2 Software Installation

2.1 Introduction

All software products developed for use with the PowerOpen operating system must follow the guidelines described here if they are to be installed and updated with the **installp** command. All PowerOpen Systems must support installp environment as defined in this chapter. This Chapter is a recommend practice guide for applications at this time. Other methods can be used for installation of applications.

The **installp** command installs and updates software products. It is designed so the most common installation tasks can use common code. Tasks that are specific to a particular software product can be provided by the product owner.

The **installp** command works in conjunction with other executables that can optionally be provided with the product installation or update package. If an installation or update package does not contain those executables, the default installation shell scripts are used. For more information on **installp**, see Installp Processing of Product Packages on page 2−6.

2.2 Creating Software Product Installation Packages

This section describes the requirements for a software product to be installed and updated with the **installp** command. In order to be installed by the **installp** command, a software product or product update must provide an installation or update package that contains certain files that control the installation or update process, as well as the files that are actually installed.

The format and contents of a PowerOpen-formatted software product installation package are described in Contents of a Software Package on page 2−18.

2.2.1 Package Design Requirements

Every installation package and procedure must adhere to the following requirements:

- Do not require user interaction for the installation. If user interaction is required to configure the product option, then that configuration must be performed outside the installation.
- Select the least disruptive state that predictably accomplishes the installation or update from the following:
  - Devices inactive
  - Files inactive
  - Subsystems stopped
  - Applications inactive
  - Single-user mode
- Ensure that the system is in the required state. If the state is not correct, display a descriptive message to enable the user to put the system in the required state and return an error.
- Do not require a system restart unless the installation or update cannot be accomplished otherwise. Require user interaction to perform the system shutdown and restart if it is required.
- Ensure that all installations of or updates to interdependent software product options can be performed at the same time. This might require that a subsystem be stopped.
- Ensure that the prerequisite file is set correctly for the order in which installation and update procedures must occur.
- Ensure that the documentation for the software product installation or update is complete by specifying what the user is expected to do and when the user is expected to do it.

2.2.2 Library and Shared Library Requirements

Whenever adding files to an existing library, during either an installation or update procedure, list the files in the **lpp.acf** file. The default **instal** and **update** script does the following to ensure the library is updated:

- Copies the library to a working directory using a temporary name. Be sure that the library name does not conflict with any existing names.
- Adds any files to the copy of the library (the temporary library).
- Moves the temporary library name to the old library name.

When the installation or update procedure creates a new library, consider the library users when determining the installation order. For example, before installing the executable files of a software product, install the libraries on which the product depends.

2.2.3 Packaging of Product Options

An installable entity within a software product package is called a *product option*. The installation package for a software product can be divided into several separately installable options. This is allowed because some software products are very large and can have many pieces that can be used independently of other pieces. When pieces of the product can be used independently, dividing the product into separately installable options allows the end user to install only those options that are needed.

Installation packaging has been divided into three parts — **share**, **usr**, and **root**. These three parts are necessary in order to use a product in a diskless/dataless environment. All parts of a software product are installed in a standard system.

Diskless and dataless workstations cannot start (boot) by themselves; they must use a remote server. *Diskless workstation* refers to a workstation that uses a remote server to boot and does not have a hard disk. *Dataless workstation* refers to a workstation that uses a remote server to boot but does have a hard disk. Systems that are not diskless / dataless workstations are called standard systems.

A Shared Product Object Tree (SPOT) is a remote file system on a server that stores software that can be used by remote diskless and dataless clients. This remote file system contains those files which are part of the **/usr** file system for a standard system.

In order to support installation in the diskless and dataless environment, the installation packaging of a software product has been divided into three parts. The three parts contain the following:

share Contains that part of the product that can be shared among several machines, even if they have a different hardware architecture. This can include non-executable files, such as documentation. Sometimes, data files (e.g., fonts) might also be included in the **share** part of a product. For a standard system, files are stored in the **/usr/share** file tree.

usr Contains that part of the product that can be shared among several machines with the same hardware architecture. For a standard system, these files are stored in the **/usr** file tree.

root Contains that part of the product that cannot be shared among machines. In a diskless environment these are the files for which there must be a unique copy for each client of a SPOT. Most of the software for which there must be a separate copy for each individual machine is associated with the configuration of the machine or product. For a standard system, files in this part are stored in the **root** (/) file tree.

The **share** part is optional because products may not have any files that can be shared among different hardware platforms. The **share** part is always packaged in a separate installation package.

The **usr** and **root** parts of a product are packaged in the same installation package. Every product has a **usr** part. For products that have separately installable options, every option will always have a **usr** part. The **root** part of a product or product option is optional because some may not have configuration or files that need to be specific to the individual machine.

Following is a brief view of the directories in the root, **/usr**, and **/usr/share** file systems and a description of what can be found in the directories. This can be used as a basis for the splitting of a product package into **root**, **usr**, and **share** parts.

/ (**root**) contains the following:

/dev Local machine devices

/etc Machine configuration files such as **hosts** and **passwd**

/sbin	System utilities needed to boot; very few commands
/var	Variable local data files such as spool and mail files

/usr contains the following:

/usr/bin	Commands and scripts
/usr/sbin	System administration commands
/usr/include	Include files
/usr/lib	Libraries, non-user commands, and architecture-dependent data
/usr/lpp	Installed software products
/usr/share	File system for machine independent files, such as **man** and **terminfo**

2.2.4 The installp Processing of Product Packages

There are four major actions that can be taken with the **installp** command — apply, commit, reject, and deinstall. These are described below.

Apply When an option within a product update package is applied, the update is installed, and information is saved so that the update can be removed later. Software updates that have been applied can be committed or rejected later. When an option within a product installation package is applied, it is installed on the system and committed, whether or not the end user has specifically requested a commit.

Commit When software is committed, the user is making a commitment to that version of the product option, and the information saved during the apply is removed from the system. Committing already applied software does not change the currently active version of a product option; it merely removes information saved for the purpose of reverting back to the previous version of the software.

Reject When an update is rejected, information saved during the apply is used to change the active version of the product option to the version previous to the rejected update. The saved information is then removed from the system. The reject operation is valid only for updates; it is not valid for the installation level of a software product option.

Deinstall When a product option is deinstalled, the option and its installed updates are removed from the system independent of their state (applied, committed, or broken). The deinstall operation is valid only for the installation level of a software product option; it is not valid for updates.

Processing for the apply, reject, and deinstall operations can be tailored by executables provided within the product package. Therefore, it is necessary to have some understanding of the **installp** processing for these operations. This processing is described below.

2.2.4.1 Processing for the Apply Operation

This section describes the steps taken by the **installp** command when a product option or update is applied.

1. Restore the **lpp_name** file for the product package from the specified device.
2. Verify that the specified software product options exist.
3. Check the level of the software product to be installed or updated.
4. Restore control files from the **liblpp.a** archive library file into the appropriate directory (**/usr/lpp/***Product_Name* for **usr** or **usr/root** packages and **/usr/share/lpp/***Product_Name* for **share** packages).

5. Check disk space requirements.
6. Check that necessary requisites are already installed or are on the list to be installed.
7. If this is an installation package, search the Software Vital Product Data (SWVPD) to see if *Product_Option* (the product option being installed). Also, check to see if any product options listed in the *Product_Option***.namelist** file are already installed on the system at any level. If so, write the option names and level to the *Product_Option***.installed_list** file.
8. If this is an installation package, call the **/usr/lib/instl/rminstal** script to do the following for each option being installed:
   a. Execute *Product_Option***.pre_rm** if it exists.
   b. If *Product_Option*.**cfgfiles** exists, then copy the list of files contained in it to the configuration save directory.
   c. If *Product_Option* is currently installed
      Remove files and SWVPD information (except history) for *Product_Option*.
   else
      If *Product_Option***.rm_inv** exists
         Remove files and SWVPD inventory information for files listed in the file
         *Product_Option***.rm_inv**
   If a file being removed is the last file in the inventory for a product option
         Remove files and SWVPD information (except history) for any product option.
   else
    If *Product_Option***.installed_list** exists and contains only one product option
       Remove files and SWVPD information (except history) for that product option.
9. For each part of a product package (i.e., **usr**, **root**, or **share**)
   a. Set INUTREE (*U* for **usr**, *M* for **root**, and *S* for **share**) and INUTEMPDIR environment variables.
   b. If an installation package
      If **instal** control file exists in current directory
         Execute **./instal**.
      else
         Execute the default script **/usr/lib/instl/instal**.
   else /* update package */
      Set INUSAVEDIR environment variable.
      If **update** control file exists in current directory
         Execute **./update**.
      else
         Execute the default script **/usr/lib/instl/update**.
   c. If a **status** file exists
      Use **status** file to determine success/failure of each option.
   else
      Assume all options failed.
   d. If apply operation was successful for an option
      Update the Software Vital Product Data (SWVPD) database.

else
Run cleanup (**lpp.cleanup** from current directory or default **/usr/lib/instl/cleanup**).

10. Remove files from configuration save directory for those files not designated to remain saved.

The following files are referenced by either **instalp** or the **rminstal** script, which is invoked by **installp.** These files are described in detail in Description of Installation Control Files on page 2−26

- This list of files can be included in the package in the **liblpp.a** archive file.

