_PROGRESSLABEL_ACTION
- PIF_PROGRESSLABEL_FILE
- PIF_PROGRESSLABEL_COMPONENT
- PIF_PROGRESSLABEL_ELAPSEDTIME
- PIF_PROGRESSLABEL_PHASE

## Radio Button

A radio button allows switching between two or more values. You can specify a list of parameter=value assignments that are set when the radio button is selected. A shell script can be specified that changes attributes or contents of fields in the installation dialog immediately.

A button can be selected using Space bar and Carriage Return. Groups of radio buttons must be organized top-down on the dialog, that is, not side-by-side.

### Text Area

A text area displays any text in an installation dialog. The text can exceed one line and is automatically wrapped within the dialog window.

We do not recommend using a text field and a text area in the same line of an installation dialog.

### Text Field

A text field is the standard input field for installation dialogs. The text entered is assigned to a parameter, which is then passed through the install process and made available for scripts. The parameter must be defined as a property of the text field. You can specify a script that validates the text entered in the text field.

We do not recommend using a text field and a text area in the same line of an installation dialog.

## Properties Box

The properties box displays the properties of a dialog element selected in the workbox, and lets you enter or change property values.

Additionally, in the properties box you can choose any of the dialog elements currently defined for the installation dialog from a drop-down list to view or change their individual properties. For example, to change the title of the dialog, choose Dialog properties from the drop-down list.

# Support of Various Packaging Formats

The products managed by the PIF SDK can be packaged in a variety of packaging formats, including:

- PIF (CA Technologies Product Interchange Format)
- PKG (UNIX System V Release 4 standard packaging format)
- RPM (Red Hat package manager format)
- IBM AIX packaging format
- Hewlett Packard (HP-UX) packaging format
- Apple Macintosh Mac OS X packaging format

Although you cannot create new products in a format other than PIF, you can register, install, or at least list, existing product packages in other packaging formats, using PIF SDK functions.

The Installer manages dependencies from PIF products to required packages in other packaging formats, including PKG, RPM, AIX, and HP-UX packaging formats.

AIX and HP-UX packages are listed by the lsm command and considered by the Installer's dependency checker.

Patches on System V Release 4 systems (that is, Sun Solaris, SCO UnixWare, and NCR) are listed by the lsm command and considered by the Installer's dependency checker.

Apple Macintosh Mac OS X packages installed in the /Applications directory are listed and registered within the DSM Explorer. The query and removal methods provided by the Installer (lsm command) apply also to Mac OS X packages. During installation of Mac OS X packages the Installer adjusts all icons and menus.

PIF packages can be registered to computers running Mac OS X and are visible on those computers within the OSX package manager.

## Functions for PKG, RPM, AIX, HP-SD, and OSX Package Integration

The following functions for PKG, RPM, AIX, HP-SD, and OSX integration are available with the Installer:

- Define dependencies to PKG, RPM, AIX, HP-SD, and OSX packages
- Install RPM, AIX, HP-SD, and OSX package hierarchies
- Remove RPM, AIX, HP-SD, and OSX package hierarchies
- Internal parameters (PIF_*xxx*) that are passed to PIF packages are also passed to RPM, AIX, HP-SD, and OSX packages.

## RPM-specific Integration Functions

The following functions are available only for RPM package integration:

- Pass the installation path to RPM (option -prefix)

  A special PIF parameter is used to pass the installation directory path to the RPM installation. This parameter is named as follows:

  PREFIX_*rpm_packagename*

  Dashes within RPM package names are replaced by underscores.

  Examples of parameters containing install paths for RPM packages are:

  $PREFIX_openoffice — contains install path for openoffice:

  $PREFIX_XFree86_libs — contains install path for XFree86-libs

- Show RPM installation progress (option -percent). The RPM installation progress information provided by the option -percent is evaluated and moves the PIF product progress bar in the progress dialog ($PIF_PROGRESSBAR_PRODUCT).

  **Note:** RPM does not provide progress information at removal time.

- Configure RPM command options per RPM package. These options are defined as comment in a script. Additional install options can be defined in an RPM pre-install (prein) script, removal options can be defined in an RPM pre-remove (preun) script. Keywords in these scripts are as follows.

  In the prein script:

  ```
  # PIF_RPM_INSTALL_OPTION=
  # PIF_RPM_REINSTALL_OPTION=
  # PIF_RPM_UPDATE_OPTION=
  ```

  In the preun script:

  ```
  # PIF_RPM_REMOVE_OPTION=
  ```

  For the method REINSTALL the default is set to --replacepkgs. For the methods INSTALL, UPDATE, and REMOVE no defaults are set.

  **Note:** In attended mode the option -percent is automatically added for install, reinstall and update jobs.

  Example of a prein script:

  ```
  #!/bin/sh
  # following option is used to overwrite the RPM update installation switches:
  # PIF_RPM_UPDATE_OPTION=--replacepkgs --replacefiles
  exit 0
  ```

## Restrictions with the Use of RPM Packages

The following restrictions apply with the use of RPM packages:

- Mixed dependency configurations are not supported, for example, if a PIF product depends on an RPM product, which depends on another PIF product.
- RPM packages are accessed solely with the help of library.dct.
- RPM autorequirements, for example, to shared libraries, shells, and so on, are not supported.
- Disk space check (for complete transaction) is not available
- Backup of RPM packages is not supported, so no rollback is available.
- As appropriate RPM information is not provided, the progress of RPM package removal cannot be shown in the progress dialog.
- Cancel of an RPM installation terminates after the particular RPM installation has been completed, to keep the RPM operation consistent.
- You must not provide both types of packages, PIF and RPM, with the same product name in the same directory.

## Native RPM and PKG Switches

You can define native switches for RPM and PKG installations.

PKG packages are installed on the according platforms (such as Solaris, UnixWare, NCR, SINIX, and so on) by using the "pkgadd" command. The lsm installer invokes the command with a fixed set of switches.

You can use the command line with the lsm option -nativeswitches to add additional switches. lsm adds these native switches to the pkgadd command.

**Example:**

```
lsm -i MY_TEST_PKG -nativeswitches "\-G"
```

The native switch "-G" is added to the pkgadd command. According to this switch, the PKG package "MY_TEST_PKG" is installed only in the "global zone" of a Solaris system.

# Installer Functions Overview

The Installer manages software installations on various target computer operating environments. All functions of the Installer are available through the lsm command and its numerous options at the command line.

Following is an overview of relevant Installer functions:

- Installs a PIF, PKG, AIX, HP-SD, OSX, or RPM product, dialog-driven or in unattended (silent) mode. To customize an unattended installation, a response file can be added.
- Updates an installed PIF product on the target computer. In the case of an update installation, the Installer creates a backup of the installed PIF product. During the update installation, the differences between the previously installed product and the new product are evaluated. Files no longer used are automatically removed from the target computer.
- Resolves all dependencies defined in the product and its components, if needed over several layers, and ensures that all dependent products are installed before the product is installed. Removes all dependent products if possible, and over several layers if needed, when a product is being removed (applies only to dependent products delivered and installed with the product).
- Tests new installation dialogs and creates a response file with the values entered in the dialogs.
- Verifies checksums on the target computer. During product creation, the Packager adds checksums to the prototype file. The PIF product file is secured by a CRC checksum (CRC = Cyclic Redundancy Check). On the target computer, the Installer is aware of this CRC checksum, and alerts you if installed files of the PIF product have been changed subsequent. An installation fails if the PIF product file has been changed from outside.
- Manages effectively shared files during the installation of a PIF product. (A shared file is installed and used by more than one product at the same time.) During an update installation of a PIF product that contains a shared file, this file is updated only if the checksums of the existing and the new file differ, and the modification date of the new file is newer than the existing one. Otherwise, the shared file is not changed.

  The installation of a PIF product aborts if one of the following conditions is true:

  - The file types of shared files differ (for example, a standard file is installed, and a directory should be created with the same name)
  - The targets of a shared link differ
- Checks an installed product for consistency. The product files are checked for existence and access rights.

- Restores files after package removal. If a new PIF installation starts, the Installer notes all files that will be overwritten by the product installation. These files are stored in a specific file, *.notremove*. The *.notremove* file is created in the Installer's home directory (/opt/CA/SharedComponents/installer) by default. The file is protected by a checksum and no user can modify it. Additionally, it contains all files that have been overwritten by the new installation.

  If the package is removed and no other package references a not removable file, the entry is removed from *.notremove*. As a result, the file is not removed it remains with its last content on the system.
- Queries an installed product or product file and shows the product properties and dependency hierarchies.
- Lists all installed products or only the shared components. Prints the version of the Installer used.
- Removes an installed product, in unattended mode if desired.
- After the last PIF product has been removed from the target computer, the Installer removes itself from the target computer (cleanup).
- Displays the progress and informative text at the console in the case a large PIF product with a long installation preparation phase is being installed.
- Installs patches.
- Installs packages multiple times.
- Restores files after failed package installations.
- Leaves files that exist before the first package installation.

# No Dependency on the libncurses Library

The lsm installer no longer links with the libncurses library. An unattended installation succeeds without having any libncurses library installed on the system. The library is necessary only if a VT100 dialog is executed.

# Self-installing PIF Products

The Packager lets you create self-installing PIF products, which do not require an Installer be installed on the target computer prior to installation of the PIF product. A self-installing PIF product is a bundle consisting of the PIF product and the Installer within a single executable file.

The structure of a self-installing PIF product is as follows:

- Script to extract the PIF product
- Optional: JRE
- Installer (compressed)
- PIF product (compressed)

A self-installing PIF product is executed as a shell script.

The Installer uses or creates the following directories on the customer system when it is installed by a self-installing PIF product:

- The Installer files are stored in the $CASHCOMP/installer directory by default. This directory will be either created or updated.
- The program lsm is added to /usr/bin.

Self-installing PIF products can be created through the Packager GUI or the pifself command on the command line.

You can split the self-installing PIF product into three separate units, the start script, the actual PIF product, and a self-installing shell script (ca-sm-installer) including the Installer. To have these separated units is of value if you are working in a software delivery environment and do not need the self-installing function. In this case, you can register only the separated actual PIF product in the Software Package Library. Whenever you want to use the self-installing function, you must ensure that the three separate units mentioned previously are located in the same folder, and run the self-installing shell script (ca-sm-installer).

When creating a self-installing PIF product, you can optionally specify the operating environment where the self-installing product should install. This option lets you create self-installing products from PIF products that are created for Any platform.

# Integration of JRE in Self-installing PIF Products

The graphical user interface of the Installer for Linux and UNIX is based on the Java technology. Therefore, the Java Runtime Environment (JRE) must be installed on the target system to run the installation dialogs. If the JRE is not installed on the target system, an alpha-numerical VT100 installation dialog is executed instead of the expected Java dialog.

To avoid a fall back to the VT100 dialog mode, you can integrate a JRE in a self-installing PIF product which will then be installed and executed on the target system.

To integrate a JRE in a self-installing PIF product, the pifself command provides the -j option. This option adds the core JRE files to the self-installing PIF product. On the target system the Installer uses this integrated JRE to run the Java wizard of the Installer.

With the -j option you can specify a JRE version to integrate or decide to use a downsized version for Linux. Enter a path name including the "jre" subdirectory, for example, /usr/java/j2sdk.1.4_10/jre.

This downsized JRE version (V1.4.2.10) for Linux platforms contains only the necessary classes and files to run the Java wizard of the Installer.

For all other platforms you can integrate any standard JRE with a version greater than 1.4 in the self-installing PIF product.

# Product Family Concept

The product family concept ensures that all packages belonging to the same product family are updated in sync to the latest available version ("upgrade one, upgrade all").

The keyword #productfamily describes the property that combines all packages that logically fit together (for example, product family = CCS). Any package can belong to a single product family.

Based on the keyword #productfamily all products of the same product family are updated to the latest version level using the Installer command lsm with the -u option. This command lets the Installer scan for all matching products on the system and update them to the highest version.

All packages are updated to their highest version. A product that is provided with a lower version (in the local folder or on the 'Update CD') is not downgraded.

The structure of the 'Update CD' or local folder requires all packages either being provided in a flat folder or described through a library.dct file that must be provided separately.

For RPM packages the tag 'group' is used to identify the product family.

# Migration of Non-PIF Products

Sometimes, during an update, it may be useful to transform previously installed non-PIF products into PIF products.

Therefore, the Installer provides the migration mechanism. This mechanism is based on a (user-written) shell script, which is started the first time a PIF product is installed on the target computer. This script is not started during updates or reinstalls. Within this script, you can arrange the internal parameters for product installation path and product install job (INSTALL or UPDATE). These parameter values overwrite the default parameter settings.

# Installer Compatibility Considerations

The following table shows whether a PIF product that was packaged with a specific Packager version (listed in the first column of the table) can be installed on the target computer, depending on the Installer version available on the target computer. The table shows also, which Installer version must be available on the target computer in order to install a PIF product packaged with a specific Packager version.

The term installable marked with an asterisk (installable *) in the table means, that the self-installing PIF product will automatically install the suitable Installer or upgrade an existing Installer (if necessary) on the target computer before the PIF product is installed.

| **PIF product to install is packaged with Packager Version** | **Installer version present on the target computer: None** | **Installer version present on the target computer: 3.1** | **Installer version present on the target computer: 4.0** | **Installer version present on the target computer: 4.1** | **Installer version present on the target computer :4.2** | **Installer version present on the target computer :4.3** |
|---|---|---|---|---|---|---|
| 3.1 | Installation not possible | installable | installable | installable | installable | installable |
| 4.0 | Installation not possible | installation fails | installable | installable | installable | installable |
| 4.1 | Installation not possible | installation fails | installation fails | installable | installable | installable |
| 4.2 | Installation not possible | installation fails | installation fails | Installation fails | installable | installable |
| 4.3 | Installation not possible | installation fails | installation fails | Installation fails | Installation fails | installable |
| 4.0, as self-installing | installable * | installable * | installable * | installable | installable | installable |
| 4.1, as self-installing | installable * | installable * | installable * | installable * | installable | installable |

| PIF product to install is packaged with Packager Version | Installer version present on the target computer: None | Installer version present on the target computer: 3.1 | Installer version present on the target computer: 4.0 | Installer version present on the target computer: 4.1 | Installer version present on the target computer :4.2 | Installer version present on the target computer :4.3 |
|---|---|---|---|---|---|---|
| 4.2, as self-installing | installable * | installable * | installable * | installable * | installable * | installable |
| 4.3, as self-installing | installable * | installable * | installable * | installable * | installable * | installable * |

**Example:** A PIF product packaged with the Packager Version 4.3 cannot be successfully installed on a target computer that runs Installer Version 4.2.

# Chapter 4: Working with the Packager User Interface

This chapter provides information on how to achieve some of the important tasks by using the PIF Packager user interface.

This section contains the following topics:

Create a New PIF Package (see page 71)
Save a PIF Package (see page 72)
Build a PIF Package (see page 73)
Register a PIF Package to Library or to Disk (see page 74)
Modify a PIF Package (see page 75)
Configure the Encryption Method for a PIF Package (see page 76)
Unload a PIF Package (see page 76)
Delete a PIF Package (see page 77)
Import a PIF Package (see page 77)
Create a Multilingual PIF Package (see page 78)
Patch Product Creation Wizard (see page 81)

## Create a New PIF Package

You can create a new PIF package by using the PIF Packager GUI. Creating a new PIF package is to create a new prototype file that contains all information about the product definition.

**To create a new PIF package**

1. Do one of the following on the Packager GUI:
   - Select File, New from the menu bar.
   - Click the New icon on the toolbar.
   - Right-click in the project pane and select New from the context menu.

   The New package dialog appears.

   **Note:** You can also use the PIF Package Creation Wizard to create a new PIF Package. To launch the wizard, click the wizard icon in the toolbar (or right-click in the Project pane and select New package wizard from the context menu), and follow the step-by-step instructions provided in the wizard.

2. Enter information about the PIF package in the following fields:

   **Package name**

   Indicates the name of the PIF package. The package name can include up to 128 characters.

   **Package version**

   Indicates the version of the PIF package. The format of the version is M.m.b.r, where "M" signifies *major*, "m" signifies *minor*, "b" signifies *build*, and "r" signifies *revision*.

   **Package platform**

   Indicates the UNIX or Linux platform specifying the hardware and operating system on which the PIF package can be installed.

   **Note:** The option "Any" lets you install the package on all supported UNIX and Linux platforms.

   **Template**

   Indicates the template used as a framework for the creation of an individual package setup. A template is formatted as a prototype file. Select one of the following templates based on your requirements:

   - standard.Any.@enu
   - advanced.Any.@enu
   - unattended.Any.@enu
   - multi_language.Any.@enu
   - multi_instance.Any.@enu

3. Click New.

   The new package is created and appears in the project pane.

# Save a PIF Package

After you create a new PIF package, you must save it before you start building it.

**To save a PIF package**

1. Select the package that you want to save in the Project pane.
2. Do one of the following on the Packager GUI:
   - Select File, Save from the menu bar.

- Click the Save icon on the toolbar.
- Right-click the package and select Save from the context menu.

The package is saved.

# Build a PIF Package

After completing the settings for the PIF package and saving it, you must build the PIF package.

**To build a PIF package**

1. Select the package in the project pane.
2. Do one of the following:
   - Right-click the selected entry and select Build from the context menu.
   - Choose File, Build from the menu bar.
   - Click the Build icon on the toolbar.

   The Build package dialog appears.
3. You can specify the following build options:

   **Self installing**

   If this check box is checked, indicates that a self-installing PIF package (including the Installer) is automatically built. A drop-down list opens and lets you choose the package platform. The following options appear:

   - Separate installer and product: Select this check box to define that PIF and installer are separated.
   - Select SDK: Select the appropriate option from this drop-down list.
   - JRE Directory: Specify an optional JRE that is added to the package. This JRE is incorporated in the self-installing PIF package. As a result, graphical dialogs during the package installation can be displayed on a graphical terminal at the target computer, even if no JRE is installed on the target computer.