  Product_Option.**cfgfiles**
  Product_Option.**namelist**
  Product_Option.**pre_rm**
  Product_Option.**rm_inv**

- This list of files can be created

  Product_Option.**installed_list**

The **install** or **update** executable is invoked from **installp** with the frist parameter being the device being used for the istallation or update. The second parameter being the device being used or the installation or update. The second parameter is the full path name to the file containing the list of option to be installed or updated, referred to below as **$OPTIONLIST.**

The following files are referenced by the default **instal** and **update** script. They are described in detail in Description of Installation Control Files on page 2−29.

- The following list of files can be included in the package in the **liblpp.a** archive file:

  Product_Option.**al**
  Product_Option.**inventory**
  Product_Option.**pre_i** (installation packages) or *Product_Option*.**pre_u** (updates)
  Product_Option.**post_i** (installation packages) or *Product_Option*.**post_u** (updates)
  Product_Option.**err**
  Product_Option.**trc**
  Product_Option.**odmdel**
  Product_Option.**odmadd**
  Product_Option.**config** (installation packages) or *Product_Option*.**config_u** (updates)

- The following list of files can be created by the **instal**/**update** script:

  Product_Name.**s**
  Product_Option.**rodmadd**
  Product_Option.**undo.err**
  Product_Option.**undo.trc**
  status

The flow within the default **instal** and **update** script is as follows:

1. Do the following while reading the **$OPTIONLIST**.

   a. If update
      Execute *Product_Option*.**pre_u** if it exists.

else
Execute *Product_Option*.**pre_i** if it exists.

b. Build a master apply list by appending *Product_Option*.**al** to the new file **master.al**.

c. Append this option to the new option list, **$OPTIONLIST.new**.

2. Call **inusave** to save current versions of the files.

3. If root part
Call **inucp** to copy files for root part.
else,
Call **inurest** to restore files for usr or share parts.

4. Do the following while reading the **$OPTIONLIST.new**.

a. If update
Execute *Product_Option*.**post_u** if it exists.
else
Execute *Product_Option*.**post_i** if it exists.

b. If *Product_Option*.**cfgfiles** exists, then call **/usr/lib/instl/migrate_cfg** to handle processing of configuration files according to their handling method.

c. Execute the **sysck** command using the *Product_Option*.**inventory** file if it exists.

d. Execute the **tcbck** command using the *Product_Option*.**tcb** file if it exists.

e. Execute **errupdate** if *Product_Option*.**err** exists.

f. Execute **trcupdate** if *Product_Option*.**trc** exists.

g. Execute the *Product_Option*.**odmdel** script if it exists.

h. Execute **odmadd** if *Product_Option*.**odmadd** exists.

i. If update,
Execute *Product_Option*.**config_u** if it exists.
else
Execute *Product_Option*.**config** if it exists.

j. Update the **status** file.

5. Link files into the **deinstl** directory for future use.

2.2.4.2 Processing for the Reject Operation

This section describes the steps taken by the **installp** command when an update to a product option is rejected.

1. Check requisites to ensure that all dependent product updates are also rejected.

2. For each part of a product package (i.e., **usr**, **root**, or **share**)

a. Set INUTREE (*U* for **usr**, *M* for **root**, and *S* for **share**) and INUTEMPDIR environment variables.

b. If **reject** control file exists in current directory
Execute **./lpp.reject**.
else
Execute the default script **/usr/lib/instl/reject**.

3. Update the Software Vital Product Data.

The **reject** executable is invoked from **installp** with the first parameter being undefined and the second parameter being the full path name to the file containing the list of options (referred to below as **$OPTIONLIST**) to be rejected for the update.

The following files are referenced by the default **cleanup** and **reject** script. They are described in detail in Description of Installation Control Files on page 2−29.

- The following list of files can be included in the package in the **liblpp.a** archive file:

  Product_Option.**al**
  Product_Option.**inventory**
  Product_Option.**unpre_i** (installation packages) or *Product_Option*.**unpre_u** (updates)
  Product_Option.**unpost_i** (installations) or *Product_Option*.**unpost_u** (updates)
  Product_Option.**unconfig** (installations) or *Product_Option*.**unconfig_u** (updates)

Note: *Product_Option*.**unpre_i**, *Product_Option*.**unpost_i**, and *Product_Option*.**unconfig** are used only for cleanup since installation packages cannot be rejected.

- The following list of files can be created by the **instal**/**update** script:

  Product_Option.**undo.err**
  Product_Option.**undo.trc**
  Product_Option.**unodmadd** or *Product_Option*.***.unodmadd**
  Product_Option.**rodmadd** or *Product_Option*.***.rodmadd**
  Product_Name.**s** (note this is referenced only for **cleanup**)

The flow within the default **cleanup** and **reject** script is as follows:

1. Do the following while reading **$OPTIONLIST**.

   a. If update
      Execute *Product_Option*.**unconfig_u** if it exists.
      else
      Execute *Product_Option*.**unconfig** if it exists.

   b. If *Product_Option*.***.unodmadd** and/or *Product_Option*.**unodmadd** exist, execute them.

   c. If *Product_Option*.***.rodmadd** and/or *Product_Option*.**rodmadd** exist, execute them.

   d. Execute **trcupdate** if *Product_Option*.**undo.trc** exists.

   e. Execute **errupdate** if *Product_Option*.**undo.err** exists.

   f. Execute **tcbck** if *Product_Option*.**tcb** exists.

   g. Execute **sysck** if *Product_Option*.**inventory** exists.

   h. If update
      Execute *Product_Option*.**unpost_u** if it exists.
      else
      Execute *Product_Option*.**unpost_i** if it exists.

   i. Build a master apply list (called **master.al**) from *Product_Option*.**al** files.

   j. Add *Product_Option* to **$OPTIONLIST.new**.

2. Do the following if **$INUTEMPDIR/master.al** exists.
   a. Change directories to / (**root**).
   b. Remove all files in **master.al**.
3. Do the following while reading **$OPTIONLIST.new**.
   a. Call **inurecv** to recover all saved files.
   b. If update,
      Execute *Product_Option*.**unpre_u** if it exists.
      else
      Execute *Product_Option*.**unpre_i** if it exists.
   c. Delete the install/update control files.
4. Remove the *Product_Name*.**s** file.

2.2.4.3 Processing for the Deinstall Operation

This section describes the steps taken by the **installp** command when a product option is deinstalled.

1. Check requisites to ensure that all dependent product options are also deinstalled.
2. For each part of a product package (i.e., **usr**, **root**, or **share**)
   a. Set INUTREE (U for **usr**, M for **root**, and S for **share**) and INUTEMPDIR environment variables.
   b. Change directory to INULIBDIR.
   c. If **deinstal** control file exists in current directory
      Execute **./lpp.deinstal**.
      else
      Execute the default script **/usr/lib/instl/deinstal**.
3. Remove files belonging to the product option from the file system.
4. Remove product option entries from the SWVPD except for history data.

The **deinstal** executable is invoked from **installp** with the first parameter being the full path name to the file containing the list of options to be deinstalled, referred to below as **$OPTIONLIST**.

The following files are referenced by the default **deinstal** script. They are described in detail in Description of Installation Control Files on page 2−29.

- The following list of files can be included in the package in the **liblpp.a** archive file:

  Product_Option.**al**
  Product_Option.**inventory**
  Product_Option.**unpre_i**
  Product_Option.**unpost_i**
  Product_Option.**unconfig**

- The following file can be delivered with the installation package:

  Product_Option.**unconfig_d**

- The following list of files can be created by the **instal/update** script:

  Product_Option.**undo.err**

*Product_Option***.undo.trc**
*Product_Option***.unodmadd** or *Product_Option***.*.unodmadd**
*Product_Option***.rodmadd** or *Product_Option***.*.rodmadd**

The flow within the default **deinstal** script is as follows:

1. Do the following while reading **$OPTIONLIST.**
   a. For each update to the product option
      If *Product_Option***.undo.err** exists, execute **errupdate**.
      If *Product_Option***.undo.trc** exists, execute **trcupdate**.
      If *Product_Option***.*.rodmadd** and/or *Product_Option***.rodmadd** exist, execute them.
      If *Product_Option***.*.unodmadd** and/or *Product_Option***.unodmadd** exist, execute them.
   b. For the base level product option
      If *Product_Option***.undo.err** exists, execute **errupdate**.
      If *Product_Option***.*.rodmadd** and/or *Product_Option***.rodmadd** exist, execute them.
      If *Product_Option***.*.unodmadd** and/or *Product_Option***.unodmadd** exist, execute them.
      If *Product_Option***.undo.trc** exists, execute **trcupdate**.
2. If *Product_Option***.unconfig_d** exists
      Execute *Product_Option***.unconfig_d**.
   else
      If *Product_Option***.unconfig** exists, execute it.
      If *Product_Option***.unpost_i** exists, execute it.
      If *Product_Option***.unpre_i exists**, execute it.

Note: If an error is returned from some call during the execution of the **deinstal** executable, the error will be logged, but execution will continue. This is different from the other scripts because execution for that option is normally canceled once an error is encountered. However, once the deinstallation of an option has begun, there is no recovery; therefore, deinstallation becomes a best effort once an error is found.