   **Strip**

   If this check box is checked, indicates that all symbolic information is removed from all binary files of the PIF package. This will reduce the size of the PIF package.

   **Overwrite existing**

   If the package already exists, indicates that this check box appears on the dialog. This box is checked by default, that is, the existing PIF package with the same name is overwritten.

4. Click Build.

   The Save in dialog appears where you can specify the location to store the PIF package. By default, the PIF package is stored in a directory referenced by $SDPcker/productlib.

After the PIF package has been successfully built, the Media tab contains a corresponding product entry.

# Register a PIF Package to Library or to Disk

After a PIF package has been successfully built, you must register it in the Software Package Library on a manager system ("register to library") or to a folder on the local disk ("register to disk").

**To register a PIF package on a manager system ("register to library")**

1. Select the package in the project pane.
2. Select the PIF file (*.@pif) of the package on the Media tab in the subproject pane.
3. Right-click the selected entry on the Media tab, choose Register, and then "to library" from the context menu.

   The Register to library dialog appears.

   **Note:** Click the >> button to expand the dialog, and the << button to collapse it.
4. Enter the following properties:

   **Manager**

   Indicates the node name of the manager system where the Software Package Library resides. It lists all used managers.

   **User**

   Indicates the UNIX user name under which the product is registered. This user must have permissions to register packages in the software package library.

   **Password**

   Indicates the user-specific password.

   **Security Provider, Domain**

   Indicates the domain name if a domain controller is used. Based on a valid manager name (Manager drop-down list), all security providers are scanned, and applicable security providers are added to the drop-down list.

**Response file(s)**

(Optional) Indicates the file or files that contain parameter settings to configure the package installation. You can enter a single response file name or a list of response file names here. A list of response files must be separated by commas without any spaces, for example, respf1,respf2,respf3.

5. Click Register.

   The registration process starts.

**To register a PIF package on the local disk ("register to disk")**

1. Select the package in the project pane.
2. Select the PIF file (*.@pif) of the package on the Media tab in the subproject pane.
3. Right-click the selected entry on the Media tab, choose Register, and then "to disk" from the context menu.

   The Register to disk dialog appears.

4. Click Register.

   A reginfo structure is created in the current product folder on disk. This reginfo structure can be written to a disk and distributed and registered in the Software Package Library using software delivery methods, including drag-and-drop functionality.

# Modify a PIF Package

You can modify a package to make any updates after you create the package. You do not need to create a new package to incorporate changes at a later stage like adding a new element to the existing dialog. This also allows you to keep the package up-to-date.

The following example explains how to modify an already created package by adding a new element to a dialog:

**To add a new element to an already existing dialog**

1. Select a package from the Project pane.
2. Select the Dialogs tab from the subprojects pane.

   A list of all the dialogs included in the package appears.

3. Select the dialog in which you want to add a new element.

   The dialog (including already existing elements) appears in the right pane.

4. Select a new element from the dialog editor's toolbar and drop it at the appropriate place in the dialog.

   The new element is added to the already existing dialog and the package is modified.

# Configure the Encryption Method for a PIF Package

If a PIF package contains a password and is likely to be installed in unattended mode, you must specify the method and algorithm that you want to use for encrypting and decrypting the password.

**Note:** Packages that are always installed interactively do not require password encryption as the password is masked anyway.

**To configure the encryption method for a PIF package**

1. Select the package from the Project pane.

   The right pane displays the properties of the package.

2. Click the Security tab.

   **Note:** This tab is enabled only if the package contains a password.

3. Select the inbuilt or external encryption method and specify the commands for encrypting and decrypting the password. The inbuilt encryption method is not FIPS-compliant because it uses Blowfish encryption algorithm. To use FIPS-compliant encryption, you need to rely on an external encryption method that is FIPS-compliant. For more information about the encryption method and commands, see the *Software Packager for Linux and UNIX Help.*

   When you are create a response file for the package using the lsm -a command, the password is encrypted using the method and algorithm that you have selected.

4. Click OK.

   The encryption method is configured.

# Unload a PIF Package

You can unload (remove) a PIF package from the project pane, but the package is not removed from the disk. The package is still contained in the product library (productlib), from where you can import it again. You can unload one or more packages at the same time.

**To unload a PIF package**

1. Select the packages in the project pane.
2. Right-click a selected package and choose Unload from the context menu.

   A dialog box appears asking you if you really want to unload the selected package.

3. Click Yes.

   The selected package is removed from the project pane.

# Delete a PIF Package

You can delete a PIF Package if you do not need it.

**To delete a PIF package**

1. Select the package that you want to delete in the Project pane.
2. Do one of the following on the Packager GUI:
   - Select File, Delete from the menu bar.
   - Click the Delete icon on the toolbar.
   - Right-click the package and select Delete from the context menu.

   A message appears stating that all related package prototype and PIF files will be deleted from the system and you will not be able to restore the package.
3. Click Yes.

   The selected package is deleted.

# Import a PIF Package

Importing a PIF package means that you are importing an already built PIF package (xxx@pif) or a PIF prototype file (xxx@prm).

When you are importing an already built PIF package, you must specify a target directory where you want the imported PIF package to be extracted.

The name of the imported PIF package appears in the Project pane of the Packager GUI, and you can modify the imported PIF product.

**To import prototype files**

1. Select Import from the File menu or right-click in the project pane and select Import from the context menu.

   The Choose a file dialog appears.
2. Browse for one or more prototype files (*.@prm) or search for all types.

   Alternatively, you can specify the names of the prototype files in the File Name input field.

   **Note:** Only valid PIF prototype files are accepted for import.
3. Click Open.

   The imported packages appear in the project pane.

**To import an already existing PIF package**

1. Select Import from the File menu or right-click in the project pane and select Import from the context menu.

   The Choose a file dialog appears.

2. Browse for the PIF package (*.@pif).

   Alternatively, you can specify the name of the PIF package in the File Name input field.

   **Note:** Only valid PIF packages are accepted for import.

3. Click Open.

   The Import package dialog appears.

4. Specify the location of the folder where you want the imported PIF package to be extracted.

5. Click Import.

   The name of the imported PIF package appears in the Project pane of the Packager GUI, and you can modify the imported PIF package.

# Create a Multilingual PIF Package

You can create a multi language PIF package by using the multi_language.Any.@pif template. The multi language PIF package lets you specify the language to use for the product installation and also lets you select (or deselect) the language-specific components for the installation.

**To create a multi language PIF package**

1. Do one of the following on the Packager GUI:

   - Select File, New from the menu bar.
   - Click the New icon on the toolbar.
   - Right-click in the project pane and select New from the context menu.

   The New package dialog appears.

2. Enter information about the PIF package in the following fields:

   **Package name**

   Indicates the name of the PIF package. The package name can include up to 128 characters.

   **Package version**

   Indicates the version of the PIF package. The format of the version is M.m.b.r, where "M" signifies *major*, "m" signifies *minor*, "b" signifies *build*, and "r" signifies *revision*.

**Package platform**

Indicates the UNIX or Linux platform specifying the hardware and operating system on which the PIF package can be installed.

**Note:** The option "Any" lets you install the package on all supported UNIX and Linux platforms.

**Template**

Indicates the template used as a framework for the creation of an individual package setup. A template is formatted as a prototype file. Select multi_language.Any.@enu from the drop-down list.

3. Click New.

   The new package is created and appears in the project pane.

## Select the Language for the Installation Dialogs

For a multilingual package, you can select the language for the installation dialog and also preview the dialog in the selected language.

**To select the language for the installation dialogs**

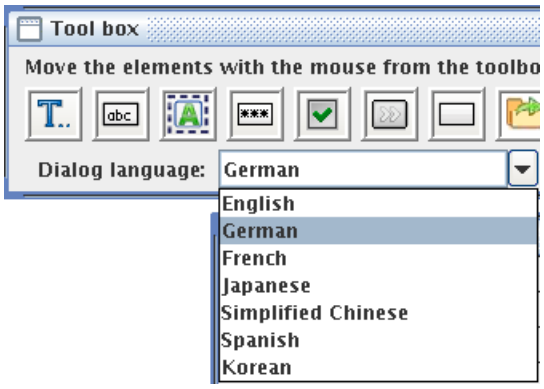
1. Select the multilingual package from the Projects tab.
2. Select the Dialogs tab.

   A list of dialogs appears.

3. Select a dialog.

   The dialog editor appears in the right pane.

4. Select the appropriate language from the Dialog language drop-down list, as shown in the following illustration:



The language is changed for all the installation dialogs, as shown in the following illustration:


## Copy and Change a Dialog Resource

To provide texts in different languages for multilingual setups, it is useful to copy and change existing dialog resources.

**To copy a dialog resource**

1. Select the package in the project pane.
2. Select a language entry on the Dialog resources tab in the subproject pane.
3. Right-click the selected resource and select Copy from the context menu.

   The Resource dialog appears.
4. Choose a new language from the drop-down list.
5. Click Copy.

   The new dialog resource appears on the Dialog resources tab in the subproject pane.

**To change a text in a dialog resource**

1. Select the package in the project pane.
2. Select the resource to change on the Dialog resources tab in the subproject pane.
3. Select an entry to change in the Dialog texts table in the GUI work pane.

4. Right-click the selected entry and choose Change from the context menu.

   The Add/change text dialog appears.

5. Edit the text in the Text input box.

6. Click Add.

   The changed text is added to the dialog resource.

# Patch Product Creation Wizard

The patch product creation wizard lets you create patches to update existing PIF products. A patch for the PIF product is created, built, distributed, and installed like a "normal" PIF product.

To create a PIF patch product with the patch product creation wizard, proceed as follows:

1. Right-click a product in the Project pane, and select New patch from the context menu.

   The Create a new patch product wizard launches and displays the Patch name page.

2. Enter information about the patch name and version in the appropriate fields, and click Next.

   **Note:** The Patch name page also provides information about the product name and version for which you are creating a patch.

   The Select patch product generation page appears.

3. Click New patch.

   The Source folder page appears.

4. Enter the location and name of the directory that contains all changed files that are part of the patch product. Click Next.

   The Replaced patches page appears.

5. Enter names of all previously assigned patches that this patch will replace. Click Next.

   **Note:** Use a comma (,) to separate the names.

   The Summary page appears.

6. Click Create patch to create the patch for the product.

   The PIF patch product appears in the Project pane of the Packager GUI, and you can modify it like any other "normal" PIF product listed in the Project pane.

**Note:** Some tabs in the Packager GUI will be grayed out, because they do not apply to PIF patch products.

# Chapter 5: Building a Sample PIF Product

An example follows how to define and build a PIF product and configure its installation. The PIF product is created step-by-step, starting with the creation of the prototype, applying many of the features described under "Software Management for Linux and UNIX."

The entries in the prototype file that define the product, its files and directories, dependencies, and so on are described, and you see where and in which format to enter the specifications. The sequence of the description follows the sequence of entering the specifications in the prototype file.

One way to enter (and modify) the specifications in the prototype file is using a text editor. This way to proceed is assumed in the descriptions following. Additionally, for each product creation step the activities are described to enter the same specifications through the command line and the Packager GUI.

For detailed information about the commands used and entries in the prototype file (keywords, properties) see "Keywords and Properties." (see page 111) When you use the Packager GUI to create the PIF product MERCHANT, we encourage you to view also the Packager online help for information.

This section contains the following topics:

Requirements and Specifications of the UNIX Application MERCHANT (see page 83)
Create the Prototype File and Specify Product Information for the PIF Product MERCHANT (see page 84)
Define Dependencies for the PIF Product MERCHANT (see page 85)
Specify Files for the PIF Product MERCHANT (see page 87)
Add a Procedure to the PIF Product MERCHANT (see page 88)
Change the Installation Path of the PIF Product MERCHANT (see page 90)
Add a Dialog to Query the Installation Path of the PIF Product MERCHANT (see page 91)
Add Language-specific Product Files to the PIF Product MERCHANT (see page 93)
Localize the Installation Dialogs for the PIF Product MERCHANT (see page 95)
Build the Final PIF Product MERCHANT (see page 96)

## Requirements and Specifications of the UNIX Application MERCHANT

The application called MERCHANT with the Version 2.1.0.0 is being packaged into a PIF product. The application and how it works is not described in detail, but the requirements of the application are determined, as follows:

- The application runs on any UNIX or Linux operating environment.

- The application consists of a Java GUI, provided as a Java archive file called merchant.jar. The shell script run_merchant.sh starts the GUI. This shell script is linked into the /usr/bin directory.
- Any application data is stored in an Ingres Enterprise Relational Database (Ingres).
- The database tables are created after the product has been installed. The user-written script create_tables.sh will handle this.
- The user should be able to choose the installation directory; the default path is preset with the value /home/merchant.
- An installation dialog is used to determine the installation directory.
- The shared library libi8n.so provides an interface to print localized messages. It is installed once and used by different products.
- The product shows localized error messages, provided by the text files merchant.enu (English messages) and merchant.deu (German messages).
- The installation dialogs are able to provide English or German dialogs.

# Create the Prototype File and Specify Product Information for the PIF Product MERCHANT

The first step to building the PIF product MERCHANT is to create the prototype file that will contain the complete product definition. Following the naming conventions for prototype files and the requirements of the application, the name of the prototype file for the sample PIF product MERCHANT is as follows:

```
MERCHANT.Any.@prm
```

To create the prototype file, open a text editor in your current working directory and save the (empty) file with this name.

Enter the product information section as follows:

```
@PRODUCT:
#phead: MERCHANT , 2.1.0.0 ;
#sys: Any ;
#locale: ENU ;
#comment: This is an example for a PIF product ;
#ppath: /home/merchant ;
@ENDPROD:
```

The product information section defines the following product properties:

**#phead: MERCHANT , 2.1.0.0 ;**

Defines the PIF product name MERCHANT and the PIF product Version 2.1.0.0.

**#sys: Any ;**

Defines the operating environment on which the PIF product can be installed. In this case, Any states that the PIF product MERCHANT can be installed on any UNIX/Linux operating environment.

**#locale: ENU ;**

Defines the default locale of the PIF product MERCHANT. As a default setting, use the string ENU (English/US).

**#comment: This is an example for a PIF product ;**

After the keyword #comment you can enter any text to describe the product.

**#ppath: /home/merchant ;**

The PIF product MERCHANT will be installed in the directory /home/merchant on the target computer. This directory is created automatically during the product installation.

To create the prototype file for the PIF product MERCHANT including product information through the command line, execute the pifproto command, as follows:

```
pifproto -b /home/merchant -a Any -n MERCHANT -v 2.1.0.0
```

**To create the prototype file for the PIF product MERCHANT including product information using the Packager GUI**

1. Run the smgui command to launch the Packager GUI.
2. Select File, New from the menu bar.

   The New product dialog appears.
3. Enter the product name MERCHANT, the version 2.1.0.0, and the platform identification Any.

# Define Dependencies for the PIF Product MERCHANT

As specified in the requirements, the application MERCHANT is a database application using the Ingres Enterprise Relational Database (Ingres). Therefore, Ingres must be installed before the PIF product MERCHANT is installed.

To ensure the product MERCHANT is installed only if an appropriate Ingres database exists on the system, you must define a dependency in the prototype file. The highlighted new line in the product information section specifies the dependency and the required product, as follows:

```
@PRODUCT:
#phead: MERCHANT , 2.1.0.0 ;
#sys: Any ;
#locale: ENU ;
#comment: This is an example for a PIF product ;
#ppath: /home/merchant ;
#pdep: Ingres , 4.0.0.0 , >= ;
@ENDPROD:
```

**#pdep: Ingres , 4.0.0.0 , >= ;**

The name of the required product is Ingres. The version number 4.0.0.0 in combination with the subsequent operator >= specifies that the required Ingres product version must be greater than or equal to 4.0.0.0. The Installer checks if an appropriate Ingres version is installed on the target computer before installing the product MERCHANT.

To define dependencies through the command line add the new #pdep: line to the prototype file, using any text editor.

**To define dependencies through the Packager GUI**

1. Run the smgui command to launch the Packager GUI.
2. Select the PIF product MERCHANT in the Project pane.
3. Select the Dependencies tab.
4. Right-click within the subproject pane and select Add from the context menu.

   The Product dependency dialog appears.
5. Enter or select the following values:

   Product name: Ingres

   Product version: 4.0.0.0

   Dependency type: Mandatory

   Operator: >=
6. Click Add.

   The dependency specification is incorporated in the prototype file.

# Specify Files for the PIF Product MERCHANT

All files that make up the final PIF product are defined in component sections in the prototype file. For the PIF product MERCHANT, add the following section to the prototype file MERCHANT.Any.@prm:

```
@COMPONENT:
#chead: component1 ;
#dir: bin , root , sys , drwxrwxrwx ;
#file: bin/merchant.jar , /myhomepath/src/merchant.jar, root , sys , -r—r—r-- , 1 ;
#file: bin/run_merchant.sh , /myhomepath/scripts/run_merchant.sh, root , sys ,
-r-xr-xr-x , 1 ;
#slink: /usr/bin/run_merchant.sh , bin/run_merchant.sh ;
@ENDCOMP:
```

The product information section defines the following product properties:

**#chead: component1 ;**

Defines the unique name of a component of the PIF product MERCHANT, in this case, component1.

**#dir: bin , root , sys , drwxrwxrwx ;**

Adds the bin directory to the PIF product MERCHANT. The user root is the owner of this directory, which is assigned to the group sys. Permissions are "read-write-execute" (rwx) for everybody.

**#file: bin/merchant.jar , /myhomepath/src/merchant.jar , root , sys , -r---r---r--- , 1 ;**

**#file: bin/run_merchant.sh , /myhomepath/scripts/run_merchant.sh , root , sys , -r-xr-xr-x , 1 ;**

Defines the product files of the PIF product MERCHANT. The first property value is the target name of the file. Directory structures have to be added to the target name, too. In this case, each file will be installed in the bin directory. The next property specifies the source file name. This is the path, where the files are located when the PIF product is built. The target's owner, group, and access permissions follow it. The last property value (1) specifies the file type standard file.

**Note:** If the target file names are entered without a leading slash, the files are installed relative to the base directory of the product; otherwise, the files are installed using the absolute path name.

**#slink: /usr/bin/run_merchant.sh , bin/run_merchant.sh ;**

Creates a symbolic link in the /usr/bin directory. The second parameter is the path name of the source file.

Specifying files for the PIF product can be automated using the pifproto command from the command line. Additionally to creating the prototype file and the product information section, the pifproto command adds the paths and properties of all files stored in the product directory to the prototype file.

To specify files through the command line

1. Create a new folder structure (for example, /tmp/merchant and /tmp/merchant/bin).
2. Copy the files to be added into the /tmp/merchant/bin folder.
3. Execute the pifproto command as follows:
   ```
   pifproto -n MERCHANT -v 2.1.0.0 -a Any -d /tmp/merchant -b /home/merchant
   ```
   The prototype file MERCHANT.Any.@prm is located in the current working directory.

**To specify files through the Packager GUI**

1. Run the smgui command to launch the Packager GUI.
2. Choose File, New from the menu bar..

   The New product dialog appears.
3. Enter the following values:

   Product name: MERCHANT

   Product version: 2.1.0.0

   Product platform: Any

   Template: standard.Any.@enu
4. Click New.

   The product MERCHANT is displayed and highlighted in the project pane. In the GUI work area, the current product properties are shown. The installation directory is preset with the value /opt/merchant. A standard component with the name component1 and a pre-install component are automatically created and displayed on the Components tab in the subproject pane.
5. Select the product MERCHANT in the project pane
6. Choose File, Save from the menu bar.

   The current prototype file is saved.

# Add a Procedure to the PIF Product MERCHANT

The PIF product MERCHANT must create the database tables after the product files have been installed. Due to the specifications of the product MERCHANT, the user-written script file create_tables.sh will do this.

The script file properties are specified in the component section, as follows:

```
@COMPONENT:
#chead: component1 ;
#dir: bin , root , sys , drwxrwxrwx ;
#file: bin/merchant.jar , /myhomepath/src/merchant.jar, root , sys , -r–r–r-- , 1 ;
#file: bin/run_merchant.sh , /myhomepath/scripts/run_merchant.sh, root , sys ,
-r-xr-xr-x , 1 ;
#slink: /usr/bin/run_merchant.sh , bin/run_merchant.sh ;
#file: bin/create_tables.sh , /myhomepath/scripts/create_tables.sh , root , sys ,
-r-xr-xr-x , 1 ;
@ENDCOMP:
```

**#file: bin/create_tables.sh , /myhomepath/scripts/create_tables.sh , root , sys , -r-xr-xr-x , 1 ;**

The script file create_tables.sh should be installed in the target directory bin. The source directory for the script file is /myhomepath/scripts. Owner is root, group is sys, read (r) and execute (x) access is granted for all users, and the file type is 1 meaning standard file.

The script execution time is specified in the product information section, as follows:

```
@PRODUCT:
#phead: MERCHANT , 2.1.0.0 ;
#sys: Any ;
#locale: ENU ;
#comment: This is an example for a PIF product ;
#ppath: /home/merchant ;
#pdep: Ingres , 4.0.0.0 ,  >= ;
#postinit: bin/create_tables.sh ;
@ENDPROD:
```

**#postinit: bin/create_tables.sh ;**

The keyword #postinit: determines that the script create_table.sh is executed after the product files have been installed.

**To specify the script file create_table.sh through the command line**

1. Copy the script file create_tables.sh into the /tmp/merchant/scripts folder.
2. Execute the pifproto command as follows:
   `pifproto -n MERCHANT -v 2.1.0.0 -a Any -d /tmp/merchant -b /home/merchant -i scripts/create_tables.sh -o`

   Through the -i option the post-install procedure is defined. The -o switch enables overwriting the existing prototype file of the product MERCHANT.

**To add a Procedure through the GUI**

1. Run the smgui command to launch the Packager GUI.
2. Select the product MERCHANT in the Project pane.
3. Select component1 on the Components tab.
4. Choose the File list property sheet.
5. Right-click within the File list property sheet and select Insert from the context menu.

   The Choose source file dialog appears.
6. Browse for the file create_tables.sh and click Choose.
7. In the file list, double-click the new file. On the appearing dialog, select the Access property sheet, check all execute check boxes, uncheck the write check boxes, and click Change.
8. Select the product MERCHANT in the Project pane.
9. Select the Procedures tab in the work pane.
10. Right-click in the procedures list and select Add internal procedure from the context menu.

    The Internal procedure dialog appears.
11. Specify or select the following values:

    Execution time: post installation

    Procedure file: create_tables.sh
12. Click Add.

    The procedure appears in the procedures list and the properties are incorporated in the prototype file of the PIF product MERCHANT.

# Change the Installation Path of the PIF Product MERCHANT

Up to now, the installation path of the PIF product MERCHANT is fixed; it cannot be changed at installation time. Now, we will define a parameter $PATHmerchant that lets you change the installation path.

The parameter must be defined in the parameter section (default section) of the prototype file, as follows:

```
@DEFAULT:
#parameter: $PATHmerchant , /home/merchant ;
@ENDDEFAULT:
```

**#parameter: $PATHmerchant , /home/merchant ;**

The name of the parameter is $PATHmerchant; its default value is set to /home/merchant.

Replace the absolute value /home/merchant for the installation path in the #ppath keyword with the parameter $PATHmerchant in the product information section of the prototype file, as follows:

```
@PRODUCT:
#phead: MERCHANT , 2.1.0.0 ;
#sys: Any ;
#locale: ENU ;
#comment: This is an example for a PIF product ;
#ppath: $PATHmerchant ;
#pdep: Ingres , 4.0.0.0 ,  >= ;
#postinit: bin/create_tables.sh ;
@ENDPROD:
```

Now, you are able to change the installation path.

One way to do this is using a shell parameter and overwriting the default setting, for example:

```
PATHmerchant=/opt/merchant; export PATHmerchant
```

Another way to change the installation path is using a response file to reconfigure the product installation.

The command line interface and the Packager GUI add the actual installation path automatically to the PIF product. No actions are necessary.

# Add a Dialog to Query the Installation Path of the PIF Product MERCHANT

In the prototype file, dialog definitions consist of two main parts, the dialog section and the resource section. In the dialog section, the layout of the dialog is defined. The resource section stores all text strings used to present the dialog and assigns numbers to the texts. Only these text numbers are used in the dialog definition.

For the sample PIF product MERCHANT, we want to create a dialog to enter the installation directory. This dialog should be the first dialog being presented when the product MERCHANT is installed.

The definition of this dialog in the dialog section is as follows:

```
@DIALOG:
#dhead: 600 , 400 , dlgInstallationDirectory , 1 ;
#label: 1 , 1 , lblInstallDirectory , 2 ;
#instdir: 1 , 2 , tfInstallationDirectory , $PATHmerchant ;
#navbutton: 1 , btnInstall , 3 , 1 ;
#navbutton: 2 , btnCancel , 4 , 0 ;
@ENDDIALOG:
```

**#dhead: 600 , 400 , dlgInstallationDirectory , 1 ;**

The width of the dialog is 600 pixels and the height is 400 pixels. The name of the dialog is dlgInstallationDirectory, and text number 1 in the resource section of the prototype file is displayed as the dialog heading.

**#label: 1 , 1 , lblInstallDirectory , 2 ;**

Defines the label of the installation directory input field. The name of the label is lblInstallationDirectory and the text number 2 in the resource section is assigned. The position of the label in the virtual grid layered over the dialog is logical row 1, column 1.

**#instdir: 1 , 2 , tfInstallationDirectory , $PATHmerchant ;**

Defines the installation directory input field. This field is presented in column 2 in the logical row 1. The value entered in the installation directory input field is assigned to the parameter $PATHmerchant.

**#navbutton: 1 , btnInstall , 3 , 1 ;**

#navbutton: 2 , btnCancel , 4 , 0 ;

Define buttons that allow navigating through the installation process.

Selecting the first button, btnInstall, installs the product. The text on the button is contained in text number 3 in the resource section. The action performed when the button is selected, is “Install the product” (1).

Selecting the second button, btnCancel, aborts the installation. The text on the button is contained in text number 4 in the resource section. The action performed when the button is selected, is “Abort the installation” (0).

The texts for this dialog in the resource section of the prototype file are as follows:

```
@RESOURCE:
#locale: ENU ;
#text: 1 , Select installation directory ;
#text: 2 , Installation directory: ;
#text: 3 , &Install ;
#text: 4 , &Cancel ;
@ENDRESOURCE:
```

**#locale: ENU ;**

Defines the language of the text strings, in this case, ENU defines English (U.S.). (Refer to the description of the language identifiers (see page 151)).

**#text: 1 , Select installation directory ;**

#text: … , ……. ;

Define the text numbers used in the dialog definition. For example, the text "Select installation directory" is assigned the text number 1, which is used to define the dialog heading in the dialog section of the prototype file.

The ampersands characters (&) in the text strings number 3 and 4 determine that the capital letters I and C) are used as shortcuts for the Install and Cancel actions.

To specify that the Select installation directory dialog is the first dialog presented when installing the product MERCHANT, use the keyword #dlgpreinit in the product information section in the prototype file, as follows:

```
@PRODUCT:
#phead: MERCHANT , 2.1.0.0 ;
#sys: Any ;
#locale: ENU ;
#comment: This is an example for a PIF product ;
#ppath: $PATHmerchant ;
#pdep: Ingres , 4.0.0.0 ,  >= ;
#postinit: bin/create_tables.sh ;
#dlgpreinit: dlgInstallationDirectory ;
@ENDPROD:
```

**#dlgpreinit: dlgInstallationDirectory ;**

Specifies that the dialog with the name dlgInstallationDirectory appears as the first installation dialog.

The command line interface and the Packager GUI add the installation dialog automatically to the PIF product. No actions are necessary.

# Add Language-specific Product Files to the PIF Product MERCHANT

The PIF product MERCHANT prints localized messages in English and German. Therefore, the text files merchant.enu and merchant.deu are added to the product.

These text files are added to language-specific components. A language-specific component is installed only if the language specified in the component matches the target system locale ($LANG). In the language-specific component the language is specified through a property of the keyword #locale, in this example, ENU and DEU.

The following section in the prototype file specifies the language-specific component that contains the English text file:

```
@COMPONENT:
#chead: lang_spec ;
#locale: ENU ;
#file: bin/merchant.enu , /myhomepath/src/merchant.enu , root , sys , -r--r--r-- ,
1 ;
@ENDCOMP:
```

The component's name is lang_spec and it contains the merchant.enu text file. This component is installed if the target system locale has a value representing the English language (like en_US).

The following section in the prototype file specifies the language-specific component that contains the German text file:

```
@COMPONENT:
#chead: lang_spec ;
#locale: DEU ;
#file: bin/merchant.deu , /myhomepath/src/merchant.deu , root , sys , -r--r--r-- ,
1 ;
@ENDCOMP:
```

The component has the same name lang_spec as the English version, but differs by the property of the #locale keyword. This component contains the merchant.deu text file, and is installed if the target system locale has a value representing the German language.

Language-specific resources can be provided as external product enhancements. Initially, the product is built with its origin text resources. Later on, the language-specific text resources can be extracted from the product using the pifextract command, as follows:

```
pifextract -f MERCHANT.Any.@pif -l ENU -L DEU -d german_folder
```

This command extracts all language-specific components into the folder german_folder, and then prepares and creates an applicable prototype file. After replacing the text resource files and adapting the applicable #file entries in the prototype file, the external PIF product can be built with the pifmk command. As a result, the PIF file MERCHANT.@deu is created.

This PIF file must be copied to the folder of the main PIF product.

# Localize the Installation Dialogs for the PIF Product MERCHANT

Since the actual dialog texts are separated from the dialog definition in the prototype file, it is easy to create a new resource section for another language. It is just copying and changing an existing resource section. In this example, copy and paste the original resource section in the prototype file, translate the English text strings into German, and change the #locale property from ENU to DEU to specify the new language.

The German resource section of the PIF product MERCHANT is as follows:

```
@RESOURCE:
#locale: DEU ;
#text: 1 , Auswahl des Installationsverzeichnisses ;
#text: 2 , Installationsverzeichnis: ;
#text: 3 , &Installieren ;
#text: 3 , &Abbrechen ;
@ENDRESOURCE:
```

## Localize the Installation Dialogs through the Command Line

Use an editor to add language-specific resource sections to the prototype file.

**To create a resource section for another language in the prototype file**

1. Copy the existing original resource section.
2. Translate the text strings.
3. Change the property in the #locale keyword to specify the new language.

## Localize the Installation Dialogs Through the GUI

You can localize the installation dialogs through the GUI.

**To create localized text strings for the installation dialogs**

1. Run the smgui command to launch the Packager GUI.
2. Select the PIF product MERCHANT in the project pane.
3. Select and right-click the English/US entry on the Dialog resources tab in the subproject pane.
4. Right-click the selected resource and select Copy from the context menu.

   The Product resource dialog appears.

5. Choose the desired language German from the drop-down menu.
6. Click Copy.

   The German language entry appears on the Dialog resources tab in the subproject pane.

   To translate the texts in the German dialog resource proceed as follows:
7. Select the German language entry on the Dialog resources tab.
8. Double-click a text in the Dialog texts table and type the German translation of the text directly into that line. Perform this step for each text string to translate.

# Build the Final PIF Product MERCHANT

Building on the previous sections, the prototype file MERCHANT.Any.@prm for the sample PIF product MERCHANT should look like this:

```
@PRODUCT:
#phead: MERCHANT , 2.1.0.0 ;
#sys: Any ;
#locale: ENU ;
#comment: This is an example for a PIF product ;
#ppath: $PATHmerchant ;
#pdep: Ingres , 4.0.0.0 ,  >= ;
#postinit: bin/create_tables.sh ;
#dlgpreinit: dlgInstallationDirectory ;
@ENDPROD:

@COMPONENT:
#chead: component1 ;
#dir: bin , root , sys , drwxrwxrwx ;
#file: bin/merchant.jar , /myhomepath/src/merchant.jar , root , sys , -r--r--r-- ,
1 ;
#file: bin/run_merchant.sh , /myhomepath/scripts/run_merchant.sh , root , sys ,
-r-xr-xr-x , 1 ;
#file: bin/create_tables.sh , /myhomepath/scripts/create_tables.sh , root , sys ,
-r-xr-xr-x , 1 ;
#slink: /usr/bin/run_merchant.sh , bin/run_merchant.sh ;
@ENDCOMP:

@DEFAULT:
#parameter: $PATHmerchant , /home/merchant ;
@ENDDEFAULT:
```

```
@DIALOG:
#dhead: 600 , 400 , dlgInstallationDirectory , 1 ;
#label: 1 , 1 , lblInstallDirectory , 2 ;
#instdir: 1 , 2 , tfInstallationDirectory , $PATHmerchant ;
#navbutton: 1 , btnInstall , 3 , 1 ;
#navbutton: 2 , btnCancel , 4 , 0 ;
@ENDDIALOG:

@RESOURCE:
#locale: ENU ;
#text: 1 , Select installation directory ;
#text: 2 , Installation directory: ;
#text: 3 , &Install ;
#text: 4 , &Cancel ;
@ENDRESOURCE:

@COMPONENT:
#chead: lang_spec ;
#locale: ENU ;
#file: bin/merchant.enu , /myhomepath/src/merchant.enu , root , sys , -r--r--r-- ,
1 ;
@ENDCOMP:

@COMPONENT:
#chead: lang_spec ;
#locale: DEU ;
#file: bin/merchant.deu , /myhomepath/src/merchant.deu , root , sys , -r--r--r-- ,
1 ;
@ENDCOMP:

@RESOURCE:
#locale: DEU ;
#text: 1 , Auswahl des Installationsverzeichnisses: ;
#text: 2 , Installationsverzeichnis: ;
#text: 3 , &Installieren ;
#text: 4 , &Abbrechen ;
@ENDRESOURCE:
```

To build the PIF product MERCHANT through the command line enter the pifmk command in the directory where the prototype file is located, as follows:

```
pifmk -f MERCHANT.Any.@prm
```

The PIF product MERCHANT is built in the current working directory; its PIF file name is MERCHANT.Any.@pif.

**To build the PIF Product MERCHANT using the Packager GUI**

1. Run the smgui command to launch the Packager GUI.
2. Select the PIF product MERCHANT in the project pane.
3. Choose Build from the toolbar.

   The Build product dialog appears.

   For this sample PIF product ignore the Strip and Self installing check boxes.

   If the product MERCHANT already exists, another check box appears that you can check to overwrite the existing product.
4. Click Build.

   The PIF product MERCHANT.Any.@pif appears in the Media table.

# Chapter 6: Installer Tasks

The Installer for Linux and UNIX (Installer) performs a number of tasks before and after the actual install process of a PIF product on the target computer. The tasks performed before the PIF product is installed are called pre-install tasks, the tasks after the actual product installation tasks are called post-install tasks.

The Installer performs the following pre-install tasks:

- Validate that the software is complete
- Extract pre-install components to a temporary folder
- Execute the migration script (only when a new product is installed)
- Perform dialogs to retrieve necessary parameters
- Create a backup from an installed product (in case of an update installation)

The Installer performs the following install tasks (actual product installation):

- Read parameter values from the environment.
- Install external pre-install components.
- Execute pre-install scripts.
- Install the components of the PIF product.
- Install external post-install components.
- Execute post-install scripts.
- Register product information in the local database.

If an update installation of a PIF product fails, the Installer restores the backup of the installed PIF product as a post-install task, to reinstall the previous product version.

This section contains the following topics:

Pre-Install Tasks (see page 100)
Tasks During Product Installation (see page 102)
Post-Install Task (see page 104)

# Pre-Install Tasks

The pre-install tasks described following are automatically performed by the Installer before the PIF product is installed.

When large PIF products are installed, the installation preparation phase may take some time, while the Installer is busy executing pre-install scripts, checking product state and disk structure, without providing any visible output for the user. During this phase it is essential to inform the user that the installation still continues running.