2.2.4.4 Executables That Can Replace Default Scripts

The **installp** command works in conjunction with other executables that can be provided with the product installation or update package. For those executables which are necessary to **installp** processing, a system default script is supplied. The **installp** command calls system default scripts unless the following executables are provided in the product package:

instal Called if an option within a software product installation package is applied. When this executable is not provided with the product package, the default script **/usr/lib/instl/instal** is called.

update Called if an option within a software product update package is applied. When this executable is not provided with the product package, the default **/usr/lib/instl/update** (which is a link to the **/usr/lib/instl/instal** script) is called.

lpp.cleanup Called if an option within a software product installation or update has been partially applied and must be cleaned up to put the product option back into a consistent state. When this executable is not provided with the installation or update package, the default script **/usr/lib/instl/cleanup** is called.

lpp.reject Called if an option within a software product update package is rejected. When this executable is not provided with the product package, the default **/usr/lib/instl/reject** (which is a link to the **/usr/lib/instl/cleanup** script) is called.

lpp.deinstal Called if an option within a software product installation package is deinstalled. When this executable is not found in the **/usr/lpp/***Product_Name* directory, the default script **/usr/lib/instl/deinstal** is called.

Each of the preceding executables must be written at a level so they can process all of the selected options for the software product package.

The default installation scripts have provisions for making calls to other executables if they are present in the installation or update package. These calls are placed strategically throughout the scripts to allow the product developer flexibility in tailoring an installation for tasks such as pre- and post-installation processing and configuration.

Vendor products must use the default installation scripts whenever possible and only supply those executable files for which calls are provided and for which the product has a specific need. The **instal**, **update**, **lpp.cleanup**, **lpp.reject**, or **lpp.deinstal** executable can be provided by the vendor when the default script is not sufficient to meet the needs of the product.

2.2.4.5 Requisite Checking

Whenever a software product has any dependencies on other software products being installed or already installed at a certain software level, requisite checking is necessary to verify that the dependencies are met. The requisite information can reference the level of a software product to a particular version number, release number, modification level, fix level, or fix ID (*vv.rr.mmmm.ffff.ppppppppp*). Requisite checking can apply to both installing new software products and updating existing software products. Requisite information is put into the **lpp_name** file.

The following requisite tests are supported:

Prerequisite Test The prerequisite test checks that the prerequisite product option is at the specified software level and is already installed or exists in the list of product options to be installed. In the latter case the **installp** command will reorder the list of product options, if necessary, to guarantee that prerequisite products are installed before their dependents.

Co-requisite Test The co-requisite test is similar to the prerequisite test, except that co-requisites can be installed in any order, whereas prerequisites require a specific order. A product option that specifies a co-requisite can be installed before the co-requisite within an installation session. At the end of an installation session, warning messages are given for any unmet co-requisites which resulted from failures during installation. Co-requisites most commonly exist between options of the same software product.

If-requisite Test The if-requisite test is similar to the co-requisite test, except that it tests for the product option level only if the if-requisite product option is installed. If that product option or update is not installed the if-requisite test is ignored. If-requisites most commonly exist between updates to different product options within the same product or across products.

Install-requisite Test The install-requisite test requests that the product option be installed if and only if the install-requisite product option is at the specified software level and is already installed or exists in the list of product options to be installed. This differs from the prerequisite test in that the **−g** flag (to automatically include requisites) never operates on this requisite type.

Group Requisites As well as having simple checks for one product option at a specified level, requisite groups can be created and tested for. A requisite group is a group of product options and their specified levels, for which not all product option tests must be successful in order for the group to pass. The number of required successes for a group is specified in the first line of a group definition. The success or failure of a requisite group is based on the number of requisite tests that passes. If the number of tests that passes exceeds the limit specified for the group, the whole group passes. If the number of tests that passes does not exceed the limit specified for the group, the whole group fails.

Requisite checking is done twice during an installation to verify that requisite requirements are met. Initial checking is done with the data in the **lpp_name** file before any installing or updating occurs. This is done by a call from the **installp** command to the requisite-checking routines, and the requisite checking is performed on the entire list of product options that are passed to **installp**. This initial call compares the current state of the system with the requirements of the product options to be installed or updated. If the option which automatically includes requisite software (the **−g** flag) is specified to the **installp** command, the requisite checking routines will add necessary requisites to the list of options specified. When all requisites have been checked and resolved, the options are ordered according to prerequisites.

The second requisite check is a call from the **installp** command to check the requisites for the options within a specific product package immediately before the call to the **instal** or **update** executable for that product package. The purpose of this second check is to ensure that any requisites for this product package that were in the list of product options to be installed did install correctly and that all requisite requirements are satisfied.

2.2.4.6 Supersedes Processing

The **installp** command provides the following special features for installing updates to software products which supersede (replace) other updates:

- If an update that is requested for installation is not present on the installation medium specified to the **installp** command, the request to install the update can be satisfied by any superseding updates that are on the installation medium.
- If a software product has requisites on an update which is not on the installation medium but a superseding update is, then the requisite can be satisfied by the superseding update.
- If an update to a software product is either explicitly OR implicitly (via a requisite expression from an explicitly requested package) requested for installation, the request will be satisfied by the newest superseding update on the installation mediumwhen the **−g** flag (automatically include requisites) has been specified.
- If an update to a software product on the installation medium is requested for installation along with a superseding update which is also on the installation medium, only the newer update will always be selected by **installp** for installation.

Whenever the **−g** flag is specified on the **installp** command to automatically include requisite software, any update that has been requsted for installation (either wxplicitly or implicitly) will always be satisfied by the newest superseding update on the installation medium. In all cases except that described by the llast bullet above, the end user can explicitly control each update that is installed on the system by merely not using the **_g** flag and explicitly listing all updates that are to be installed.

In the last case, if a user wishes to apply a certain update and its superseding updated frim the installation medium, thane the user must do this by doing two separate **installp** operations: one for the earlier update and then another for its later superseding update. Note that this kind of operation would be neaningless if the tow updates were to be both applied and committed (**−ac**) since the process of committing the second update essentially removes the first update from the system.

2.3 The Software Product Installation Package

This section describes the format and contents of the software product installation package that must be supplied by the product owner. It gives a description of the files, some required and some optional, which are part of a product installation or update package.

2.3.1 Format of a Software Package

An installation or update package must be one file in backup format that can be restored by the **installp** command during installation. This file can be distributed on tape, diskette, or CDROM. See the section Format of Distribution Media on page 2−45 for information concerning the format that must be used to place product packages on each type of media.

2.3.2 Rules and Naming Conventions

Software product packages are divided into product options that can be installed separately; a product package can contain only one installable option. In this section the term *product name* refers to the entire software product package. The term *product option name* refers to the entire name of an option within the product package.

2.3.2.1 Software Product and Product Option Names

The following conventions must be used when naming a software product and its options:

- All product option names must be of the form *Product_Name.Option* where *Option* is unique for the installable option and *Product_Name* is the name of the software product package.
- If a product has only one installable option, then it is allowed for the product option name to be the same as the product name. (This name cannot end in a dot.)
- All product names must be unique. Having two different software products with the same product name is not allowed. (The one exception to this is that if a product has a **share** part, which must be packaged separately, then the package for the **share** part can have the same product name as the package for the **usr** part or **usr** and **root** parts.)
- A product name from one software product package cannot be equal to a product option name from within another software product package.
- The software product name is contained in the **lpp_name** file, and it can have as many dot-separated levels as desired (e.g., **bos.net**).
- All product option names must be made up of ASCII characters.
- Product option names must be greater than one character in length and must begin with a letter or an underbar (_). Subsequent characters must be a letter, a digit, an underbar, or a dot (.).
- The maximum length for a product option name is 144 bytes.

2.3.2.2 Software Product Level

The level of a software product is defined as *vv.rr.mmmm.ffff.ppppppppp*, where

- *vv* is a numeric field of 1 to 2 digits that identifies the version number.
- *rr* is a numeric field of 1 to 2 digits that identifies the release number.
- *mmmm* is a numeric field of 1 to 4 digits that identifies the modification level.
- *ffff* is a numeric field of 1 to 4 digits that identifies the fix level.
- *ppppppppp* is a character field of 1 to 9 characters that identifies the fix ID and is only used for updates.

- All product options within a given product package must have the same version, release, modification level, fix level, and fix ID (for updates) values.
- For all new releases of a product installation package, the software product level must be monotonically increasing for the *vv.rr.mmmm.ffff* part of the field in order for the **installp** command to perform a proper check for a later level of the product on subsequent installations.

  The higher precedence of the four fields goes from left to right (i.e., 3.2.0.0 is a higher level than 2.3.0.0).
- The fix level field is ignored by the installation commands other than for the check for monotonically increasing levels. (It is present only for backwards compatibility.)

The following rules apply to the fix ID field:

- The fix ID field is required for the product level of all update packages.
- The fix ID field is not allowed as part of the product level for a product installation package.
- The fix ID field must be made up of ASCII characters. The first character must be a letter. Subsequent characters must be letters or digits.
- All fix IDs must be unique within a product.
- All fix IDs that begin with U4 are reserved.

2.3.2.3 File names

The following naming conventions apply to files delivered with the software product:

- File names can contain non-ASCII characters.
- File names cannot contain a comma or a colon. Commas and colons are used as delimiters in the *Product_Option*.**inventory** file, and the processing of this file by the **sysck** command will not occur correctly for filenames that contain commas or colons.

2.3.2.4 Other Rules

- The **lpp_name** file must be the first file in the backup/restore file that makes up a product installation package.
- The **copyright** file must be the first file archived into the **liblpp.a** file.
- All updates for a product option must have either a prerequisite on the installation level of the product option or a prerequisite on another update for the product option.
- The content field of the **lpp_name** file (**U** for **usr** part or **B** for both **usr** and **root** parts) must reflect the parts being updated for an update package and not the parts contained in the original installation package.
- Because some product options can exist that do not actually install new files but only add information to existing system files, the **liblpp.a** file need not have an apply list file (*Product_Option*.**al**) or an inventory file (*Product_Option*.**inventory**) for each product option. An example might be a product option that only adds stanzas to an existing ODM (Object Data Manager) database.
- For any product−provided files that are included in the **liblpp.a** archive file, it is recommended that the following naming conventions be used:
  - If the file is to be used in the installation of a specific produvt option, the file name should begin with "*Product_Option*"
  - If the file is to be used as a commom file for several product options, the file name should being with "**lpp.**"

2.3.3 Contents of a Software Package

This section gives a description of the files that are contained in the backup format file that makes up an installation or update package.

A description of the files that are contained in the backup format file and that make up an installation or update package is given below. File path names are given for installation packages. For update packages, wherever *Product_Name* is part of the path name, it is replaced by *Product_Name*/**inst**_*FixID* where *FixID* is the fix ID of the update.

The backup format file that makes up an installation or update package for the **usr** part of a software product must contain the following files:

- **./lpp_name**

  This is the software product name file. It provides information about the **usr** part of the software product to be installed or updated. See The lpp_name File on page 2−19 for more information.

- **./usr/lpp/**_Product_Name_/**liblpp.a**

  This file is an archive file that contains control files that are used by the installation process for installing or updating the **usr** part of the software product. See Description of Installation Control Files on page 2−29 for specific information about files contained in this archive library. An example of the path to the **usr** part **liblpp.a** file is **./usr/lpp/bos.adt/liblpp.a** for the **bos.adt** software product installation package, and the path to the **usr** part **liblpp.a** file is **./usr/lpp/bos.adt/inst_U409991/liblpp.a** for a **bos.adt** software product update package with the fix ID of U409991.

- All files, backed up relative to root, that are to be restored for the installation or update of the **usr** part of the software product.

In addition to the preceding files, the backup format file that makes up the package must contain the following files if the software product has a **root** part that is to be installed or updated:

- **./usr/lpp/**_Product_Name_/**inst_root/liblpp.a**

  This library file contains control files that are used by the installation process for installing or updating the **root** part of the software product. An example of the path to the **root** part **liblpp.a** file is **./usr/lpp/bos.adt/inst_root/liblpp.a** for the **bos.adt** software product installation package, and the path to the root part **liblpp.a** file is **./usr/lpp/bos.adt/inst_U409991/inst_root/liblpp.a** for a **bos.adt** software product update package with the fix ID of U409991.

- All files that are to be restored for the installation or update of the **root** part of the software product; instead of being backed up relative to root, these files must be backed up relative to **./usr/lpp/**_Product_Name_/**inst_root** for installation packages or relative to **./usr/lpp/**_Product_Name_/**inst**_*FixID*/**inst_root** for update packages.

If the software product has a **share** part, then it must be packaged in a separate installation package from the **usr** and **root** parts. The backup format file that makes up an installation or update package for the **share** part of a software product must contain the following files:

- **./lpp_name**

  This file provides information about the **share** part of the software product to be installed or updated.

- **./usr/share/lpp/**_Product_Name_/**liblpp.a**

  This library file contains control files that are used by the installation process for installing or updating the **share** part of the software product. An example of the path to the **share** part **liblpp.a** file is **./usr/share/lpp/bos.adt/liblpp.a** for the **bos.adt** software product installation package, and the path to the **share** part **liblpp.a** file is **./usr/share/lpp/bos.adt/inst_U409991/liblpp.a** for a **bos.adt** software product update package with the fix ID of U409991.

- All files, backed up relative to root, that are to be restored for the installation or update of the **share** part of the software product.

2.3.3.1 The lpp_name File

The software product name file, **lpp_name**, provides the **installp** command with information about the software product package and each installable option within it. This file must be contained inside each software product package. Refer to Figure 2−1. The numbers and arrows in the figure refer to fields that are described in the table that follows.

```
4 R  S xlccmp {
xlccmp.obj  01.01.0000.0000.U412345 1 N U En_US AIX XL C Compiler/6000 # ...
[
>0 {
lppname1  v>2
}
%
/bin  6
/usr/lpp/xlc/bin 469
/usr/lpp/SAVESPACE 4744
%
xlccmp.obj U411111
xlccmp.obj U412222
%
NetLS_vendor_id
NetLS_product_id
NetLS_product_version
]
}
```

Figure 2−1. Example lpp_name File.

Table 2−1 defines the fields in the **lpp_name** file.

Table 2−1. Fields in the lpp_name File.

Field Name	Format	Separator	Description
1. Format	Integer	White space	Indicates the release level of PowerOpen for which this package was built. The format values are: 1 — Pre-PowerOpen version 2 — Pre-PowerOpen version 3 — Pre-PowerOpen version 4 — Version 1 PowerOpen
2. Platform	Character	White space	Indicates the platform for which these software products were built. The values are: ***E—AIX/ESA (370)*** ***P—PS/2*** ***R—RISC System/6000***
3. Package Type	Character	White space	Indicates whether this is an installation or update package and what type. The values are: I—installation S—single update G — single update, gold SR — single update, required M — maintenance packaging update MC — cumulative packaging update ME — enhancement packaging update
4. Product name	Character	White space	The name of the product package (called *Product_Name*), which must be unique. This name is stored in the product database and is used to form the path name of the directory where the control files from the product package are unpacked.
	{	New line	Indicates the beginning of the repeatable section.
5. Option name	Character	White space	The complete name of the installable option (called *Product_Option*); this must begin with the product name.
6. Level	*vv.rr.* *mmmm.ffff.* *ppppppppp*	White space	The software level of the option to be installed. The values are: *vv*—Version *rr*—Release *mmmm*—Modification level *ffff*—Fix level *ppppppppp*—Fix ID (Optional).
7. Diskette Volume	Integer	White space	Indicates the diskette volume number where this product package is located if shipped on diskette.

Table 2−1. Fields in the lpp_name File.

Field Name	Format	Separator	Description
8. Quiesce / bosboot	Character	White space	Indicates whether the subsystem must be stopped before updating this option and whether a bosboot is needed following the installation. The values are: Y=Must stop subsystem; do not invoke bosboot N=Do not need to stop subsystem; do not invoke bosboot B=Must stop subsystem; must invoke bosboot b=Do not need to stop subsystem; must invoke bosboot
9. Content	Character	White space	Indicates the contents of this option. The content field is used to determine which SWVPD is updated for this option. The values are: B — **usr** and **root** part D — Microcode H — **share** part U — **usr** part
10. Language	Character	White space	Indicates the language supported by this installable option.
11. Description	Character	# sign or new line	Description of the installable option.
12. Comments	Character	New line	(Optional) Additional comments.
	[	New line	Indicates the beginning of the installable option information.
13. Requisite information	Shown below	New line	(Optional) Installation dependencies that this option has on other software. See section below for detailed description.
	%	New line	Indicates the separation between requisite and size information.
14. Size information	Shown below	New line	Size requirements for this option by directory. See section below for detailed description.
	%	New line	Indicates the separation between size information and supersede information.
15. Supersede information	Shown below	New line	(Optional) Contains a list of the fix IDs that this update supersedes. See section below for detailed description.
	%	New line	Indicates the separation between supersede information and NetLS information.

Table 2–1. Fields in the lpp_name File.

Field Name	Format	Separator	Description
16. NetLS vendor ID	Character	New line	(Optional) The vendor ID registered with NetLS as the owner of the product. This field is used if the product uses NetLS license passwords. This and the following two fields should not be present if the product option does not use NetLS license passwords.
17. NetLS product ID	Character	New line	The ID registered with NetLS as the ID of the product. This field must be present if and only if the NetLS vendor ID is present.
18. NetLS product version	Character	New line	The value registered with NetLS as the version of the product. This field must be present if and only if the NetLS vendor ID and product ID are also present.
	]	New line	Indicates the end of the installable option information.
	}	New line	Indicates the end of the repeatable section.

Table 2–1. Fields in the lpp_name File.

This file is a variable record length file. The first line contains four tokens, delimited by one or more blank characters, that specify the format, platform, package type, and product name followed by a set of braces ({ }) that contains information about the separate software product options.

Within the braces is one record for each software product option. The record for each software product option contains the following:

- A line containing several fields of information, separated by white space. The fields include the following:
  - product option name
  - level
  - diskette volume
  - quiesce/bosboot
  - content
  - language
  - description
- Optionally, a # (pound sign) or new line followed by comments.
- Optionally, a new line followed by a set of [ ] (brackets) that contain the following:
  - Optional requisite information. This information can contain a set of { } (braces) if group requisites are used. See the description in Requisite Information below.
  - Optional % (percent sign) used as a delimiter.

– Size information; sometimes required. See the description in Size Information on page 2–25.

– Optional % (percent sign) used as a delimiter.

– Optional supersede information. Only valid for updates. See the description in Supersede Information on page 2–26 for more information.

– Optional % (percent sign) used as a delimiter.