The keyword #splashscreen lets you specify a graphic in GIF format and an optional text to display. This text is shown on the console, in the line following the input command line, for example:

```
lsm -i NSM.Linux.@pif

Preparing installation: ........
```

Progress is shown only within an interactive installation that runs an installation interview phase.

## Validate the Software

As the first task, the Installer validates that all software required for the PIF product installation is part of the installation media (DVD, CD-ROM). For example, when external components are referenced and are not already installed, the PIF product files containing those components must exist in the same folder as the main PIF product. Also, required PIF products that are not installed must exist in the same folder as the main PIF product.

## Extract Pre-install Components to a Temporary Folder

Before installation dialogs and scripts are executed, the product pre-install components are extracted into a temporary folder.

If product scripts need to call scripts of other required products of the same installation transaction they need to know the name of this temporary folder. Therefore, the parameter $PIF_PREINSTALL_DIR provides the name of the temporary pre-install folder.

Pre-install components are stored at the following location:
`$PIF_PREINSTALL_DIR/`*`prodname`*`.d/\$PRODUCT_PATH`

where $PRODUCT_PATH contains the product specific installation path parameter.

**Example:**

If PIF_PREINSTALL_DIR is set to /tmp/EXTRACT12345/, the scripts of the Packager product are located at:
`/tmp/EXTRACT12345/SMPACKAGER.d/\$SDPcker/scripts`

**Note:** Masked parameters (such as abc\$efg) referring to an intermediate product installation directory are preserved now.

## Execute a Migration Script

To migrate non-PIF product installations, you can provide a migration script, which is executed by the Installer prior to the first installation of the product. You must specify the migration script for the PIF product through the keyword #prepif in the Product information section of the prototype file.

With the migration script, you can change your product-specific installation path parameter that has been defined through #ppath (for example, *$MY_PIF_INSTALL_PATH*) and the installation job *($PIF_INSTALL_JOB*).

PIF parameters are passed through the shell environment and a temporary response file. The PIF product installation process recognizes updated parameters, which are appended to the response file, later on.

### Example: Migration Script

The following script checks old installation base directories, sets the installation directory, and changes the installation mode from INSTALL to UPDATE**.**

```
##  check if response file parameter is passed
[ ! "$PIF_RESPONSE_FILE" -o ! -s "$PIF_RESPONSE_FILE" ] && exit 1
## check non PIF installation: product is either installed under /home or /opt
directory
if [ -d /home/merchant ]
then
        echo "MY_PIF_INSTALL_PATH=/home/merchant" >> $PIF_RESPONSE_FILE
        echo "PIF_INSTALL_JOB=UPDATE" >> $PIF_RESPONSE_FILE
elif [ -d /opt/merchant ]
then
        echo "MY PIF_INSTALL_PATH=/opt/merchant" >> $PIF_RESPONSE_FILE
        echo "PIF_INSTALL_JOB=UPDATE" >> $PIF_RESPONSE_FILE
fi
exit 0
```

## Run Installation Dialogs

The Installer runs the installation dialogs defined for the PIF product. From these dialogs, the Installer gets the parameter values to configure the installation of the PIF product.

When the target computer has the Java Runtime Environment (JRE) 1.4 installed, and the monitor is graphics-enabled, the installation dialogs appear in the graphical mode (Java GUI). If these prerequisites are not met on the target computer, the installation dialogs appear in a VT100 compatible mode.

You can include a JRE in a self-installing PIF product, that will then install and use the included JRE to run the Java wizard of the Installer on the target computer.

## Run Post-Interview Script

The Installer processes the script after all installation dialogs have been run. The script can be used to modify the currently configured installation job. The script is processed at each installation phase (install, update, or reinstall) and also in unattended mode.

In the prototype file, the script is specified through the #postinterview keyword. The post-interview script must be added to the pre-install component.

## Create a Backup

When a PIF product is being updated, the Installer automatically creates a temporary backup of the installed product, which is restored in the case the update installation fails.

The internal parameter PIF_BACKUP controls backup creation prior to updating or reinstalling an existing product version.

Backup creation can be switched off using the lsm command with the -F switch.

# Tasks During Product Installation

The tasks the Installer performs when installing a PIF product on the target computer are described following.

## Read Parameter Values from the Environment

Parameters configure the installation of the PIF product. They define installation paths and include or exclude components from the installation process. Product-specific parameters can also be passed to the installation. These parameters are used inside the installation procedure scripts.

The PIF product installation resolves parameters in the following sequence:

1. PIF parameters passed by a response file are read. The user can specify a response at installation time using the lsm command with the -r switch. Parameters collected in an installation dialog are also passed to the installation process through a temporary response file.
2. Environment variables overwrite response file parameters in case of an unattended installation. Environment variables must be set before the PIF product installation starts.

## Execute Pre-Install Scripts

If you have specified pre-install scripts in the prototype file, these are installed now. Pre-install scripts are defined for the PIF product or for a component of the PIF product, and must be executed before the installation of files and directories starts. A pre-install script ensures the success of the installation process, for example, it stops running daemon processes or checks the installation environment. (Have disks been shared? Are scalability servers available?)

Parameters are passed as shell parameters to all (pre- and post-) install scripts. New or updated parameters have to be appended to the response file.

### Example: Script to check for server availability

The following script checks whether the server, whose name has been passed by the $SERVERNAME parameter, is available. If so, the script activates the component installation flag (#cinstall=$INSTALL_SERVER in the component section of the prototype file).

```
##  exit if response file has not been passed
[ ! "$PIF_RESPONSE_FILE" -o ! -s "$PIF_RESPONSE_FILE" ] && exit 1

## exit if the server name has not been passed
[ ! "$SERVERNAME" ] && exit 1

## server responds, activate the server component
if [ 'ping $SERVERNAME 2>/dev/null; echo $?' -eq 0 ]
then
    echo "INSTALL_SERVER=1" >> $PIF_RESPONSE_FILE
```

```
else
    echo "INSTALL_SERVER=0" >> $PIF_RESPONSE_FILE
fi
```

## Install Components

The main task of the Installer is installing the components of the PIF product. The contents of the components, that is, the files, directories, and symbolic links are installed on the target computer in the following sequence:

1. All directories of all components are created.
2. All files are installed.
3. Symbolic links are created.

## Run Post-Install Scripts

After external post-install components have been installed, the Installer executes post-install scripts, if specified in the prototype file. Post-install scripts are defined for the PIF product or for a component of the PIF product. Post-install scripts typically configure and activate the installed PIF product, for example, they start daemon processes or create database tables.

## Register Product Information

Registering the PIF product in the target computer's local PIF database completes the main installation tasks of the Installer. Parameter values are registered together with the PIF product. The registration of the PIF product guarantees that checking, querying, and removing the PIF product will work correctly.

# Post-Install Task

If an update installation of a PIF product fails due to an error, the Installer automatically restores the previously installed PIF product version from the backup.

# Chapter 7: Reference

Detailed descriptions of conventions, standard templates, file formats and structures, Packager and Installer commands, and supported operating environments and languages follow.

This section contains the following topics:


File Naming Conventions (see page 105)
Prototype File Format and Structure (see page 106)
Keywords and Properties of the Prototype File (see page 111)
Packager and Installer Commands (see page 131)
Install a Self-Installing PIF Product (see page 148)
Supported Linux and UNIX Operating Environments (see page 150)
Language Identifiers (see page 151)
Template Parameters and pifproto Command Switches (see page 152)


## File Naming Conventions

When you use the PIF SDK to create prototype files and PIF products, you must meet file naming conventions, as described following.

**Prototype file name**

The file name of the prototype file is as follows:
*prodname*.*platform*.@prm

**PIF product file name**

The file name of the PIF product is as follows:
*prodname*.*platform*.@pif

**Prototype file name for a language resource**

The file name of the prototype file for a language resource is as follows:
*prodname*.*languageID*.@prm

In this file name, *languageID* is a three-character language identifier (see page 151), for example, ENU (English (U.S.), DEU (German), FRA (French).

**PIF product file name for a language resource**

The file name of the PIF product for a language resource is built as follows:
*prodname*.@*languageID*

In this file name, *languageID* is a three-character language identifier (for example, ENU (English (U.S.), DEU (German), FRA (French)).

**Delta product file name**

The file name of the delta product (which contains the files that differ between a prior version and a resulting version of the same PIF product) is built as follows:
*prodname*.*platform*.*priorversion-resultingversion*.@pif

When you create a delta product from the Packager GUI, its file name is automatically generated in this format. We recommend following this file naming convention also when creating the delta product with the pifdelta command, although it is not mandatory.

# Prototype File Format and Structure

The prototype file is an ASCII file used to specify information about the PIF product. The information is bundled in sections, depending on the purpose and the contents of the information. A section can be omitted when it is not relevant for the PIF product. The prototype file can contain the following sections:

- Product information section
- Component section
- Parameter section (Default section)
- Dialog section
- Resource section

The prototype file is created using the pifproto command or by entering appropriate values through the Packager GUI. The name of the prototype file is created as follows:
*prodname*.*platform*.@prm

Following notation conventions apply to prototype files for PIF products:

- All entries in the prototype file are structured as follows:
  #*keyword*: *property1* , *property2* , ... ;

  Each keyword is lead by a hash sign (#), and terminated by a colon (:).

  Each property is separated from the subsequent property by a space-comma-space sequence ( , ).

  Properties shown in square brackets [ ] are optional and can be omitted. In lists of properties you must keep the separating space-comma-space sequences for compiler reasons, for example, if properties prop2, prop4, and prop5 are optional:
  #keyw: prop1 , , prop3 , , , prop6 ;
- Each entry is terminated by a space-semicolon sequence ( ;).
- Entries must not extend over several lines.

- Entries shown in square brackets [ ] are optional. Optional entries can be entered without values. You can also omit the complete entry.
- Comments must start with a double slash (//), or start with the characters slash asterisk (/*) and end with the characters asterisk slash (*/).
- Entries in the prototype file must not contain any control character, like carriage return (CR) or line feed (LF).

## Product Information Section

The product information section is framed by the keywords @PRODUCT and @ENDPROD. This section contains general information about the PIF product and product-related procedures. The product information section consists of the entries listed below. For the detailed description of an entry see the appropriate keyword description.

```
@PRODUCT:

#phead: ..... ;
#sys: ..... ;
#ppath: ..... ;
#locale: ..... ;
[#build: ..... ;]
[#supplier: ..... ;]
[#comment: ..... ;]
[#productfamily: ..... ;]
[#frameheader: ..... ;]
[#patchproduct: ..... ;]
[#replacepatches: ..... ;]
[#vname: ..... ;]
[#admininst: ..... ;]
[#multiinst: ..... ;]
[#timer: ..... ;]
[#prepif: ..... ;]
[#prestart: ..... ;]
[#postinterview: ..... ;]
[#preinitcomp: ..... ;]
[#preventdowngrade: ..... ;]
[#pdep: ..... ;]
[#dis: ..... ;]
[#preinit: ..... ;]
[#postinit: ..... ;]
[#rmpreinit: ..... ;]
```

```
[#rmpostinit: ..... ;]
[#extpreinit: ..... ;]
[#extpostinit: ..... ;]
[#extrmpreinit: ..... ;]
[#extrmpostinit: ..... ;]
[#dlgcancel: ..... ;]
[#dlgend: ..... ;]
[#dlgendrm: ..... ;]
[#dlgerror: ..... ;]
[#dlgerrordisk: ..... ;]
[#dlgerrorpath: ..... ;]
[#dlgnextdisk: ..... ;]
[#dlgnextpath: ..... ;]
[#dlgpreinit: ..... ;]
[#dlgpreupd: ..... ;]
[#dlgprogress: ..... ;]
[#splashscreen: ..... ;]
[#postcheck: ]

@ENDPROD:
```

## Component Section

The component section is framed by the keywords @COMPONENT and @ENDCOMP. This section describes a component, which is a small installation unit within the PIF product. The component section consists of the entries listed below. For the detailed description of an entry see the appropriate keyword description.

```
@COMPONENT:

#chead: ..... ;
[#sys: ..... ;]
[#locale: ..... ;]
[#cinstall: ..... ;]
[#cpath: ..... ;]
[#preinit: ..... ;]
[#postinit: ..... ;]
[#rmpreinit: ..... ;]
[#rmpostinit: ..... ;]
[#extpreinit: ..... ;]
[#extpostinit: ..... ;]
[#extrmpreinit: ..... ;]
[#extrmpostinit: ..... ;]
[#file: ..... ;]
[#dir: ..... ;]
```

```
[#slink: ..... ;]
[#reservedspace: ..... ;]

@ENDCOMP:
```

## Default Section (Parameter Section)

The default section (or parameter section) is framed by the keywords @DEFAULT and @ENDDEFAULT. This section defines product-specific parameters and their default value, using the keywords #parameter and #globalparameter, as shown below. For the detailed description of an entry see the appropriate keyword descriptions.

The Installer uses the default values as initial values in installation dialogs, or for an unattended product installation.

```
@DEFAULT:

[#sys: ..... ;]
#parameter: ..... ;

@ENDDEFAULT:
```

## Dialog Section

The dialog section is framed by the keywords @DIALOG and @ENDDIALOG. This section defines the layout of the dialogs executed during an attended product installation. The dialog section consists of the entries listed below. For the detailed description of an entry see the appropriate keyword description.

The same definitions are used for Java- and VT100-based dialogs. The Installer manages the proper visualization on the target computer.

```
@DIALOG:

#dhead: ..... ;
[#graphic: ..... ;]
[#textfield: ..... ;]
[#passwordfield: ..... ;]
[#instdir: ..... ;]
[#textarea: ..... ;]
[#filearea: ..... ;]
[#label: ..... ;]
[#checkbox: ..... ;]
[#button: ..... ;]
```

```
[#navbutton: ..... ;]
[#combobox: ..... ;]
[#radiobutton: ..... ;]
[#progressbar: ..... ;]
[#progresslabel: ..... ;]
[#progressarea: ..... ;]
[#fileselection: ..... ;]
[#scrollarea_begin: ..... ;]
[#scrollarea_end: ..... ;]
[#tree_begin: ..... ;]
[#tree_end: ..... ;]
[#image:]
[#fileselection:]
[#inputorder:]

@ENDDIALOG:
```

## Fontlist Section

The Fontlist section is framed by the keywords @FONTLIST and @ENDFONTLIST. This section contains font definitions for installation dialogs. The Installer allows changing font properties like size, bold, italic, color, and also Java logical fonts like Serif, SansSerif, Monospaced, Dialog, and DialogInput. For the detailed description of the #font entry see the keyword description.

```
@FONTLIST:

#font: ..... ;

@ENDFONTLIST:
```

## Desktop Section

The Desktop section is framed by the keywords @DESKTOP and @ENDDESKTOP. In this section you can specify menus and desktop icons for the PIF Packager.

For the detailed description of the entries see the appropriate keyword description.

```
@DESKTOP:

[#desktopicon: ..... ;]

[#desktopmenu: ..... ;]

@ENDDESKTOP:
```

## Resource Section

The resource section is framed by the keywords @RESOURCE and @ENDRESOURCE. This section contains the language-specific text strings for the PIF product. The text strings are used, for example, with the installation dialogs. They are referred to by a text identification number. The locale entry specifies the default language of the PIF product.

```
@RESOURCE:

#locale: ..... ;
#text: ..... ;
#utf8encoded: ..... ;

@ENDRESOURCE:
```

**Note:** Internally, the text strings are formatted in multi-byte character mode. During PIF product creation these characters are converted into UTF8 code. At installation time, the Installer converts the characters back to multi-byte character mode.

# Keywords and Properties of the Prototype File

Keywords and properties are used in the prototype file to specify all components, features, files, and parameters of the PIF product. Keywords are lead by a hash sign (#).

The detailed product specifications are defined through properties that follow each keyword.

## Keywords

All keywords that can be used in the different sections of the prototype file for a PIF product are listed in alphabetical order.

**#admininstall: *flag* ;**

Specifies whether the product should be installed with root user privileges (flag value = True (1)) or with non-root user privileges (flag value = False (0)).

**#build:[ *buildinformation*] ;**

Contains information about the product build for user purposes only, for example, to identify different packages of the same product version. This information is not checked by the Installer.

**#button: *row* , *column* , *ctrlname* , *textID* , *action* , [*icon*] , [*nextdialog*] , [*paramset*] , [*fontname*] , [*resultpane*] ;**

Specifies a button and its position within the input area of an installation dialog. The *textID* option specifies the text that is displayed. *Nextdialog* is the name of the next dialog or the name of a script that performs an action, for example, dynamically detects the dialog name. An icon can be specified for the button. The *paramset* option specifies a list of parameters that are set when the button is pressed. The *action* is a number that specifies an action to be initiated when the button is selected (for example, 2 = Show next dialog. The complete list of values of *action* can be found in the Prototype File Properties section under the *action* property.) The result of a script execution (*action* = 9) can be displayed in a result pane; the *resultpane* parameter can specify a progress area or a text area.

**#chead: *compname*[ , *version*] ;**

Specifies the name and version of a component of the PIF product.

**#checkbox: *row* , *column* , *ctrlname* , *paramname* , [*actparamset*] , [*deactparamset*] , [*textID1*] , [*textID2*] , [*dependencylist*] , [*actionscript*][ , *fontname*] ;**

Specifies a check box and its position within the input area of an installation dialog. The environment variable paramname contains the state of the check box. If the check box is checked (True), the value of the environment variable is set to one (1); otherwise (False), the variable is set to zero (0). Two lists of comma-separated parameter=value assignments can be specified that are activated when the check box is checked (*actparamset*) or cleared (*deactparamset*). The *textID1* option specifies label text that is displayed on the right side of the check box. The textID2, dependencylist, and actionscript options are for use in check box trees only. The *textID2* option specifies additional descriptive text for an item selected in the tree, dependencylist specifies a comma-separated list of check box ctrlnames, and actionscript specifies a script that changes field attributes or contents in the installation dialog immediately.

**#cinstall: cinstparam ;**

Specifies whether the component should be installed with the product or only on demand (that is, by selecting the component from an installation dialog.) In the latter case the component is called optional.