– Optional NetLS information (vendor ID, product ID, product version). Only used if product uses NetLS license passwords.

Note: To distinguish and delimit the sets of information contained within the brackets, use the % (percent sign); trailing % are not needed.

2.3.3.1.1 Requisite Information

The requisite information consists of information about installation dependencies on other software products. If a software product has any dependencies on other software products being installed or installed at a certain product level, supply a prerequisites file. Requisite information can reference a software product's level to a particular version number, release number, modification level, fix level, or fix ID (*vv.rr.mmmm.ffff.ppppppppp*).

A description of supported requisite tests can be found in Requisite Checking on page 2–13. These requisites and their associated key words are:

- Prerequisite; key word is ***prereq**
- Co-requisite; key word is ***coreq**
- If-requisite; key word is ***ifreq**
- Install-requisite; key word is ***instreq**

Each record (line of text) in the requisite information describes a single requisite check or defines the relationship for a group of requisites.

The first field (following a space or tab) in a single requisite check record can be either one of the key words (*prereq, *coreq, *ifreq or *instreq), or it can be the name of the dependent software product. If it is the name of a software product, the key word *prereq is assumed. If the first field is a key word, then the next field must be the dependent product name. The fields following the dependent product name field define the acceptable software level for the dependent software product.

The acceptable software level is defined by one or more logical expressions using the letters *v*, *r*, *m*, *f* and *p* and a numeric or alphanumeric value associated with the letter. The letters specified represent the version number (*v*), release number (*r*), modification level (*m*), fix level (*f*), and fix ID (*p*) that are required of the specified software product. The letters *v*, *r*, *m*, and *f* have numeric values. The letter p can have an alphanumeric value. The accepted symbols within the logical expressions are <, =, or >. Each logical expression is considered one field. The fields in each record are separated by one or more spaces or tabs.

An exception to the above rule is permitted by an extension to the if-requisite type of requisite. The *v, r, m, f* information can be specified in the logical expression separated by dots and enclosed in parentheses: `(v.r.m.f)`. The *r, m,* and *f* values are optional, and the parenthetic expression must be followed by a fix ID expression.

For example, the following record indicates that *Version1*, *Release3* or later of a software product named database is a required prerequisite:

```
database v=1 r>2
```

This could also be expressed as:

```
*prereq database v=1 r>2
```

In the following example, the record indicates that if the software product named spreadsheet is installed on the system it must be version 2 or later (if-requisite test). If product spreadsheet is not installed on the system, the test is ignored.

```
*ifreq spreadsheet v>1
```

In the following example if the software product spreadsheet is installed at any version and release, then it must have fix U412345 installed (or being installed) in order to satisfy the requisite record. (The fix U412345 must belong to the product at version 1, release 2.)

```
*ifreq spreadsheet v=1 r=2 p=U412345
```

As a final variation of the if-requisite check, the following record specifies that if spreadsheet is installed at *v=1 r=2 m=3*, then it must have the fix U412345 installed (or being installed) in order to satisfy the requisite record. If spreadsheet is not installed or is installed at a different *vrm*, then the test is ignored.

```
*ifreq spreadsheet (1.2.3) p=U412345
```

It is also possible to specify more than one version number, release number, modification level, fix level, or fix ID for a certain software product by specifying the or option (*o*) between the values for a particular specification. For example, the following record indicates that Version 1, Release 3, and either Modification 1 or 2 of the software product named timemgr is required.

```
*prereq timemgr v=1 r=3 m=1 o=2
```

A relationship between a group of requisite records can be defined by a record in the requisite information beginning with the greater than symbol (>), followed by an integer and then a { (left brace). Following this relational group-definition record are one or more single requisite records. The relationship is closed by a record containing only a } (right brace). The integer on the first record in the group relationship declares how many of the software products listed are required.

For example, the following relational prerequisites record specifies that at least one of the software products specified must be installed; both can be installed. If installed, the spreadsheet_1 software product must be *Version1*, *Release2*, and the spreadsheet_2 software product version must be later than *Version1*.

```
>0 {
*prereq spreadsheet_1 v=1 r=2
*prereq spreadsheet_2 v>1
}
```

Suppose a product has a prerequisite on the product spreadsheet that can be satisfied by *Version 1, Release 2*; *Version 1, Release 3*; or *Version 2, Release 0* of spreadsheet. The following relational prerequisites record could be used to satisfy the prerequisite on spreadsheet.

```
>0 {
*prereq spreadsheet v=1 r=2 o=3
*prereq spreadsheet v>1
}
```

It is possible to state any sort of relationship with this format. For example, relational requisite groups can be nested. However, a complex relationship requires a complex set of statements and checking. It is possible to include statements in the requisite information that are mutually exclusive and that check for a certain number of failures.

2.3.3.1.2 Size Information

The size information consists of information about the disk space requirements for the components of a software product option. This information is used by the install process to ensure that enough disk space is available to allow the installation or update to succeed.

The permanent-space and temporary-space specifications are numeric fields specifying disk space in 512-byte blocks. The permanent-space value represents disk space needed after the installation is complete. The temporary-space value represents disk space needed during installation but not afterward. The temporary space field is optional. Additionally, the product owner can specify **PAGESPACE** or **INSTWORK** in the full-path name field to indicate disk space requirements for paging space and work space needed during the installation process.

Size information is required for product packages of type I, S, G, and SR. It is not required for product packages of type M, MC, and ME (packaging updates); since these package types consist only of a list of requisite packages, the size information is not needed.

Field 1 Contains the full path name of the directory that has size requirements.

Field 2 Contains the size (in 512-byte blocks) of the permanent space needed for the installation or update. Permanent space is space that is needed after the installation completes. This field can have a different meaning in the following cases:

If field 1 is **PAGESPACE** field 2 represents the size of page space needed (in 512-byte blocks) to perform the installation.

If field 1 is **INSTWORK** field 2 represents the number of 512-byte blocks that are needed by **installp** for extracting control files that are used during an installation. These control files are the files that are archived into the **liblpp.a** library file. The number of blocks required for the un-extracted **liblpp.a** file is listed as temporary space for **INSTWORK**.

Field 3 Contains the size (in 512-byte blocks) of the temporary space needed for the installation. Temporary space is released after the installation completes. A good example of temporary space is the space needed to rebind an executable object file. Another example is the space needed to archive an object file into a library. To archive into a library, the **installp** command makes a copy of the library, archives the object file into the copied library, and then moves the copied library over the original library. The space for the copy of the library is considered temporary space. This temporary space field is optional.

The following is an example of size information:

```
/usr/bin        30
/lib    40      20
PAGESPACE       10
INSTWORK        10   6
```

Because it is difficult to predict how disk file systems are mounted in the file tree, the directory path name entries in this file must be as specific as possible. For example, it is better to have an entry for **/usr/bin** and one for **/usr/lib** than to have a combined entry for **/usr**, because **/usr/bin** and **/usr/lib** can exist on different file systems that are both mounted under **/usr**. In general, it is best to include entries for each directory into which files are installed.

For update packages only, the size information must also include any old files (to be replaced) that will be moved into the save directory for use in restoring the old files if the update is later rejected. In order to indicate the size requirements to save the old version of files being replaced, an update package must specify the following special directory entries:

- **/usr/lpp/SAVESPACE** directory entry for usr part files
- **/lpp/SAVESPACE** directory entry for root part files

- **/usr/share/lpp/SAVESPACE** directory entry for share part files

The actual directories where the old versions of updated files are saved are the following:

- **/usr/lpp/***Product_Name***/inst_***FixID***.save** directory entry for usr part files
- **/lpp/***Product_Name***/inst_***FixID***.save** directory entry for root part files
- **/usr/share/lpp/***Product_Name***/inst_***FixID***.save** directory entry for share part files

2.3.3.1.3 Supersede Information

Supersede information is optional and is only applicable to update packages. This information consists of a list of the fix IDs that are superseded by (that is, they can be replaced by) this software product update.

The supersede information consists of one line per fix ID that is included in this software product update. Each line has two fields delimited by white spaces. The first field is the name of the product option. The second field is a fix ID that is superseded by this update.

The following is an example of supersede information:

```
bos.rte U498777
bos.rte U498666
```

For a product option, one update is said to supersede other updates for that product option if it contains all the fixes for the superseded updates. An example is given below to help explain supersedes.

Suppose there is a product option, **Product_A.option_1** at version 1.1.0.0, that delivers several files. Suppose three problems are later reported for this produvt option and **file_a** and **file_b** are fixed and updates sent out in the following fashion:

1. The first problem reported and fixed is in **file_a**. An update is created for **Product_A.option_1** at version 01.01.0000.0000.U400001 that includes the fix for **file_a**. This update does not supersede any other updates.
2. The second problem fixed is in **file_b**. An update is created for **Product_A.option_1** at version 01.01.0000.0000.U400002. Besides including the fix for **file_b**, this update also includes the fix previously made for **file_a** so that both fixes may be delivered to the customer in one update. The supersede information for update U400002 will contain the following:

```
Product_A.option_1 U400001
```

3. The fix for the third problem is also contained within **file_a**, like the first problem. The source modules used already contain the fix for the first problem, so the fixed **file_a** now contains fixes for problems one and three. An update is created for **Product_A.option_1** at version 01.01.0000.0000.U400003. Again, besides including **file_a**, which fixes problems one and three, the fix for the second problem will also be included by including **file_b** in the update. Therefore, update U400003 will deliver **file_a** and **file_b** and will include supersede information that contains the following:

```
Product_A.option_1 U400001
Product_A.option_1 U400002
```

2.3.3.2 The liblpp.a File

The software product installation library file, **liblpp.a**, contains the library of the software product's installation or update related files for the part of the product (**usr**, **root**, or **share**) that is being installed or updated. The **liblpp.a** file contains both data files and executable files that are used to control the processing of installation activities.