**#combobox: *row* , *column* , *ctrlname* , *flag* , *listparam* , *resultparam* , [*dialogvalidationscript*] , [*actionscript*][ , *fontname*] ;**

Presents a combo box or a list box, depending on the *flag* value. When the flag value is False (0), a List box is presented; when the flag value is True (1), a Combo box is presented. Both types of boxes provide a list of possible values and allow selecting one of them. In the combo box, the user can alternatively enter another value. A shell script can be specified that validates the input, and another script (*actionscript*) that changes field attributes or contents in the installation dialog immediately.

**#comment: *textstring* ;**

Specifies any comment, for example, to describe the product in detail. The first 127 bytes of the text string are displayed as a comment in the DSM Explorer.

**#cpath: *comppath* ;**

Specifies the installation path of a component of the PIF product.

**#desktopicon: *textID1 , targetpath , argument , iconpath , textID2*[ , *userinstall*][ , *allinstall*] ;**

Specifies a desktop icon for the PIF Packager. *textID1* is the number of a text that is displayed with the icon, *textID2* is the number of a text containing localized comment. The parameters userinstall and all install indicate whether the icon should be installed for the current user or for all users.

**#desktopmenu: *textID1 , targetpath , argument , iconpath , textID2*[ , *userinstall*][ , *allinstall*] ;**

Specifies a desktop menu for the PIF Packager. *textID1* is the number of a text that is displayed with the menu, *textID2* is the number of a text containing localized comment. The parameters userinstall and all install indicate whether the menu should be installed for the current user or for all users.

**#dhead: *width , height , ctrlname , textID*[ , *dialogvalidationscript*][ , *fieldwithfocus*] ;**

Specifies the name, position, and headline of an installation dialog. A shell script can be specified that validates the input. You can also define a dialog element (field) that gains the focus when the installation dialog is started.

**#dir: *target , ownerID , groupID , access* ;**

Specifies the directories that relate to the current component.

**#dis:[ *prodname , version*[ , *operator*]] ;**

Specifies an incompatible dependency. Enter all those products, together with their versions, which must not be installed on the target computer when this product is installed. You must use one #dis: entry for each dependent software product. If there is no incompatible dependency, leave this entry blank, or omit the entry.

**#dlgcancel: *dlgref* ;**

Specifies the dialog that is shown whenever the installation of the PIF product is canceled by the user.

**#dlgend: *dlgref* ;**

Specifies the dialog that is shown after the product installation has successfully finished.

**#dlgendrm: *dlgref* ;**

Specifies the dialog that is shown when the product removal has been completed.

**#dlgerror: *dlgref* ;**

Specifies the dialog that is shown when an error occurs during installation of the PIF product.

**#dlgerrordisk: *dlgref* ;**

(Multiple disk support only) Specifies the dialog that is shown when the next inserted disk does not match.

**#dlgerrorpath: *dlgref* ;**

(Multiple disk support only) Specifies the dialog that is shown when the next inserted path does not match.

**#dlgnextdisk: *dlgref* ;**

(Multiple disk support only) Specifies the dialog that is shown to request insertion of a new disk.

**#dlgnextpath: *dlgref* ;**

(Multiple disk support only) Specifies the dialog that is shown to request a new path.

**#dlgpreinit: *firstinstalldialog* ;**

Specifies the starting dialog used when a PIF product is being initially installed.

**#dlgpreupd: *firstupdatedialog* ;**

Specifies the starting dialog used when a PIF product is being updated.

**#dlgprogress: *dlgref* ;**

Specifies the dialog that is presented to visualize the installation progress during the PIF product installation.

**#extpostinit: *externalpostinstallprocedure* ;**

Specifies an external procedure executed after PIF product installation.

**#extpreinit: *externalpreinstallprocedure* ;**

Specifies an external procedure executed prior to PIF product installation.

**#extrmpostinit: *externalpostuninstallprocedure* ;**

Specifies an external procedure executed after PIF product removal.

**#extrmpreinit: *externalpreuninstallprocedure* ;**

Specifies an external procedure executed prior to PIF product removal.

**#file: *target , source , ownerID , groupID , access , filetype* ;**

Specifies a file contained in the current component. One #file: entry is required for each file contained in the component.

**#filearea:** ***row , column , ctrlname , filename*****[ ,** ***fontname*****] ;**

Specifies an area on the installation dialog where the contents of the referenced file are displayed. Supported file extensions are: .htm, .html, .rtf, .txt for Java, and .txt for VT100 user interfaces. The contents of the file are wrapped within the file area, and scrolling is supported. Parameters can be used to dynamically set the file name. Parameters of special use are: PIF_RESPONSE_FILE, PIF_ERROR_FILE, and PIF_LOG_FILE. For example, when you specify the internal variable $PIF_RESPONSE_FILE as file name, the contents of the internally used response file can be displayed.

**#fileselection:** ***row , column , ctrlname , paramname*****[ ,** ***validationscript*****][ ,** ***fontname*****] ;**

Specifies name and position of a file name input field, and the environment variable to store the file name in. A shell script can be specified that validates the input.

**#font:** ***fontname*****[ ,** ***fontlogical*****][ ,** ***fontsize*****][ ,** ***fontbold*****][ ,** ***fontitalic*****][ ,** ***fontcolor*****] ;**

Specifies a font and font properties to be used in Java installation dialogs.

**#frameheader:** ***textID*** **;**

Specifies the window frame header in the Java installation dialogs. The textID is the number of a text string specified in the Resource section. Default value is: CA Software Management Installer.

**#graphic: [***GIFfilewest***] , [***GIFfilenorth***] , [***GIFfilecenter***] , [***GIFfilesouth***][ ,** ***GIFfileeast*****] ;**

For Java dialogs only. Specifies one or more graphics that are displayed in the according graphics area on the installation dialog. The graphics files specified must be in the GIF format.

**#image:** ***row , column , ctrlname , imagefilename*****[ ,** ***scalablewidth , scalableheight*****] ;**

Specifies an image (picture) that is displayed in an installation dialog. The parameters scalablewidth and scalableheight are optional; if they are specified, the image is scaled to the defined size.

**#instdir:** ***row , column , ctrlname , paramname*****[ ,** ***validationscript*****][ ,** ***fontname*****] ;**

Specifies name and position of the installation directory input field, and the environment variable to store the directory in. A shell script can be specified that validates the input.

**#inputorder: ctrlname ;**

Specifies the order in which the input fields of an installation dialog should be processed during the installation of the PIF package.

**#label:** ***row , column , ctrlname , textID*****[ ,** ***fontname*****] ;**

Specifies descriptive text that leads input fields, check boxes, or installation directories. The referred text string can consist of a maximum of 1024 characters, but when the label is displayed on the dialog it does not exceed one line. That means the text of the label is not wrapped within the dialog window.

**#locale: *localeID* ;**

Specifies the default language of the PIF product. The language specified by localeID is used when starting installation dialogs with their default language. (Refer to the description of the language identifiers (see page 151)).

**#multiinst: *flag* ;**

Specifies whether the product is enabled or disabled for multiple installations on the target computer. When the flag value is False (0), which is the default, no further instances of this product version can be created on the target computer. When the flag value is True (1), the PIF product can be installed multiple times (multiple instances) on the target computer.

**#navbutton: *column* , *ctrlname* , *textID* , *action*[ , *icon*][ , *dlgref*][ , *fontname*] ;**

Specifies the next installation dialog dlgref to appear when this navigation button is selected. You can specify a shell script that determines the next installation dialog. Through textID you specify text that appears in the navigation button and visualizes its function. Additionally, you can specify an icon (as a GIF file) that appears in the navigation button. The action is a number that specifies an action to be initiated when this navigation button is selected (for example, 2 = Show next dialog). In an installation dialog, you can define up to four navigation buttons in the navigation area.

**#parameter: *paramname* , *paramvalue* , *paramcomment* , *paramtype* ;**

Defines a parameter and its default value that is used to configure the product installation. The parameter value assigned can contain previously defined parameters.

Example:

```
#parameter: $installdir , /opt/CA/installpath ;
#parameter: $my_install_path , $installdir/myproduct1 ;
```

The paramcomment property is used to describe the parameter. The value of paramtype controls whether the parameter is published (value = 0), private (value = 1), or global (value = 2).

**#passwordfield: *row* , *column* , *ctrlname* , *paramname*[ , *validationscript*] ;**

Specifies the name and position of the password input field, and the environment variable to store the password in. A shell script can be specified that validates the input.

**#patchproduct: *prodname* ;**

Specifies the name of the product that this current patch is going to change.

Example: #patchproduct: common_utilities ;

**#pdep:[ *prodname* , *version*[ , *operator*][ , *instparam*]] ;**

Specifies a product dependency. Specify the product and its version, which must be installed on the target computer, before the current PIF product is installed. You must use one #pdep: entry for each product the current product is dependent on. If there is no product dependency, leave this entry blank or omit the entry.

**#phead: *prodname* , *version* ;**

Specifies the name and version of the PIF product.

**#postinit:[ *postinstallprocedure*[ , *interpreter*]] ;**

Specifies an internal procedure executed after PIF product installation. An interpreter other than the shell can be specified for the procedure.

**#postinterview: *script* ;**

Specifies the name of a script that is executed after the interview phase, that is, after all dialogs have run. The script must be added to the pre-install component.

**#ppath: *prodpath* ;**

Specifies the installation path of the PIF product.

**#preinit:[ *preinstallprocedure*[ , *interpreter*]] ;**

Specifies an internal procedure executed prior to PIF product installation. An interpreter other than the shell can be specified for the procedure.

**#preinitcomp:[ *preinstallcomponent*] ;**

Specifies a component that is installed prior to the PIF product.

**#prepif:[ *migrationscript*[ , *interpreter*]] ;**

Specifies a procedure, that is, a user-written script to migrate a non-PIF product into a PIF product. You can arrange the PIF product installation path and the PIF product install mode (UPDATE, INSTALL). The prepif procedure is executed only prior to a new installation of the PIF product; it is not executed during an update install or reinstall. An interpreter other than the shell can be specified for the script.

**#prestart:[ *prestartprocedure*[ , *interpreter*]] ;**

Specifies an internal procedure executed directly after a prepif procedure before the product installation or any installation dialog starts. This procedure is executed each time before the first installation dialog starts to define pre-settings needed by the installation dialogs. The prestart procedure is also started when no installation dialog is defined. An interpreter other than the shell can be specified for the procedure.

**#preventdowngrade: *flag* ;**

Specifies whether a downgrade to a lower product version is forbidden. If the flag value is 1 (True) the product cannot be downgraded, if the flag value is 0 (False), a product downgrade is enabled. In a transaction with product dependencies only the main product is checked, all other installed required products remain with their highest versions.

**#productfamily: *prodfamilyname* ;**

Specifies the name of the product family the PIF package belongs to. This keyword lets you update in sync all packages with the same product family name to the latest available product version. Any package can belong to a single product family.

**#progressarea: *row* , *column* , *dlgref*[ , *fontname*] ;**

Displays the progress of script execution using output texts from the script. Long text lines are not wrapped within the dialog window. The progress area is reset whenever a new script is invoked. This function is not available for RPM packages.

**#progressbar: *row* , *column* , *dlgref* , *progbartype* , *flag*[ , *fontname*] ;**

Displays the progress of the installation of a product or a component or the overall installation progress (specified through the value of progbartype). In Java dialogs, Java standard elements are used to visualize the installation progress. In VT100 mode dialogs, the progress bar has a length of 40 characters; the installation progress is visualized using the pipe character (|) and a percentage value. The dialog containing the progress bar is specified by the option dlgref. The flag value switches the percentage string in the progress bar off (value = 0 (False)) or on (value = 1 (True)).

**#progresslabel: *row* , *column* , *dlgref* , *proglabeltype*[ , *fontname*] ;**

Informs about the current install action (for example, install, backup, remove) and optionally the installation phase, the name of the file or component currently being installed, or the elapsed installation time (specified by the value of proglabeltype). The dialog containing the progress label is specified by dlgref.

**#radiobutton: *row* , *column* , *ctrlname* , *textID* , *buttongroup* , *resultparamlist* , *flag*[ , *actionscript*][ , *fontname*] ;**

Enables switching between two or more values. The option textID specifies the text to be shown with the button, and button group is the name of the group that organizes the radio buttons. Resultparamlist is list of parameter=value assignments that are activated when the radio button is selected. The flag value controls whether the radio button is selected (value = 1 (True)) or not selected (value = 0 (False)) by default. The option actionscript specifies a script that changes attributes or contents of fields in the installation dialog immediately.

**#replacepatches: *prodlist* ;**

Specifies a comma-separated list of patch packages that are replaced with the current patch. No spaces must be around the commas in this list.

Example: #replacepatches: common_utilities_patch1,network_patch7 ;

**#reservedspace: *virtualfilename* , *diskspace* ;**

Reserves disk space for the virtual file specified. The required disk space of the virtual file is specified through the diskspace option in bytes.

**#rmpostinit:[ postuninstallprocedure[ , interpreter]] ;**

Specifies an internal procedure executed after PIF product removal. An interpreter other than the standard shell can be specified for the procedure.

**#rmpreinit:[ *preuninstallprocedure*[ , *interpreter*]] ;**

Specifies an internal procedure executed prior to PIF product removal. An interpreter other than the standard shell can be specified for the procedure.

**#scrollarea_begin: *row* , *column* , *ctrlname* , [*textarea*][ , *scrollmode*] ;**

Specifies the beginning position of a scroll area within a check box tree on the installation dialog. The end of the scroll area is defined by the keyword #scrollarea_end. You can define a text area in the current installation dialog that provides additional information about the items in the scroll area. The scroll mode specification is used for VT100 only.

**#scrollarea_end: ;**

Specifies the end of the scroll area.

**#slink: *symboliclink* , *linksource*[ , *ownerID* , *groupID*] ;**

Specifies whether a file is linked to other files.

**#splashscreen: *GIFfile*[ , *textID*] ;**

Specifies a graphic in GIF format and an optional text that appear as first screen when the product installation is started in attended mode, that is, prior to any dialog of the interview phase. The graphic with the name GIFfile must be in the GIF format; textID is the number of a text string specified in the Resource section.

**#supplier:[ *vendorname*] ;**

Specifies the name of the product vendor.

**#sys: *platform* ;**

Specifies the operating environment where the PIF product can be executed.

**#text: *textID* , *textstring* ;**

Specifies a text resource, for example, to appear in an installation dialog. In the prototype file, the text string is referred to by means of the textID number only.

**#textarea: *row* , *column* , *ctrlname* , *textID*[ , *fontname*] ;**

Specifies an area on the installation dialog where the referenced text is displayed. The text is wrapped within the text area, and scrolling is supported.

**#textfield: *row , column , ctrlname , paramname*[ , *validationscript*][ , *fontname*] ;**

Specifies the name and position of a user-specific input field, and the environment variable to store the entered text in. The input data is written into a temporary response file, which is passed to the product installation. Optionally, a shell script can be specified that validates the input.

**#timer: *timeout* ;**

Specifies a time-out value for the product installation. In case you have long running scripts that may exceed the default value of the timer (10 minutes), it may be useful to specify a greater time-out value that fits the requirements of the installation script.

**#tree_begin: *ctrlname , flag* ;**

Specifies the beginning of a tree within a scroll area. The end of the tree is defined by the #tree_end keyword. You can specify a text area that provides additional information about the items in the scroll area. In this case the text area must be defined in the current installation dialog. The flag specifies the initial shape of the tree and has the values: False = 0 meaning Tree is initially collapsed, and True = 1 meaning Tree is initially expanded. Default value is False (0).

**#tree_end: ;**

Specifies the end of the tree within a scroll area.

**#utf8encoded: 1 ;**

Specifies that text strings have to be entered in UTF-8 encoded format. If the keyword is not specified, the text strings are entered in MBCS code.

**#vname: *prodname , version , flag* ;**

Specifies a virtual package prodname that will be substituted by this current package. The specified version is the lowest version of the virtual package. During installation the version number is verified using the >= (greater than or equal to) operator. The flag indicates if the original package (or another virtual package) should be removed after installation. Flag value 0 specifies "Do not remove", flag value 1 specifies "Remove". Default is 1.

## Properties

All properties that can be specified in connection with keywords in the prototype file for PIF products are listed in alphabetical order.

**access**

Specifies a 10-byte access rights string, for example, –rwxr–xr–x.

**Action**

Indicates the type of action that is performed when the button is clicked. This property can have one of the values mentioned in the following table:

| Numerical value | Descriptive Text / Action performed |
|---|---|
| 0 | Cancel installation. |
| 1 | Install product. |
| 2 | Next dialog. Show the next dialog. |
| 3 | Remove product. |
| 4 | Show previous dialog. |
| 5 | Dynamic dialog. Determine next installation dialog dynamically by evaluating a script. |
| 6 | End installation and start a user script. |
| 7 | Go to another installation dialog and take origin of that dialog. |
| 8 | Start script and continue. The current dialog is closed, the specified script is executed, and the initial dialog is started again. |
| 9 | Execute script. The specified script is executed and the result is shown in the specified result pane, which can be a progress area or text area. |

**actionscript**

Specifies the name of a script that changes attributes or contents of fields in an installation dialog, for example, activate or deactivate a field.

**actparamset**

Is a comma-separated list of parameter=value assignments that are activated when the check box is checked.

**buildinformation**

Is a text string consisting of a maximum of 1024 alphanumeric characters, for example, FINAL BUILD. The text string must not contain any control characters like carriage return (CR) and line feed (LF).

**buttongroup**

Specifies the name of the group that organizes the radio buttons.

**cinstparam**

Specifies the name of the parameter that specifies whether the component should be installed with the product or only on demand (optional component). If the value of the parameter is 0, the component is not installed with the product. By default, the component will be installed. (See the property paramname for information about the syntax of a parameter name.)

**column**

Specifies the column number of the dialog element within the virtual grid of 4 columns and 10 rows. The value range is 1 – 4.

**compname**

Specifies the name of a component, which is a small installable unit within the PIF product. The name can consist of a maximum of 128 characters.

**comppath**

Specifies the installation path under which the files for the current component are stored. You can enter an absolute path name or an environment variable. The first character of the environment variable must always be the dollar sign ($). The second character must neither be a dollar sign ($) nor a number. The path name must match the operating system conventions.

**ctrlname**

Specifies the name of a dialog element, for example a button or a label. The name can consist of 1 – 32 characters.

**deactparamset**

Is a comma-separated list of parameter=value assignments that are activated when the check box is cleared.

**dependencylist**

(Used for check box trees only) Is a comma-separated list of check box *ctrlnames*. No spaces are allowed in this list.

**dialogvalidationscript**

Specifies the name of a shell script (provided by the user) that can be used to validate the input of the current dialog. All values entered are passed as a shell parameter to the dialog validation script. If each entered value and the combination of all values are correct, the script must return 0. If the entered values are wrong, the script must return the number of a matching error text in the resource section in the prototype file. This error text number must be in the range of 1 – 255.

**dlgref**

Specifies the name of the next dialog in the product installation process. The name can consist of 1 – 32 characters maximum. In a special manner it is possible to specify a shell script as the dialog reference. This shell script must print the name of the dialog to standard output.

**externalpostinstallprocedure**

Specifies the name of a post-install procedure that is not part of the current PIF product file. Be careful using external procedures, because a particular procedure may be missing on a target computer.

**externalpostuninstallprocedure**

Specifies the name of post-uninstall procedure that is not part of the current PIF product file. Be careful using external procedures, because a particular procedure may be missing on a target computer.

**externalpreinstallprocedure**

Specifies the name of a pre-install procedure that is not part of the current PIF product file. Be careful using external procedures, because a particular procedure may be missing on a target computer.

**externalpreuninstallprocedure**

Specifies the name of a pre-uninstall procedure that is not part of the current PIF product file. Be careful using external procedures, because a particular procedure may be missing on a target computer.

**fieldwithfocus**

Specifies the name of a dialog element (field) that gains the focus when a new installation dialog is started.

**filename**

Specifies the name of a file to display on the installation dialog. The name is a character string that consists of a maximum of 255 characters.

**filetype**

Specifies the file type. The value 1 references a standard file (file type 1). The value 2 references a configuration file (file type 2), which is not changed during a PIF product update. The value 3 references a temporary file (file type 3).

**firstinstalldialog**

Specifies the name of the starting dialog for an initial PIF product installation. The name can consist of 1 – 32 characters.

**firstupdatedialog**

Specifies the name of the starting dialog for a PIF product update installation. The name can consist of 1 – 32 characters.

**flag**

Is a boolean indicator. The values of flag can be False, represented by 0, or True, represented by 1.

**fontbold**

Is a flag that indicates if the font appears bolded or not in the installation dialog. Flag value = 1 specifies bold, flag value = 0 specifies normal font presentation.

**fontcolor**

Specifies the color (RGB) of the font. The format is R,G,B; for example, 255,0,0 specifies red color.

**fontitalic**

Is a flag that indicates if the font appears italicized or not in the installation dialog. Flg value = 1 specifies italic, flag value = 0 specifies standard font presentation.

**fontlogical**

Specifies the Java-specific name of the logical font.

**fontname**

Specifies the specific name of the font in the PIF product.

**fontsize**

Is a boolean indicator. The values of flag can be False, represented by 0, or True, represented by 1.

**GIFfile**

Specifies the file name of a graphic in the GIF graphics format. This graphic is displayed in the graphics area of the installation dialog. The GIF file name can consist of 1 – 255 characters.

Since the size of the graphic influences the overall size of the dialog, we recommend that the graphic does not exceed 400 x 200 pixels (approximately) to support screen resolutions of 800 x 600 pixels.

**groupID**

Identifies the group, for example, sys. The value of groupID can consist of a maximum of 32 bytes.

**height**

Specifies the height of the dialog window in number of pixels.

**Note:** The overall height of the dialog is controlled by the height of the graphic specified in the #graphic entry in the dialog section.

**icon**

Specifies the name of a file that contains an icon that appears on the button. The icon must be provided in the GIF graphics format. The GIF file containing the icon must be added to the pre-install component. The name of the GIF file can consist of 1 – 255 characters.

**installtime**

Specifies the time when the external component is installed. Value 0 specifies that the external component is installed prior to PIF product installation. If the value is 1, the external component is installed after PIF product installation.

**instparam**

Specifies the name of the parameter that specifies whether the dependent product should be installed with the PIF product or only on demand (optional). If the value of the parameter *instparam* is 0, the dependent product is not installed; if the value is 1, the dependent product is installed with the PIF product. By default, the dependent product will be installed. (See the property paramname for information about the syntax of a parameter name.)

**interpreter**

Specifies the name of an interpreter other than the standard shell.

**linksource**

Specifies the name of a link source file. The name must match with the value of the *target* parameter of a preceding #file: entry.

**listparam**

Specifies the name of a parameter that contains the list of values for selection in the list box or combo box.

**localeID**

Identifies the language to be used as a three-character text string, for example, ENU (English (U.S.)). Enter one of the supported language identifiers (see page 151).

**migrationscript**

Specifies the name of a script (provided by the user) to migrate a non-PIF product to a PIF product. The script is started only when a PIF product is being new installed on a computer. This script is not executed when updating or re-installing a PIF product. The script should be located in the pre-install component. If a non-PIF installation is detected on the computer, we recommend that you set the following PIF parameters: PIF_INSTALL_JOB=UPDATE and *PIF_INSTALL_PATH=existing installation path*.

**nextdialog**

Specifies the name of the next dialog in the product installation process. The name of the dialog can consist of 1 – 32 characters.

**operator**

Indicates the operator defining the relationship to the specified product version. Operator can be one of the following characters: =, >, >=

For example, = 2.1.0.3 means exactly product Version 2.1.0.3; >= 2.1.0.3 means Version 2.1.0.3 and all newer versions of the product, such as 2.1.1.0 or 2.2.0.0.

**ownerID**

Identifies the user, for example, root. The ownerID can consist of a maximum of 32 bytes.

**paramcomment**

Is a text describing the parameter.

**paramname**

Specifies the name of an environment variable. The name starts with a dollar sign ($), and consists of a maximum of 64 characters, including the dollar sign. The following characters are not allowed within a parameter name:

```
.  ;  :  \t  \n  \\  \"  /  ,  $  (  )  -  {  }  white space
```

**paramset**

Is a comma-separated list of assignments in the format parameter=value, for example:

```
$PARAM1=Yes,$PARAM2=x,$INSTALLDIR=/dev/null
```

The commas in the list must not be separated by spaces! When the according button is clicked, the values are assigned to the parameters.

**paramtype**

Specifies the type of the parameter. The following values are accepted: 0 (= published), 1 (= private), and 2 (= global).

**paramvalue**

Specifies the default value for a parameter. Paramvalue is a character string and consists of a maximum of 255 characters. The parameter value can be a combination of static values and previously defined parameters. Within a combined parameter, the end of a static value is marked by one of the following characters:

```
.  ;  :  \t  \n  \\  \"  /  ,  $  (  )  -  {  }  white space
```

**platform**

Specifies the operating environment of the target computer. Enter one of the valid platform identifiers.

**postinstallprocedure**

Specifies the name of a procedure to execute after installation of the PIF product. This procedure must be provided by the user as a script or executable. Enter the path name (0 – 255 characters) relative to the path specified in the #ppath: entry. The file must be stored in the product installation directory, or a subordinate directory. If there is no such procedure leave this entry blank or omit it.

**postuninstallprocedure**

Specifies the name of a procedure to be executed after the removal of the PIF product. The user must provide this procedure as a script or executable. Enter the path name (0 – 255 characters) relative to the path specified in the #ppath: entry. If there is no such procedure leave this entry blank or omit it.

**preinstallcomponent**

Specifies the name of a component, which contains all text, script, and resource files used prior to PIF product installation. Typically, pre-install components contain dialog resource files like GIF files, license agreement documents, and pre-install scripts. The pre-install component is not installed within the PIF product installation.

**preinstallprocedure**

Specifies the name of a procedure to execute prior to installation of the PIF product. The user must provide this procedure as a script or executable. Enter the path name (0 – 255 characters) relative to the path specified in the #ppath: entry. The file must be stored in the product installation directory or a subordinate directory. If there is no such procedure leave this entry blank or omit it.

**prestartprocedure**

Specifies the name of a procedure to execute before the product installation or any installation dialog starts, for example, to define pre-settings needed by the installation dialogs. The user must provide this procedure as a script or executable, as part of the pre-install component. Enter the path name (0 – 255 characters) relative to the path specified in the #ppath: entry. If there is no such procedure leave this entry blank or omit it.

**preuninstallprocedure**

Specifies the name of a procedure to execute prior to the removal of the PIF product. This procedure must be provided by the user as a script or executable. Enter the path name (0 – 255 characters) relative to the path specified in the #ppath: entry. If there is no such procedure leave this entry blank or omit it.

**prodfamilyname**

Specifies the name of a family of PIF products, for example, CCS. The name can consist of a maximum of 128 characters, but must not contain any spaces or slashes. The product family name is not case-sensitive.

**prodlist**

Is a comma-separated list of product names. The list must not contain spaces around the commas.

**prodname**

Specifies the name of a PIF product. The name can consist of a maximum of 128 characters.

**prodpath**

Specifies the installation path for the PIF product on the target computer. Enter the absolute path name on the target computer, where you want to install the product. Enter the path name starting from the root directory (/). Instead of a path name, you can enter the name of an environment variable that indicates the path name on the target computer. The first character of the environment variable name must always be the dollar sign ($). The second character must be neither a dollar sign ($) nor a number. The path name must match the operating system conventions.

**progbartype**

Is an environment variable which describes the installation progress of the current product or component. The appropriate values are as follows:

```
$PIF_PROGRESSBAR_PRODUCT
$PIF_PROGRESSBAR_COMPONENT
$PIF_PROGRESSBAR_COUNT
```

**proglabeltype**

Is an environment variable which describes the currently running install action, the file being installed, or the component name. The appropriate values are as follows:

```
$PIF_PROGRESSLABEL_ACTION
$PIF_PROGRESSLABEL_FILE
$PIF_PROGRESSLABEL_COMPONENT
$PIF_PROGRESSLABEL_ELAPSEDTIME
$PIF_PROGRESSLABEL_PHASE
```

**resultparam**

Specifies the name of a parameter that contains the value selected from the list box or combo box.

**resultparamlist**

Is a list of parameter=value assignments that are activated when the radio button is selected.

**row**

Indicates the row number of the dialog element within the virtual grid of 4 columns and 10 rows. The value range is 1 – 10.

**script**

Specifies the name of a script file.

**scrollmode**

(Used for VT100 only) Specifies scroll mode, using the values: 0 = Do not show a frame (box) and 3 = Show a frame (box).

**source**

Specifies the name of the source file you want to package into a PIF product. The source file name (0 – 255 characters) can be relative. In this case, the source file path can be lead by an environment variable. This variable must be set prior to creation of the PIF product.

**symboliclink**

Specifies the file name (0 – 255 characters) of a symbolic link. A symbolic link is a file that contains a path name. When the shell encounters a file name that represents a symbolic link, it replaces its name with the specified path name. Thus, what you access is not the symbolic link, but the file to which the path name points. You can set up symbolic links to any files or directories, even to those on different file systems on the computer.

You can also create symbolic links that are relative to the target file (if possible). The target link is not referenced with a fully qualified path, but with a path that is determined relatively to the symbolic link. The advantage is that the installed folders containing symbolic links can be moved without breaking the link.

Enter the symbolic link in accordance with one of the following checks:

- If the specified symbolic link is already an absolute path name, then the specified target path will be used to create the symbolic link.
- If the product path is absolute and the component path is relative (or empty), then the symbolic link will be created relative to the concatenation of the absolute product path and the relative component path. The same applies if the absolute product path is represented by an environment variable.
- If the specified component path is an absolute path name, then the symbolic link will be created relative to the absolute component path. The same applies if the absolute component path is represented by an environment variable.

**target**

Specifies the target path name (0 – 255 characters) in accordance with one of the following checks:

- If the specified file path is already an absolute path name, then the specified path will be used for installation.
- If the product path is absolute and the component path is relative (or empty), then the file will be installed relative to the concatenation of the absolute product path and the relative component path. The same applies if the absolute product path is represented by an environment variable.
- If the specified component path is an absolute path name, then the file will be installed relative to the absolute component path. The same applies if the absolute component path is represented by an environment variable.

**textarea**

Specifies the name of a text area dialog element that contains additional information of the items within the scroll area. The text area must be defined in the current dialog.

**textID**

Indicates the reference number of a text string defined in the Resource section in the prototype file. The value range of the reference number is 0 to 9999.

**textstring**

Specifies the text to be used, for example, in an installation dialog. You can enter 1 – 1024 characters, but the text must not contain any control characters like carriage return (CR) and line feed (LF).

**timeout**

Indicates a time-out value used for long executing scripts. Enter the value in seconds. Default value is 600 (ten minutes).

**validationscript**

Specifies the name of a shell script (provided by the user) that validates the input from a text field. The value entered in the text field is passed as first argument ($1) to the validation script. If the entered value is valid, the script must return 0. If the entered value is wrong, the script must return the number of a matching error text in the Resource section in the prototype file. This error text number must be in the range of 1 – 255.

The property *validationscript* is a character string that consists of a maximum of 255 characters.

**vendorname**

Specifies the name of the product vendor.

**version**

Indicates the version of the PIF product, or version of a component of the PIF product. The version has the format M.m.b.r, where M (major), m (minor), b (build), and r (revision) are numeric values in the range of 0 – 99,999. Each of the values for M, m, b, and r can consist of a maximum of five digits, for example, 10 or 00010 are allowed values.

**width**

Specifies the width of the installation dialog window in number of pixels.

# Packager and Installer Commands

You can use the following commands at the command line to create and register PIF products:

**pifproto**

Create a prototype file for a PIF product.

**pifmk**

Generate the PIF product file.

**pifself**

Generate a self-installing PIF product.

**pifextract**

Extract a PIF product file.

**pifdelta**

Create a delta PIF product.

**piflib**

Generate a PIF library file.

**pifpatch**

Generate a PIF patch product automatically.

**sd_registerproduct**

Register a PIF product in the Software Package Library.

You can use the following command to install PIF products and manage installed products on the target computer:

**lsm**

Install, remove, save, query, or list products.

## pifproto - Create a Prototype File for a PIF Product

The pifproto command creates a prototype file that describes the PIF product. The prototype file is used by the pifmk command to generate a PIF product. The prototype file is created in the current working directory. The name of the prototype file is:

*prodname*.*platform*.@prm

You can specify that the PIF product should be created immediately after the run of the pifproto command, or later. For individual configurations, you can modify the prototype file manually.

**Note:** A temporary subdirectory .pif* is created during the execution of the pifproto command. If the command is successfully terminated, this subdirectory will be automatically deleted. If the command fails, the subdirectory remains in the directory tree.

This command has the following format:

```
pifproto -d proddir  -a platform  -n prodname  -v version  -b basedir
    [-c comment] [-s vendorname] [-B buildID]
    [-I preinstallprocedure]
    [-i postinstallprocedure]
    [-r preuninstallprocedure]
    [-R postuninstallprocedure]
    [-T template]
    [-p  |  -S]
    [-o]
```

**-d *proddir***

Specifies the name of the destination directory, where the item is stored. The current working directory must be outside of the specified *proddir*.

**-a *platform***

Specifies the name of the operating system platform of the target computer where the PIF product shall be installed.

**-n *prodname***

Specifies the name of the PIF product to package. Product names that differ by lowercase and uppercase letters cause different PIF products.

**-v *version***

Specifies the version of the PIF product to create.

**-b *basedir***

Specifies the directory where the PIF product should be relatively installed, for example, /usr/bin, or a variable name, for example, $SDPATH.

**-c *comment***

Specifies a product-specific comment.

**-s *vendorname***

Identifies the product supplier or product vendor.

**-B *buildID***

Specifies the build, where *buildID* can consist of a maximum of 1024 alphanumeric characters, but must not contain any control characters like carriage return (CR) and line feed (LF).

**-I *preinstallprocedure***

Specifies the shell script or executable file, which is called before the PIF product has been installed.

**-i *postinstallprocedure***

Specifies the shell script or executable file, which is called after the PIF product has been installed.

**-r *preuninstallprocedure***

Specifies the shell script or executable file, which is called before the PIF product is removed.

**-R *postuninstallprocedure***

Specifies the shell script or executable file, which is called after the PIF product is removed.

**-T *template***

Specifies a template used as a framework for the creation of an individual product setup. A template is formatted as a prototype file.

**-p**

Specifies that the PIF product is automatically created after the run of the pifproto command by internally executing the pifmk command. Symbols of binary files are discarded automatically.

**-S**

Specifies that a self-installing PIF product is automatically created after the run of the pifproto command by internally executing the pifmk and pifself commands.

**-o**

Specifies overwrite. If an item with the same name already exists in the destination directory, it is overwritten.

### Exit status:

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

### Example: Create a prototype file

The following command creates a prototype file named test-pif.AIX.@prm, which contains the files located under the /home/bin directory.

```
pifproto -a AIX -d /home/bin -b /usr/bin -v 1.0.0.0 -n test-pif
```

## pifmk - Generate a PIF Product

The pifmk command generates a PIF product using the prototype file created by the pifproto command. Sufficient disk space is required for the PIF product generation; the minimum size is the size of the product to be created. By default, the pifmk command creates the PIF product files in a compressed format in the current working directory of pifmk. The name of the PIF product is *prodname*.*platform*.@pif. You can specify another destination directory where the PIF product should be stored.

**Note:** Compressed PIF products are not backward compatible, that is, they cannot be installed with the Installer of previous PIF SDK for Linux and UNIX versions (see table Installer Compatibility Considerations (see page 69)).

This command has the following format:

```
Pifmk -v
```

**-v**

Prints the version of pifmk.