The files that are to be included in the **liblpp.a** library file are described in the following section. The files are separated into lists of data files and executable files. Some of the files only apply to update packages.

In the lists that follow, the word *Product_Option* is a variable that is replaced by the name of the separate option to be installed within the software product. For example, the apply list file for the **bos.net.tcp.client** option of the **bos.net** software product would be **bos.net.tcp.client.al**.

2.3.3.2.1 List of Data Files Contained in the liblpp.a File

The following data files can be part of the **liblpp.a** file for both installation and update packages.

fixinfo — A brief description of problems that have been fixed in this package.

lpp.acf — Archive control file; needed only if adding or replacing a member file to an archive file that already exists on the system.

lpp.doc — Can be used to describe any specific instructions that must be followed in order to install or use the product or update.

lpp.README — Information the user should read before using the software product.

Product_Option.**al** — Apply list; list of all files to be installed or updated for this software product option.

Product_Option.**cfgfiles** — List of configuration files that need to be saved from a previous version of the product option.

Product_Option.**cfginfo** — A file that can be used to signal **installp** that the system must be rebooted before the installation will take effect.

Product_Option.**copyright** — Required. This is copyright information for the product option. This file can be empty, but it must be present.

Product_Option.**err** — Error report templates.

Product_Option.**inventory**. — Specific information about each file in the software product option.

Product_Option.**mrg** — Use if the option updates the root part of the file tree and contains configuration files. This file contains specific information about configuration files that must be saved from a previous release.

Product_Option.**odmadd** — Contains stanzas to be added to any ODM (Object Data Manager) database.

Product_Option.**trc** — Trace report templates.

productid — Contains product identification information.

The following data files can optionally be a part of the **liblpp.a** file for installation packages only:

Product_Option.**namelist** — A list of any product option names that are different from the current name but that contained a previous version (or parts of a previous version) of this product option.

Product_Option.**rm_inv** — The list of files to be removed from the system if they are found installed. This file is designed to be used in cases where the product option is packaged differently from the previous version, and it is not correct to remove previously installed files based on the product option's entries in the inventory database.

2.3.3.2.2 List of Executable Files Contained in the liblpp.a File

The following executable files can optionally be a part of the **liblpp.a** file for both installation and update packages:

lpp.cleanup If not provided, the default script **/usr/lib/instl/cleanup** is used.

Product_Option.**odmdel** If present, is executed in the default **instal/update** script to save and delete any entries that are being replaced in any ODM databases.

Product_Option.**unodmadd**
If present, is executed in the default **cleanup/reject** script to delete any entries that were added to any ODM databases during the installation or update.

The following executable files can optionally be a part of the **liblpp.a** file for installation packages only:

instal If not provided, the default script **/usr/lib/instl/instal** is used.

lpp.deinstal If not provided, the default script **/usr/lib/instl/deinstal** is used.

Product_Option.**config** If present, is executed at the end of the default **instal** script to perform any necessary configuration processing.

Product_Option.**unconfig** If present, is executed in the default **cleanup/reject** script to undo any configuration processing that was performed during installation.

Product_Option.**pre_d** If present, is executed on a deinstall operation before files and data have removed.

Product_Option.**pre_rm** If present, is executed on an apply operation from the **rminstal** executable before any files or data from a prefious version of the product option have been removed.

Product_Option.**post_i** If present, is executed after the restore of files in the default **instal** script to perform any necessary post-installation processing.

Product_Option.**unpost_i** If present, is executed in the default **cleanup/reject** script to undo any post-installation processing that was done during installation.

Product_Option.**pre_i** If present, is executed after requisite checking and before save of previous version of files in the default **instal** script to perform any necessary pre-installation processing.

Product_Option.**unpre_i** If present, is executed in the default **cleanup/reject** script to undo any pre-installation processing that was done during installation.

The following executable files can optionally be a part of the **liblpp.a** file for update packages only:

update If not provided the default script **/usr/lib/instl/update**, which is a link to the default installation script, is used.

Product_Option.**config_u** If present, is executed at the end of the default **update** script to perform any necessary configuration processing.

Product_Option.**unconfig_u**
If present, is executed in the default **cleanup/reject** script to undo any configuration processing that was performed during the update.

Product_Option.**post_u** If present, is executed after the restore of files in the default **update** script to perform any necessary post-installation processing for the update.

Product_Option.**unpost_u** If present, is executed in the default **cleanup/reject** script to undo any post-installation processing that was done during the update.

Product_Option.**pre_u** If present, is executed after requisite checking and before save of previous version of files in the default **update** script to perform any necessary pre-installation processing for the update.

Product_Option.**unpre_u** If present, is executed in the default **cleanup/reject** script to undo any pre-installation processing that was done during the update.

The following executable files can optionally be a part of the **liblpp.a** file for update packages only:

lpp.reject If not provided, the default script **/usr/lib/instl/reject** (a link to the cleanup script) is used.

2.3.4 Description of Installation Control Files

2.3.4.1 Data Files

The following data files are control files used during the installation to control processing. Most are files provided by the product owner in the installation package; some are files created during the installation process for later use.

- The **fixinfo** File
- The **lpp.acf** File
- The **lpp.doc** File
- The **lpp.README** File
- The *Product_Name*.**s** File
- The *Product_Option*.**al** File
- The *Product_Option*.**cfgfiles** File
- The *Product_Option*.**cfginfo** File
- The *Product_Option*.**copyright** File
- The *Product_Option*.**err** File
- The *Product_Option*.**installed_list** File
- The *Product_Option*.**inventory** File
- The *Product_Option*.**mrg** File
- The *Product_Option*.**namelist** File
- The *Product_Option*.**odmadd** and *Product_Option*.***.odmadd** Files
- The *Product_Option*.**rm_inv** File

- The *Product_Option***.rodmadd** and *Product_Option.****.rodmadd** Files
- The *Product_Option***.trc** File
- The *Product_Option***.undo.err** File
- The *Product_Option***.undo.trc** File
- The **productid** File

2.3.4.1.1 The fixinfo File

The **fixinfo** file is an ASCII file that gives a brief description of problems that have been corrected in deliverable files included in the product package. This file is optional. It is usually included in a product update package to describe the bug fixes that are included in that update. Although not often used in an installation package, it could be used to describe all problems fixed from the previous version of the product.

2.3.4.1.2 The lpp.acf File

The **lpp.acf** file is only needed when the software product contains files that are to be archived into a library file that already exists on the system and is not part of the to-be-installed or to-be-updated software product option. The **lpp.acf** file consists of one or more entries, each representing a constituent file to be archived into a single library. Each entry consists of two fields delimited by white space. The two fields are defined as follows:

FileName — Refers to the path name, relative to root, for the file installed from the software product option.

ArchiveFileName — Refers to the path name, relative to root, of the already existing target library into which this file is to be archived.

For example, if a software product option contained a file named **doprnt.o** in the directory **./usr/lib/lpp/***Product_Name* that is to be archived into **/usr/lib/libcurses,** the entry in the **lpp.acf file** would be similar to the following:

```
./usr/lib/lpp/Product_Name/doprnt.o ./usr/lib/libcurses
```

Note: *FileName* must also be identified in the *Product_Option***.al** file for whichever option it is a part of. If *FileName* is not listed in the *Product_Option***.al** file it is not restored and archived into the target archive file.

The **lpp.acf** file must contain all entries for all options that are part of the software product.

2.3.4.1.3 The lpp.doc File

The **lpp.doc** file can be used to describe any specific instructions that must be followed in order to install or use this product or update.

2.3.4.1.4 The lpp.README File

The **lpp.README** file can be used to contain information the customer needs to know about the software product prior to using it.

2.3.4.1.5 The *Product_Name*.s File

The *Product_Name***.s** file is a status file created by the default **instal/update** script during the apply operation for use by the default **cleanup** script. It is used to determine what parts of the installation or update processing were completed and, therefore, what parts need to be backed out.

2.3.4.1.6 The *Product_Option*.al File

The *Product_Option.***al** file contains an apply list for a software product option. The installation apply list contains the names of all the files, one per line, to be restored from the distribution medium during installation or update. The file path names must be listed relative to the root directory. For example, the file path names must appear as **./usr/lib/sendmail**. During the installation process these are restored to by appropriate system directories.

The **inurest** command, which archives and restores files for the **installp** command, restores these files to the appropriate system directories. The current directory is always set to root by the **inurest** command.

The entries in the *Product_Option*.**al** file can be used to locate the corresponding entries in an archive control file (/**usr**/**lpp**/*Product_Name*/**lpp.acf**) when a restored file is being archived.

2.3.4.1.7 The *Product_Option*.cfgfiles File

The *Product_Option*.**cfgfiles** file lists any configuration files that need to be saved in order to migrate the user from a previously installed version of the product option. The goal is to save any configuration data that is needed to migrate the user to a newly-installed version without losing any user-configured data.