Or

```
pifmk  -f  PIFprototypefile [-d proddir] [-a platform]
        [-v version] [-B buildID] [-o] [-s] [-t tracefile]
        [-c] [-e] [-A platform] [-c] [-V (40|42|43)]
```

**-f *PIFprototypefile***

Specifies the name of the prototype file created by the pifproto command.

**-d *proddir***

Specifies the name of the destination directory, where the item is stored. The current working directory must be outside of the specified *proddir*.

**-a *platform***

Specifies the name of the operating system platform of the target computer where the PIF product shall be installed.

**-v *version***

Specifies the version of the PIF product to create.

**-B *buildID***

Specifies the build, where *buildID* can consist of a maximum of 1024 alphanumeric characters, but must not contain any control characters like carriage return (CR) and line feed (LF).

**-o**

Specifies overwrite. If an item with the same name already exists in the destination directory, it is overwritten.

**-s**

Specifies strip. Symbols are discarded from the binary files.

**-t *tracefile***

Traces the command execution in the specified trace file.

**-c *check only***

Checks the package integrity without checking the file existence.

**-e *write an empty package***

Writes the package without including any file.

**-A *<platform>***

Specifies a new platform. This overwrites the default platform from the PIF prototype file.

**-C *separate***

Separates PIF and self installer.

**-V *40|42|43***

Creates a package that is compatible to the PIF-SDK versions 4.0, 4.2, or 4.3.

### Exit status:

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

### Example: Create a PIF product from a prototype file

The following command creates a PIF product using the prototype file test-pif.AIX.@prm. A build ID "Final build" is specified. The PIF product is stored as test-pif.AIX.@pif.

```
pifmk -f test-pif.AIX.@prm -B "Final build"
```

## pifself - Generate a Self-Installing PIF Product

The pifself command generates a self-installing PIF product using the PIF product file created by the pifmk command. The PIF product can be installed without using any installation program on the target computer. A standard shell invokes the installation procedure. Sufficient free disk space is required for the product generation; minimum size is the size of the product to be packaged.

The -a option lets you create a self-installing PIF product, even if the product was created for “Any” platform.

The -j option lets you integrate a Java Runtime Environment (JRE) in a self-installing PIF product. This JRE will then be used by the Installer on the target system to run the installation dialogs.

When you use the -s option, you can split the self-installing PIF product into three separate units, the start script, the actual PIF product, and a self-installing shell script including the Installer. To have these separated units is of value if you are working in a software delivery environment and do not need the self-installing function. In this case, you can register only the separated actual PIF product in the Software Package Library. Whenever you want to use the self-installing function, you must ensure that the three units mentioned previously are located in the same folder, and run the self-installing shell script.

This command has the following format:

```
pifself -f PIFfile -d selfinstallingfile
        [-a platform] [-j {JREpath|"default"}] [-s] [-o]
```

**-f *PIFfile***

Specifies the name of the PIF product file.

**-d *selfinstallingfile***

Specifies the file name of the self-installing PIF product.

**-a *platform***

Specifies the name of the operating system platform of the target computer where the PIF product shall be installed.

**-j {*JREpath* | "default"}**

Imports the Java Runtime Environment (JRE) from the specified path or uses the default JRE. The default JRE is a stripped and downsized JRE version for Linux platforms. The value of *JREpath* should be a valid JRE directory like /usr/lib/jvm/jre-1.4.2sun.

**-s**

Specifies separate. The PIF product file specified through the -f option is separated from the self-installing file.

**-o**

Specifies overwrite. If an item with the same name already exists in the destination directory, it is overwritten.

**Exit status:**

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

### Example: Create a self-installing PIF product

The following command creates a self-installing PIF product using the file test-pif.AIX.@pif. The self-installing PIF product is stored as test-pif.AIX.sh.

```
pifself -f test-pif.AIX.@pif -d test-pif.AIX.sh
```

## pifextract - Extract a PIF Product

To modify an already built PIF product, for example, change its product definition, and add, modify, or remove files, you can use the pifextract command. After modification, the product can be rebuilt using the pifmk command.

The pifextract command extracts an already built PIF product, or parts of this PIF product to a specified destination directory. The destination directory either must not exist or must be empty. Sufficient free disk space is required to extract the product. The prototype file is also added to the destination directory.

This command has the following format:

```
pifextract  -f  PIFfile -d proddir [-l localeID -L newlocaleID] [-p]
                [-t tracefile]
```

**-f *PIFfile***

Specifies the name of the PIF product file.

**-d *proddir***

Specifies the name of the destination directory where the PIF product is extracted. The directory must not exist or must not contain any files.

**-l *localeID***

Specifies a three-character language identifier (see page 151). According to the specified language identifier, only the language-specific parts of the PIF product are extracted.

**-L *newlocaleID***

Specifies a three-character language identifier (see page 151), which determines the desired new language.

**-p**

Extracts all dialog-relevant sections and the pre-install components into a temporary folder, and the PIF product is built. The new PIF product is stored in the current working directory. The name of the new product is *prodname*.*platform*.@prf.

**-t *tracefile***

Traces the command execution in the specified trace file.

### Exit status:

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

### Example: Extract a PIF product

The following command extracts the PIF product test-pif.AIX.@pif into the directory /tmp/extractdir:

```
pifextract -f test-pif.AIX.@pif -d /tmp/extractdir
```

## pifdelta - Create a Delta Product

The pifdelta command compares the contents of two versions of the same PIF product, *priorversionPIF* and *resultingversionPIF*, and writes the differences into the output file *deltaPIF*.

The *deltaPIF* is a small installable unit in the PIF packaging format. Instead of installing the complete PIF product *resultingversionPIF*, you can install the small *deltaPIF* unit on all target computers where the PIF product *priorversionPIF* is already installed. For example, *deltaPIF* contains uninstall instructions for files in the *priorversionPIF* that are not used in *resultingversionPIF*. The *deltaPIF* also contains all changed and new files from *resultingversionPIF*.

This command has the following format:

```
pifdelta  -p  priorversionPIF -v resultingversionPIF
        -d deltaPIF [-o] [-t tracefile]
```

**-p *priorversionPIF***

Specifies the absolute path name of a file containing a PIF product. This product is to be replaced by the new version *resultingversionPIF* of the same PIF product.

**-v *resultingversionPIF***

Specifies the absolute path name of a file containing a version of the PIF product, which is different from *priorversionPIF* and shall replace *priorversionPIF*.

**-d *deltaPIF***

Specifies the absolute path name of the output file. This file is an installable unit in the PIF packaging format, and contains only the files that differ between *priorversionPIF* and *resultingversionPIF*.

We recommend using the naming convention for the delta product file name as used by the Packager GUI:
*prodname*.*platform*.*priorversion*-*resultingversion*.@pif

In this file name, *prodname* is the common product name and *priorversion* and *resultingversion* are the version numbers of *priorversionPIF* and *resultingversionPIF*, for example:

```
MERCHANT.Any.2.1.0.0-3.0.0.0.@pif
```

**-o**

Specifies overwrite. If an item with the same name already exists in the destination directory, it is overwritten.

**-t *tracefile***

Traces the command execution in the specified trace file.

### Exit status:

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

### Example: Create a delta product from two product versions

The following command compares the versions of the PIF product MERCHANT located in the subdirectories called 2100 and 3000 of the product installation directory and creates the PIF delta product containing only the differences between the product versions 2.1.0.0 and 3.0.0.0 in the subdirectory called deltas.

```
pifdelta -p $PATHmerchant\2100\MERCHANT.Any.@pif
    -v $PATHmerchant\3000\MERCHANT.Any.@pif
    -d $PATHmerchant\deltas\MERCHANT.Any.2.1.0.0-3.0.0.0.@pif
```

## piflib - Generate a PIF Library File

The piflib command generates a PIF library file with the fixed name library.dct in the current working directory. This library file references all PIF products stored under the current working directory, and is used by the Installer to detect PIF products that contain common shared components. The piflib command follows symbolic links when generating the library file (dictionary).

This command has the following format:

```
piflib [-o] [ -d directory ] [-P]
```

**-o**

Specifies overwrite. If an item with the same name already exists in the destination directory, it is overwritten.

**-d *directory***

Optionally specifies the name of a folder where the library.dct file is stored.

**-P *scan PKG packages***

Scans for PKG packages on non-PKG platforms (like Linux). By default, PKG packages are scanned only on PKG platforms (like Solaris, UnixWare, or IRIX).

**Exit status:**

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

## pifpatch - Generate a Patch Product Automatically

The pifpatch command generates a patch automatically, based on the PIF delta product creation mechanism. The command compares two sets of packages and queries the dependency structure of the main product.

This command has the following format:

```
Pifpatch -v
```

**-v**

Prints the version of pifmpatch.

Or

```
pifpatch -p originalpackage -f package_with_patch
    -d destinationfolder -I prestart_script
    -i postinstall_script -n anchorpatchPIF -v version
    [-c comment] [-o] [-t tracefile] [-x]
```

**-p *originalpackage***

Specifies the absolute path name of the original product package.

**-f *package_with_patch***

Specifies the absolute path name of the package version containing the patch.

**-d *destinationdirectory***

Specifies the name of an empty directory where the package containing the patch is written to.

**-I *prestartprocedure***

Specifies a procedure or script to execute before the patch is installed.

**-i *postinstallprocedure***

Specifies a procedure or script to execute after installation of the patch.

**-n *anchorpatchPIF***

Specifies the name of the patch product.

**-v *version***

Specifies the version number of the patch.

**-c *comment***

Specifies any comment.

**-o *overwrite***

Overwrites the patch.

**-t *tracefile***

Traces the command execution in the specified trace file.

**-x *extract only***

Extracts all necessary files and writes the PIF prototype file, but does not build the package. You can modify the patch package before building it.

**Exit status:**

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

## sd_registerproduct - Register a PIF Product in the Software Package Library

The sd_registerproduct command registers a product in one of the DEB, PIF, PKG, OSX, HPSD, BFF, or RPM packaging formats in the Software Package Library on a manager system. If a file name is specified (using the -f option), the sd_registerproduct command checks for the packaging format of this file. If a directory name is entered (-d option) the packaging format is assumed to be a PKG directory structure. With the –r option you can specify a list of response files that are used as input files for an unattended installation of a DEB, PIF, PKG, OSX, HPSD, BFF, or RPM product. Response files are registered in the Software Package Library together with the corresponding product.

This command has the following format:

```
sd_registerproduct {-f filename  |  -d directoryname}
        -o remoteserver -u remoteuser
        [-r responsefilelist] [-b] [-n] [-S] [-N package name] [-V version] [-A
platform]

sd_registerproduct -f filename -R
```

**-f *filename***

Specifies the file name of a PIF, PKG, or RPM product.

**-d *directoryname***

Specifies the location where the PKG product (directory format) is stored.

**-o *remoteserver***

Specifies the name of the manager system, where the Software Package Library is located.

**-u *remoteuser***

Specifies the user account, under which the product is registered in the Software Package Library. The user account *remoteuser* has the following format:

```
[domain\]user::password
```

**-r *responsefilelist***

Specifies a comma-separated list of response file names. In the list, no spaces must be around the commas.

**-b**

Registers a product that can be installed only before system shutdown.

**-R**

Creates a reginfo structure in the current directory. This reginfo structure lets you register a product in the Software Package Library using drag-and-drop.

**-n**

Suppresses dependent package registration.

**-S**

Enables software delivery.

**-N *package name***

Replaces the package name of the package that is registered.

**-V *version***

Replaces the version of the package that is registered.

**-A *platform***

Replaces the architecture of the package that is registered.

**Exit status:**

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

**Example: Register a PIF product on a manager system**

The following command registers the PIF product test-pif.AIX.@pif in the Software Package Library on the manager system named daisy, for the specified user named donald, and password mypasswd:

```
sd_registerproduct -f /tmp/test-pif.AIX.@pif -o daisy -u donald::mypassword
```

## lsm—Manage Software Installations on Linux and UNIX Systems

The lsm command lets you manage products of various different packaging formats on target computers. These packaging formats include:

- PIF (CA Technologies Product Interchange Format)
- PKG (UNIX System V Release 4 standard packaging format)
- RPM (Red Hat package manager format)
- IBM AIX packaging format
- Hewlett Packard (HP-UX) packaging format
- Apple Macintosh Mac OS X packaging format

The lsm command provides methods to install, remove, list, backup, check, query installed products, query PIF product files, and update installed products. Therefore, root permission is required on the target computer.

The lsm command has the following formats:

- Install

  `lsm -i` *`prodfile`* `[-r` *`responsefile`*`] [-s] [-F] [-V] [-R] [-p` *`log file`*`]`

- Install a new instance

  `lsm -i` *`prodfile`* `[-I` *`instance name`*`] [-r` *`responsefile`*`] [-s] [-F] [-V] [-p` *`log file`*`]`

- Remove

  `lsm -e` *`prodname`* `[-s] [-R]`

- List

  ```
  lsm -l [-O {pif | pkg | rpm | bff | osx | hpsd | patches}] [-g prodfamilyname]
  [-f filename]
  ```

- Backup

  ```
  lsm -A prodname -d prodfile [-o]
  ```

- Check

  ```
  lsm -c prodname
  ```

- Query

  ```
  lsm -q prodname [-l] [-ODEPLIST]
  ```

- Query Package

  ```
  lsm -Q prodfile [-l] [-ODEPLIST]
  ```

- Ask

  ```
  lsm -a prodfile -r responsefile
  ```

- Print version

  ```
  lsm -v
  ```

- Update product family

  ```
  lsm -u prodfamilyname -d prodpath
  ```

- Print message text

  ```
  lsm -M textID [-f prototypefile]
  ```

- Print error text

  ```
  Lsm -E textID [param1] [param2] [param3] [param4]
  ```

- Export internal scripts

  ```
  lsm -x prodfile exportspec [-d directory] [-o] [-s]
  ```

- Configure

  ```
  Lsm -C prodname
  ```

- Print system info

  ```
  Lsm -L
  ```

- All

  ```
  [ -t tracefile]
  ```

**-i *prodfile* [-r *responsefile*] [–s] [–F] [–V] [–R]**

Installs a PIF, PKG, or RPM packaged product. The full path name of the product must be specified as prodfile. A response file can be added to customize the unattended installation.

The –s option lets the installation run in unattended (silent) mode.

The -F option performs a forced installation; that is, if the backup of an existing product fails, the installation continues.

The -V option instructs the Installer to use the text mode even if Java is installed.

The -R option retains configuration files.

**-e *prodname* [-s] [-R]**

Removes the specified installed product.

The -s option lets the removal run in unattended (silent) mode.

The -R option specifies to leave all configuration files unchanged on the system.

**-l [-O {pif | rpm | pkg | patches}] [-g *prodfamilyname*] [-f *filename*]**

Lists all installed products.

**Note:** If the list command is used during the installation process, further information is printed to the screen.

The -O option lists only products of the specified packaging format or only patch products.

The -g option lists all products that are assigned to the specified product family.

The -f option lists the product that installed the specified file.

**-A *prodname* -d *prodfile* [-o]**

Creates the backup file *prodfile* of the installed PIF or PKG product prodname.

The -o option overwrites an existing *prodfile*.

**-c *prodname***

Checks the specified installed product and all depending products for consistency. That is, the files of the product are checked for existence and for proper access, user and owner rights. Checks also if required products are installed.

**-C *prodname***

Configures the specified product. Executes the configuration wizard (first dialog is specified by the keyword #dlgconfig keyword.)

**-q *prodname* [-l] [-ODEPLIST]**

Queries the installed product prodname and shows the product properties. The -ODEPLIST option prints the dependency hierarchy (INSTALL_ORDER) for the specified product to the console. The -l option provides a long list including all installed product files.

**-Q *prodfile* [-l] [-ODEPLIST]**

Queries the PIF product file prodfile, and shows the file properties. The -ODEPLIST option prints the dependency hierarchy (INSTALL_ORDER) for the specified product file to the console. The -l option provides a long list including all installed product files.

**-a *prodfile* -r *responsefile***

Runs the installation dialogs and creates a response file using the values entered. The PIF product is not installed.

**-v**

Prints the version of the Installer used.

**-u *prodfamilyname* -d *prodpath***

Directs the Installer to scan the system for all matching product packages with the same product family name, and to update those products to the latest version level. The -d option specifies the location where the latest product version is stored. The latest product version can be provided as an update CD or in a local folder.
All product packages are updated to the highest version. Product packages on the update CD that have lower versions than the target packages are skipped during update.

**-M *textID1*[-*textID2*] -f *prototypefile***

Refers to a single text (specified by *textID1*) or a range of texts (specified by *textID1-textID2*) in the resource section of the specified PIF product prototype file. Each script of a PIF product can use this command option in order to print localized texts.

**-x *prodfile exportspec* [-d *directory*] [-o] [-s]**

Runs through the setup process and extracts a script and the complete environment to execute and test the script. Interviews and the response file are also extracted to let the user change parameters.

*exportspec* indicates which script environment to export and can have one of the following values:

| | |
|---|---|
| --exportprestart | Exports the prestart script environment |
| --exportpreinit | Exports the preinit script environment |
| --exportpostinit | Exports the postinstall script environment |
| --exportrmpreinit | Exports the preremove script environment |
| --exportallscripts | Exports all script environments listed above within one call |

The -d *directory* option specifies the target directory. You must enter -d pwd or -d . (dot) for the current directory.

The *-o* option specifies that existing files in the target directory are overwritten.

The *-s* option specifies silent mode, that is, there is no user interaction.

### Exit status:

Displays the status of the command execution. The value zero (0) means OK, any non-zero value indicates Error.