For any product options that contain product configuration files that must be saved before the previous level of the product option is removed from the system, a *Product_Option* **. cfgfiles** file must be provided in the proper **liblpp.a** file (i.e., **usr**, **root**, or **share**). Files that must be saved are those that contain user configuration data that can be retrieved and used to re-establish the user's configuration.

The *Product_Option*.**cfgfiles** file must contain a list of configuration files to be saved. There must be a one-line entry for each file to be saved, and it must contain the file name as a path name relative to root and white space separator and the key word for the appropriate handling method for the migration of the file. The following are valid handling methods:

preserve The **preserve** key word means that **installp** will replace the installed new version of the file with the saved version from the save directory. After replacing the new file with the saved version, **installp** will remove the saved file from the configuration save directory.

user_merge The **user_merge** key word means that **installp** will leave the installed new version of the file on the system and that the old version of the file will be saved and kept in the configuration save directory where the end user will have access to it in helping to perform any user merge that may be necessary.

auto_merge Use of the **auto_merge** key word means that during the *Product_Option*.**post_i** processing the product-provided executables will merge necessary data from the installed new version of the file into the previous version of the file saved in the configuration save directory. After the *Product_Option*.**post_i** processing, the **installp** command will replace the installed new version of the file with the merged version in the configuration save directory (if it exists) and will then remove the saved file.

safe_merge Use of the **safe_merge** key word means that during the *Product_Option*.**post_i** processing the product-provided executables will merge necessary data from the installed new version of the file into the previous version of the file saved in the configuration save directory. However, the **installp** command will do no further processing on the file; the installed new version of the file will remain on the system, and the merged file will remain in the configuration save directory where the end user will have access to it for performing any further manipulation or for using to replace the installed file.

hold_new The **hold_new** key word means that **installp** will replace the installed new version of the file with the saved version from the save directory and that the new version of the file will be placed into the configuration save directory in place of the old version. This enables the end user to easily access the new version of the file.

other The key word **other** can be used in any case where none of the other defined handling methods are sufficient. In this case, the installp command will provide no support other than saving the file into the configuration save directory. Any other manipulation and handling of the configuration file will be done by the product-provided executables, and it will also be the responsibility of the product owner to document the handling of the file.

The product owner can use the *Product_Option*.**post_i** executable to accomplish any specific manipulating or merging of configuration data that cannot be accomplished via the generic work handled by the **installp** command.

Any configuration files that are listed in the *Product_Option* . **cfgfiles** file will be saved under the configuration save directory with the same relative path name as that used in the *Product_Option* . **cfgfiles** file. The name of the configuration save directory will be stored in the environment variable **MIGSAVE**. The save directory corresponds to the part of the package being installed. The following directories are the configuration save directories:

- **/usr/lpp/save.config** for the **usr** part
- **/lpp/save.config** for the **root** part
- **/usr/share/lpp/save.config** for the **share** part

If a product option needs to deal with a different list of files depending on the currently installed level of the product option, then the *Product_Option* . **cfgfiles** file must contain the entire list of configuration files that might be found, and the *Product_Option*.**post_i** executable (or other product-provided executables) must deal with those differences if necessary.

The **installp** command will not produce any warning or error messages if a file listed in the *Product_Option* . **cfgfiles** file is not found nor will it produce any messages for files that are not found during the phase following *Product_Option*.**post_i** processing when saved configuration files are processed according to their handling methods. If any warning or error messages are desired, they must be produced by the product-provided executables.

As an example of the *Product_Option* . **cfgfiles** file, suppose the product option **Product_X.option1** has three configuration files from which it needs to recover user-configuration data. If these files are all within the **root** part of the option, then the product owner will include the file **Product_X.option1 . cfgfiles** in the **root** part **liblpp.a** file, and the example file might contain the following:

```
./etc/cfg_xxx preserve
./etc/cfg_yyy user_merge
./etc/cfg_zzz preserve
```

2.3.4.1.8 The *Product_Option*.cfginfo File

The *Product_Option*.**cfginfo** file provides additional information about certain configuration actions that might need to take place for the installation of a product or update. This file can be supplied for the **usr** part, the **root** part, or both parts of a product.

Information contained in the *Product_Option*.**cfginfo** file is in the form of key words that are used to convey special configuration needs. Each key word must be followed by a new-line character as the separator between key words.

The following key word is currently defined:

BOOT Indicates that the system needs to be rebooted before the installation will take effect.

The **BOOT** key word can either be placed in the *Product_Option*.**cfginfo** file in the **liblpp.a** file in the product package, or it can be added during installation during the execution of product-provided configuration scripts.

The install scripts will check for the presence of the **BOOT** key word in the *Product_Option*.**cfginfo** file after the call to the last possible product-provided script so that a product owner has the flexibility to add the **BOOT** key word to the *Product_Option*.**cfginfo** file in any of the product-provided scripts. If any product dynamically adds the **BOOT** key word during its product configuration, it must append the key word if the *Product_Option*.**cfginfo** file already exists, rather than destroying an existing file.

The **BOOT** key word can be supplied in the *Product_Option*.**cfginfo** file in the **usr** part or the **root** part. If in the **usr** part, a message will be given from **installp** that the system and any diskless and/or dataless clients must be rebooted. If in the **root** part, the message will indicate that the system must be rebooted. It is believed that the **BOOT** key word should normally exist in the root part of a product, but there may be exceptions to this.

2.3.4.1.9 The *Product_Option*.copyright File

The *Product_Option*.**copyright** file contains the copyright information associated with the specified product option. This file must be present for each product option contained in the installation or update package. Copyright information for a software product option consists of the full name of the software product followed by the copyright notice.

The **installp** command displays the *Product_Option*.**copyright** file when installing an option within an installation package. When installing an update, **installp** only displays the *Product_Option*.**copyright** file if it is different from the existing copyright file for that product option.

2.3.4.1.10 The *Product_Option*.err File

The *Product_Option*.**err** file contains error report format templates for the software product. If a software product has to add error report format templates during installation, an *Product_Option*.**err** file must be supplied. The default **instal/update** script passes the *Product_Option*.**err** file to the **errupdate** command to add the error report format templates to the system.

2.3.4.1.11 The *Product_Option*.installed_list File

The *Product_Option*.**installed_list** file is created by the **installp** command when installing the product option from an installation package if it is found that the product option (or some form of it) is already installed on the system at some level. The SWVPD is searched to determine if either *Product_Option* or any product options that might be listed in the file *Product_Option*.**namelist** (if it exists) are already installed on the system. If so, the product option (or options) and the installation level are written to the *Product_Option*.**installed_list** file.

If the product option is alredy installed under the same name or a previous for ot the product option is installed under some other product option name, then the file *Product_Option*.**installed_list** will be abailable at the time for the calls to the **rminstal** and **install** executables. The *Product_Option*.**installed_list** file can exist for the **usr,root,** or **share** parts and will be located in the following directories:

- /**user**/**lpp**/*Product_Name* for the **usr** part
- /**lpp**/*Product_Name* for the **root** part
- /**sur**/**share**/**lpp**/*Product_Name* for the **share** part.

The *Product_Option*.**installed_list** file will contain a one−line entry for each product option that was installed. Each entry will contain the product opion name and the installation level

An example of this file is that if we are installing **Product_A.option_a** at level 1.2.0.0 and it is found that **Product_A.option_a** is already installed at level 1.1.0.0, then the file **Product_A.option_a.installed_list** will contain the following:

```
Product_A.option_a 01.01.0000.0000
```

Another example of this file would be where there is a *Product_Option*.**namelist** file involved. Suppose that for **Product_B.option_a** there is a **Product_B.option_a.namelist** file that contains the entries **Product_B.option_1** and **Product_B.option_2**. If we are installing **Product_B.option_a** at levl 2.3.0.0 and it is found that **Product_B.option_1** and **Product_B.option_2** are currently installed at level 2.2.0.0, then the file **Product_B.option_a.installed_list** will contain the following:

```
Product_B.option_1 02.02.0000.0000
Product_B.option_2 02.02.0000.0000
```

2.3.4.1.12 The *Product_Option*.inventory File

The *Product_Option*.**inventory** file contains specific information about each file that is to be installed or updated for the software product option. The file name, product name, type, checksum, size, link, and symlink information are entered into the SWVPD database by the **sysck** command.

It is required for options that install files into the **usr** or **share** parts of the file tree. If files are installed into the **root** part then this file is required for the **root** part. Conceivably, a software product could have a **root** part that does not contain files to be installed but only does configuration. In this case, the *Product_Option*.**inventory** file is not required for the **root** part.

The inventory file consists of ASCII text in stanza format. The name of the stanza is the full path name of a file that is part of the installation or update. The stanza name must be terminated by a : (colon) followed by a new-line character. The attributes of the file follow the stanza name and are in the form of `attribute=value` where `attribute` is replaced by one of the key words listed below. Each attribute entry is terminated by a new-line character.

The following are the valid attributes of a file:

Attribute	Description
type	Specifies the type of the file. The following key word values are valid file types:

Type	Meaning
FILE	Ordinary file.
SYMLINK	A Symbolic link to a file.
LINK	A hard link to a file.
DIRECTORY	Directory.
FIFO	First-in-first-out file.
BLK_DEV	Block device special file.
CHAR_DEV	Character device special file.
MPX_DEV	Multiplexed device special file.