### Example: Install PIF product on local system

The following command installs the PIF product test-product.Any.@pif on the local system.

```
lsm -i test-product.Any.@pif
```

### Example: Remove PIF product from local system

The following command removes (uninstalls) the PIF product test-product.Any.@pif from the local system in unattended (silent) mode.

```
lsm -e test-product -s
```

### Example: Create backup file of PIF product

The following command creates a backup file of the installed PIF product test-product. The backup file has the name test-product.bckp, and is located in the /tmp directory.

```
lsm -A test-product -d /tmp/test-product.bckp
```

### Example: Update product family

The following command scans the system for all packages with the product family name CCS, and updates them with product packages that are stored in the local folder /tmp/2.0.0.0.

```
lsm -u ccs -d /tmp/2.0.0.0/
```

### Example: Using lsm -M in a script

In the following example a script of a PIF product uses the lsm -M option to print the localized text with the number 100 from the resource section.

```
@RESOURCE:
#text: 100 , Dieses ist ein lokalisierter Text ;
@ENDRESOURCE:
```

The script looks like this:

```
#!/bin/sh
echo `lsm -M 100 2>/dev/null`
exit 0
```

### Example: Extract the prestart script and its environment for testing purposes

The following lsm command version runs the setup of the PIF product test-product.Any.@pif and extracts the prestart script and creates a "wrapper" script which contains the complete environment information to execute the prestart script for testing purposes. The extracted information also includes dialogs for user interaction and the response file, so that parameters can be updated during the test phase.

The name of the "wrapper" script is constructed by the Installer (lsm command) as start_prestart.sh.

The target directory for the command output is the current working directory.

```
lsm -x test-product.Any.@pif --exportprestart -d pwd
```

# Install a Self-Installing PIF Product

A self-installing PIF product is executed as a shell script. The command options specify how to install the PIF product on the target computer or that only a response file is created.

This command has the following format:

```
sh selfinstallingfile [-t tracefile] [-p logfile] [-r responsefile]
   [-s] [-F] [-x] [-a newresponsefile] [-v] [-I instance name] [-V] [-o] [-M textID]
```

***selfinstallingfile***

Name of the self-installing PIF product, as defined, for example, through the pifself command.

**-t *tracefile***

Traces the installation in the specified trace file.

**-p *logfile***

Logs the installation in the specified log file.

**-r *responsefile***

Installs the product using the specified response file.

**-s**

Installs the product in unattended (silent) mode using default parameter values.

**-F**

Forces the product installation. Backup errors are ignored.

**-x**

Only extracts the product to the /tmp directory. The file name used for the extracted product is internally defined and echoed to the command line.

**-a *newresponsefile***

Creates only a response file with the specified name. This response file can be used for a later installation of the PIF product.

**-v**

Lists the version of the installer included.

**-I *instance name***

Specifies the name of a new product instance (for multi instance installation).

**-V**

Enforces VT100 dialogs.

**-o**

Overwrites existing response during response file generation (-a).

**-M *textID1[-textID2] -f prototypefile***

Refers to a single text (specified by textID1) or a range of texts (specified by textID1-textID2) in the resource section of the specified PIF product prototype file. Each script of a PIF product can use this command option in order to print localized texts.

### Example: Execute a self-installing PIF product and record trace information

The following command executes the self-installing PIF product test-pif.AIX.sh and installs the corresponding PIF product in unattended (silent) mode. Trace information is written to the specified trace file test-pif.AIX_tracef in the mytemp folder:

```
sh test-pif.AIX.sh -s -t /mytemp/test-pif.AIX_tracef
```

# Supported Linux and UNIX Operating Environments

In the PIF product prototype file, you specify a platform identifier that states on which hardware and operating environments the PIF product can be installed. The following table provides an overview of the supported operating environments and their platform identifiers. The platform identifier Any states that the PIF product can be installed in any of the supported operating environments, because the PIF product has no dependencies on the operating system.

| Platform Identifier | Operating Environment |
|---|---|
| AIX | IBM AIX computers |
| Any | Any of the supported operating environment |
| HPUX | Hewlett Packard computers |
| Irix | Silicon Graphics workstations |
| Linux | Linux (Intel) systems |
| LinuxS390 | IBM Linux mainframes |
| MacOSX | Apple Macintosh OS systems (PPC) |
| MacOSXIntel | Apple Macintosh OS systems (Intel) |
| MacOSXUB | Apple Macintosh OS systems (Universal Binary) |
| NCR | NCR UNIX systems |
| SolarisSparc | Sun Solaris (Sparc) systems |
| SolarisIntel | Sun Solaris (Intel) systems |
| TRU64 | DEC/HP Alpha servers |
| Unix | Any UNIX or Linux operating environment |
| Unixware | SCO UNIX OS for Intel and AMD processors |

# Language Identifiers

The following table gives an overview of the three-character language identifiers that can be used in Software Management to specify a language, for example, through the variable *localeID*.

| Language Identifier | Language |
|---|---|
| Any | Any language / language-independent |
| CHS | Simplified Chinese |
| CHT | Traditional Chinese |
| CSY | Czech |
| DAN | Danish |
| DEU | German |
| ELL | Greek |
| ENU | English (U.S.) |
| ESP | Spanish |
| FIN | Finnish |
| FRA | French |
| HUN | Hungarian |
| ITA | Italian |
| JPN | Japanese |
| KOR | Korean |
| NLD | Dutch |
| NOR | Norwegian |
| PLK | Polish |
| PTB | Brazilian Portuguese |
| PTG | Portuguese |
| ROM | Romanian |
| SVE | Swedish |
| TRK | Turkish |

# Template Parameters and pifproto Command Switches

The following table shows the parameters predefined in the standard templates of the PIF SDK and the corresponding pifproto command switches:

| Template Parameter | pifproto Command Switch |
|---|---|
| [$PIF_PRODUCT_PLATFORM] | -a |
| [$PIF_PRODUCT_NAME] | -n |
| [$PIF_PRODUCT_VERSION] | -v |
| [$PIF_INSTALL_PATH] | -d |
| [$PIF_COMMENT] | -c |
| [$PIF_SUPPLIER] | -s |
| [$PIF_PRODUCT_BUILD] | -B |
| [$PIF_PRE_INSTALL] | -I |
| [$PIF_POST_INSTALL] | -i |
| [$PIF_PRE_REMOVE] | -r |
| [$PIF_POST_REMOVE] | -R |

# Chapter 8: Frequently Asked Questions

Following are frequently asked questions that occur while working with or installing the PIF Packager and Installer. Solutions for problems and proposed actions to continue working are provided.

This section contains the following topics:

Log File Collection Tool dsminfo (see page 153)
Self-installing PIF Product can be Splitted (see page 154)
Register Multiple PIF Files As One Product (see page 154)
The Installer Uses Directories on a Customer System (see page 155)
The Packager Uses Directories to Store its Files (see page 155)
X11 Emulation Needs to be Configured (see page 155)
Packager and Installer Require a Certain Type of Java Runtime Environment (see page 156)
Product Installation Logged in a File (see page 156)
Selective Uninstall Supported (see page 157)
Interaction Between a Prepif Script and a Response File During Silent Installations (see page 157)
PIF Products Created With the Packager of PIF SDK Version 4.1 Installable on Unicenter Software Delivery 4.0 Agents (see page 158)
Packager Installation on a Computer that has no DVD Drive (see page 158)

## Log File Collection Tool dsminfo

CA Technologies provides the dsmInfo tool, which collects diagnostic information from systems that have CA ITCM installed. The data collected is compressed into a single file that contains log files, system information, directory structures, and registry and environment information. This diagnostic tool is available in the CA ITCM product installation media under the DiagnosticTools folder.

If a problem with CA ITCM is reproducible, then run the following command to change the trace level to DETAIL:

```
cftrace -c set -l DETAIL
```

Reproduce the problem and collect the diagnostic information with the dsmInfo tool.

**Notes:**

For more information about this tool, see the DSMInfoReadMe.txt file available under the DiagnosticTools folder in the product installation media.

The dsmInfo tool produces ".7z" files by default. These files provide better compression than zip files, so uploading to CA Technologies is easier.

# Self-installing PIF Product can be Splitted

**Symptom:**

I need both a self-installing PIF product and a PIF product for software delivery purpose. Do I need to provide two packages?

**Solution:**

You do not need to provide two complete packages. Use the command pifself with the option -s when creating the self-installing PIF product. This creates three files in the same destination directory:

- a start script to extract the PIF product
- a self-installing shell script (ca-sm-installer) containing the binaries of the Installer
- the actual PIF product

When the self-installing PIF product is used, the shell script is executed. At execution time, the corresponding (actual) PIF product must be located in the same directory.

When only the PIF product is needed for software delivery purpose, just use the single PIF product file for registration.

# Register Multiple PIF Files As One Product

**Symptom:**

I have multiple PIF product files belonging to the same product. Can I register multiple files as one software delivery product?

**Solution:**

**To register multiple PIF product files as one software delivery product**

1. Put all files belonging to the PIF product (for example, the PIF product itself, some PIF files containing external components and some PIF files containing language resources) into the same directory.
2. Create the required administrative data by executing the sd_registerproduct command with the option -R in that directory.
   A “reginfo” structure is created in the directory.
3. Register the product in the Software Package Library by dragging-and-dropping the directory onto the Explorer. In this specific case, do not use the context menu of the Explorer (Register, PKG, PIF or RPM package), because this would register a single PIF product file only.

# The Installer Uses Directories on a Customer System

**Symptom:**

I install the Installer using a self-installing PIF product. Which directories does the Installer use or create on the customer system?

**Solution:**

The Installer stores its files in the $CASHCOMP/installer directory. This directory will be either created or updated on the customer system.

The program lsm is added to /usr/bin.

A link is created from /opt/CA/installer to $CASHCOMP/installer.

# The Packager Uses Directories to Store its Files

**Symptom:**

I install the Packager using a self-installing PIF product. In which directories on the development system does the Packager copy its files?

**Solution:**

The Packager stores the files in /opt/CA/packager. The file etc/.profile is extended by the following environment variables:

- $SDPcker, pointing to the home directory of the Packager,
- $PATH, $LD_LIBRARY_PATH and $MANPATH, which are expanded by Packager directories.

If not already present, the Installer will be installed as well.

# X11 Emulation Needs to be Configured

**Symptom:**

Do I need to configure the X11 emulation to be able to use the GUI of the Packager?

**Solution:**

X11 is a windowing system designed for client/server architecture. The X11 server is the system where your application starts. The X11 client presents this graphical application. The client and server can run on the same system, for example, on a local console.

- On the X11 server side, access for the X11 client must be configured, as follows:

  Register your local system with X11:
  `"xhost +client_name"`

- On the client side, X11 client software must be installed and activated, as follows:

  Set the $DISPLAY variable to the X11 client's node name:
  `DISPLAY=client_name:0; export DISPLAY`

# Packager and Installer Require a Certain Type of Java Runtime Environment

**Symptom:**

My system has a gcj Java installed. Can it be used for Java mode dialogs?

**Solution:**

The gcj Java implementation does not support several features provided by a standard SUN JRE 1.2 and higher. For example, libgcj is missing java.awt. For graphical dialog mode it is therefore required to have a real standard JRE as provided by SUN installed.

# Product Installation Logged in a File

**Symptom:**

Can I find information about the product installation in a log file?

**Solution:**

Yes, Software Management provides an installation log file.

The name of the log file is *product_name.job*.log. You can find this file at the following location:

$CASHCOMP/installer/log/*product_name.job*.log

# Selective Uninstall Supported

**Symptom:**

I install a PIF product that contains an external component. Then I install another product by not using the Installer and software delivery. One of the soft links of the latter product points to the external component in the PIF product. As a result, software delivery does not know that this product uses the external component. Is it possible to get a warning that when I remove the external component other products might be disabled, and can I get the option to not removing the external component?

**Solution:**

Yes, the PIF SDK provides a solution to prevent components from being removed from the target computer. You can add a pre-remove script to the product using the #prerm keyword in the prototype file. This pre-remove script must check whether the component is being used by other products, and exit with a non-zero exit value. Then, the Installer leaves the component unchanged on the computer.

# Interaction Between a Prepif Script and a Response File During Silent Installations

**Symptom:**

I create and edit a response file on my own, and I create a prepif script that gets run before product installation. When I run a silent install, will the response file get passed into the prepif script, and if so, will any changes there overwrite the values passed into the install? When I set the install path in the response file and then the prepif script sets the default installdir, which value will be used?

**Solution:**

A prepif script is executed only the first time the PIF product is installed. During an update installation this script is not executed! The purpose of the prepif script is to migrate non-PIF installations. After migration, you can change the PIF_INSTALL_JOB and your product-specific PIF install path parameter. The path in the response file is overwritten. Note, that you can overwrite only your product-specific path that has been defined using #ppath in the prototype file. The PIF_INSTALL_PATH is for informational use only and cannot be overwritten.

# PIF Products Created With the Packager of PIF SDK Version 4.1 Installable on Unicenter Software Delivery 4.0 Agents

**Symptom:**

I am building PIF products using the Packager of the PIF SDK Version 4.1. Can I install these PIF products with software delivery agents of Version 4.0 running on the target computers?

**Solution:**

Due to functional enhancements in the PIF SDK Version 4.1, incompatibilities cannot be excluded in your case. We recommend that you build your PIF product as a self-installing PIF product and register it as a software delivery product (of type “command”). In this case, the Installer present on the target computer is automatically updated at installation time with the current Installer version included in the self-installing PIF product.

# Packager Installation on a Computer that has no DVD Drive

**Symptom:**

I want to install the Packager for Linux and UNIX on a computer that has no DVD drive. How must I proceed?

**Solution:**

If you want to install either the Packager for Linux and UNIX or the Command Line Interface (required to register a package from Linux or UNIX to a manager on Windows), you must download the Packager directory structure (using FTP or SFTP, for example). Then run install-sm-packager.sh from the directory on the Linux computer that contains the whole downloaded Packager directory.

# Index


## A

action script • 45

## B

backup • 102

## C

CASHCOMP • 19, 66
character-oriented install mode • 18
check box • 55
check box tree • 46, 55
combined parameter value • 34
combo box • 55
comment for parameter • 34
compatibility • 69
configuration component • 25
configuration file • 27

## D

dependencies to PIF products • 31
dialog dynamically detected • 43
dialog element grayed out • 54
dialog elements • 54
dialog verification • 43
directories in PIF products • 27
directories used by the PIF Packager • 155
directories used or created by the Installer • 155
disk space reservation • 28
dynamic parameter value • 34

## F

file area • 56
file naming conventions • 105
font specification for installation dialogs • 47

## G

generation with pifmk • 134
global parameter • 33
global parameter in optional required products • 35
global parameter set through environment • 35
graphical or character-oriented install mode • 18
graphics areas in installation dialogs • 53

## I

incompatible dependency • 31
install • 104
install PIF Packager • 18
installation directory field • 57
installation folder for Installer • 19
installation folder for PIF Packager • 19
installation log file • 43, 156
Installer (PIF)
    Installer (PIF) compatibility • 69
    Installer (PIF) functions overview • 65
    Installer (PIF) installation folder • 19
installing PIF product with previous Installer version • 158
install-sm-packager command • 19
internal parameters during product install • 36

## J

Java Runtime Environment (JRE) required • 40, 156
JRE in self-installing PIF product • 17, 40

## L

label • 57
language identifiers • 151
language-specific component • 26
library.dct • 139
logging • 43
long running scripts support • 59

## M

main product path • 36
mandatory dependency • 31
migration of non-PIF products • 68
multi-language support • 39
multiple PIF files registered as one product • 154

## N

navigation button • 57

## O

optional component • 27
optional dependency • 31

## P


Packager
- PIF Packager install command • 19
- PIF Packager installation • 18
- PIF Packager installation folder • 19
- PIF Packager installation program • 19
- PIF Packager installation without DVD drive • 158
- PIF Packager removal • 20

packages to provide • 154
parameters • 32
- parameter comment • 34
- parameter default value • 34
- parameter internal • 36
- parameter type • 33
- parameter value combined • 34
- parameter value dynamic • 34
- parameter value static • 34
- parameter value type • 34

password field • 59
patch product • 22
patch product generated automatically • 140
PIF_ (internal parameters) • 36
PIF_CHANGECONTROL_ • 36
PIF_CHANGEPARAM_ • 54
PIF_CONTROL_NAME • 36
PIF_CREATION_DATE/TIME • 36
PIF_DIALOG_LANGUAGE • 36
PIF_DIALOG_MODE • 36
PIF_DIALOG_NAME • 36
PIF_INSTALL_JOB • 36, 101
PIF_INSTALL_MODE • 36
PIF_INSTALL_PATH • 36
PIF_LOCALE • 36
PIF_LOG_FILE • 43
PIF_MAINPRODUCT_DIR • 36
PIF_PREINSTALL_DIR • 27, 36, 100
PIF_PRODUCT_NAME • 36
PIF_PRODUCT_PLATFORM • 36
PIF_PRODUCT_VERSION • 36
PIF_PROGRESSBAR_ • 59
PIF_PROGRESSLABEL_ • 60
PIF_RESPONSE_FILE • 36
PIF_RETAIN_CONFIGURATION • 27, 36
PIF_SUPPLIER • 36
pifextract command • 137
piflib command • 139
pifmk command • 134
pifpatch command • 140
pifproto command • 131
platform-specific component • 25
post-install scripts • 104
post-interview script • 102
preinstall components • 27
pre-install directory • 27, 100
pre-install progress • 100
preinstall scripts • 103
pre-install tasks • 100
pre-PIF script and response file interaction • 157
private parameter • 33
product family upgrade • 68
progress area • 59
progress bar • 59
progress during installation preparation phase • 100
progress label • 60
progress of a script • 31, 59
properties box • 61
prototype file • 131
published parameter • 33


## R


radio button • 60
read values from the environment • 103
registerproduct command • 141
response file for PIF products • 36


## S


scripts • 29
- script execution progress • 31, 59
- scripts timer • 31

sd_registerproduct command • 141
selective deinstall • 157
self-installing PIF product • 66, 148, 154
splash screen • 45
standard component • 25
standard file • 27
standard installation dialog • 42
standard template • 39
static parameter value • 34
symbolic link • 27


## T


template parameters • 152
templates for PIF products • 39
temporary file • 27
text area • 61
text field • 61

toolbox • 54

## U

update product family • 68

## V

validate software prior to installation • 100
validate values entered in an installation dialog • 43
validation script • 43
virtual file • 28
virtual package name • 24

## X

X11 emulation • 155