Attribute	Description
owner	Owner of the file. This can be in the owner name or owner ID format.
group	File group. This can be in the group name or group ID format.
mode	Indicates the mode of the file. The value must contain the permissions of the file in octal format. The permissions value can be preceded by any of the following key words. Items in the list are separated by commas.

Mode Items	Meaning
tcb	Part of the Trusted Computing Base.

tp	Part of the Trusted Process.
svtx	Text will be saved on swap for this file.
suid	File has the set user ID bit set.
sgid	File has the set group ID bit set.

Note that for files of `type=SYMLINK` or `TYPE=LINK,the mide value must be 777 (mode=777)`

target Valid only for `type`=SYMLINK. This is the path name of the file that the file that the link points to. symbolic link points to.

program Software product to use to verify the file.

source Location of the original copy of the file.

size Size of the file in blocks.

checksum Checksum values of the file.

2.3.4.1.13 The *Product_Option*.mrg File

The *Product_Option*.**mrg** file is used to identify configuration files that are owned by the software product and system configuration files that can be modified by the installation or update process. An example would be the **/etc/hosts** file that is needed by TCP/IP. Essentially, the *Product_Option*.**mrg** file must contain the list of configuration files that might need to be saved if installing a new release of the software product.

2.3.4.1.14 The *Product_Option*.namelist File

The *Product_Option*.**namelist** file is used in cases where either the product option has had its name changed from a previous version or where the option contains files that were previously packaged in different options. In these cases the file *Product_Option*.**namelist** must be provided in the **usr. root** or **share** part **liblpp.a** file, and it must contain the names of all product options that previously contained files now included in this product option. Product option names contained in the file must be separated by a white space.

At the beginning of the installation of *Product_Option*, the **installp** command will search the Software Vital Product Data (SWVPD) to determine if either *Product_Option* itself or any other product option listed in the *Product_Option*.**namelist** file is already installed on the system, and the name and level of any option that is found installed will be written to the *Product_Option*.**installed_list** file so that this information will be available to product-provided executables. The *Product_Option*.**namelist** file is then removed.

The *Product_Option* . **namelist** file is only valid for installation packages; it is not valid for update packages.

A simple example of a *Product_Option* . **namelist** file is the case where a product option has simply changed its name from the previous version. The **bos.INed** product option for version 4.1 is an example of this; it was previously released as the **INed.obj** product option at version 3.2. The 4.1 **bos.INed** product package, therefore, contains the **bos.INed.namelist** file in its **usr** part **liblpp.a** file, and the **bos.INed.namelist** file contains the following entry:

```
INed.obj
```

Another type of example of a *Product_Option* . **namelist** file is the case where a product option contains files that were previously a part of two or more product options. Suppose that in its version 1.1 product package **Product_A** contains three options: **Product_A.option_a**, **Product_A.option_b**, and **Product_A.option_c**. However, for the new 1.2 version of **Product_A**, the product owner plans to combine **Product_A.option_a** and **Product_A.option_b** into one option and call it **Product_A.option_ab**. Therefore, the product owner needs to include in the **liblpp.a** file a **Product_A.option_ab.namelist** file with the following contents.

```
Product_A.option_a
Product_A.option_b
```

In the above example, since the name of **Product_A.option_c** remains unchanged, there does not need to be a **namelist** file for that option.

A third example of a *Product_Option* **. namelist** file is the case where a product option from a previous version is being broken into two or more product options for the current version. Suppose that **Product_B.option_1** is being divided into a client and a server part for its new version and that the new option names are **Product_B.option_1.client** and **Product_B.option_1.server**. Therefore, the **usr** part **liblpp.a** file for **Product_B** will contain a **Product_B.option_1.client.namelist** file that contains the following entry:

```
Product_B.option_1
```

Also, the **liblpp.a** file will contain a **Product_B.option_1.server.namelist** file that contains the following entry:

```
Product_B.option_1
```

Note: For the above example both **namelist** files contain the same data since both were previously a part of the same product option.

2.3.4.1.15 The *Product_Option*.odmadd and *Product_Option.**.odmadd Files

The *Product_Option***.odmadd** and *Product_Option.****.odmadd** files contain stanzas that will be added to ODM databases from the default **instal/update** script.

2.3.4.1.16 The *Product_Option*.rm_inv File

The *Product_Option***.rm_inv** file must be used in cases where there does not exist a one-to-one correspondence between a previous version of the product option and the current version of *Product_Option*. (The use of the term one-to-one correspondence here means that the files from the old product option only belong to the single new option and no other new option, and no more than one old option contributes files to the new option). The *Product_Option***.rm_inv** file contains the list of files, links, and directories to be removed from the system if they are found installed. This file is designed to be used in cases where the product option is packaged differently from the previous version, and it is not correct to remove previously installed files based on the product option's entries in the inventory database.

Note that a simple name change for a product option is not sufficient to require that the *Product_Option***.rm_inv** file be used. The file is necessary where a new product option is either a subset of an old option or is a mixture of parts of old product options. If a *Product_Option.***namelist** file exists and if it contains entries for more than one product option (thereby breaking the one−to−one correspondence rule), then the *Product_option***.rm_inv** file must be used to have previously installed versions of the files removed from the system.

The *Product_Option***.rm_inv** file consists of ASCII text in stanza format. The name of the stanza is the full path name of a file or directory that is to be removed if found on the system. The stanza name must be terminated by a : (colon) followed by a new-line character. A file or directory may have attributes following the stanza name in the form of `attribute=value` where `attribute` can be replaced by one of the key words below and is used to identify hard links or symbolic links that are also to be removed. Each attribute entry is terminated by a new-line character.

The following are valid attributes of a file in the *Product_Option***.rm_inv** file.

Attribute	Description
links	One or more hard links to the file. The full path names of the links are delimited by commas.
symlinks	One or more symbolic links to the file. The full path names of the links are delimited by commas.

As an example, take the case where a product option from a previous version is being broken into two or more product options for the current version. Suppose that **Product_B.option_1** contains the following files: **file_a**, **file_b**, **file_c**, **file_x**, and **file_y** (a symbolic link to **file_x**). Suppose that for a new version this option is being divided into a client and a server part and that the new option names are **Product_B.option_1.client** (which contains **file_a**, **file_b**, and **file_c**) and **Product_B.option_1.server** (which contains **file_x**, and **file_y**). Examples of the *Product_Option***.rm_inv** files that might be included in the **liblpp.a** file follow.

The following is an example of the **Product_B.option_1.client.rm_inv** file:

```
/usr/lpp/Product_B/bin/file_a:
/usr/lpp/Product_B/bin/file_b:
/usr/lpp/Product_B/bin/file_c:
/usr/lpp/Product_B/bin
```

The following is an example of the **Product_B.option_1.server.rm_inv** file:

```
/usr/lpp/Product_B/bin/file_x:
        symlinks=/usr/lpp/Product_B/bin/file_y
/usr/lpp/Product_B/bin
```

If the object to be removed is a directory, then that directory must be empty in order for it to be removed. Therefore, directories created and owned by a product option must be placed at the end of the *Product_Option***.rm_inv** file in the reverse order of creation in order for them to be properly removed.

2.3.4.1.17 The *Product_Option*.rodmadd and *Product_Option.**.rodmadd Files

The *Product_Option***.rodmadd** and *Product_Option.****.rodmadd** files are data files that contain stanzas needed to restore the ODM databases to the way they were before the *Product_Option***.odmdel** and *Product_Option.****.odmdel** procedures were run. They are created by the default **instal/update** script and are used in the default **cleanup/reject** script.

2.3.4.1.18 The *Product_Option*.trc File

The *Product_Option***.trc** file contains trace report format templates and event numbers for the software product. If a software product has to add trace report format templates during installation, an *Product_Option***.trc** file must be supplied. The default **instal/update** script passes the *Product_Option***.trc** file to the **trcupdate** command to add the error report format templates to the system.

2.3.4.1.19 The *Product_Option*.undo.err File

The *Product_Option***.undo.err** file is a file created by the default **instal/update** script when error record templates are added. It is used by the default **cleanup/reject** script to remove those templates.

2.3.4.1.20 The *Product_Option*.undo.trc File

The *Product_Option***.undo.trc** file is a file created by the default **instal/update** script when trace report templates are added. It is used by the default **cleanup/reject** script to remove those templates.

2.3.4.2 Executable Files

The following executable files are control files used during the installation to control processing. Most are files provided by the product owner in the installation package; some are files created during the installation process.

- The **instal** and **update** Files
- The **lpp.cleanup** File
- The **lpp.deinstal** File
- The **lpp.reject** File
- The *Product_Option***.config** File
- The *Product_Option***.config_u** File
- The *Product_Option*.**odmdel** and *Product_Option.**.**odmdel** Files
- The *Product_Option***.pre_d** File
- The *Product_Option***.pre_rm** File
- The *Product_Option***.post_i** File
- The *Product_Option***.post_u** File
- The *Product_Option***.pre_i** File
- The *Product_Option***.pre_u** File
- The *Product_Option*.**unconfig** File
- The *Product_Option***.unconfig_d** File
- The *Product_Option*.**unconfig_u** File
- The *Product_Option***.unodmadd** and *Product_Option.***.unodmadd** Files
- The *Product_Option***.unpost_i** File
- The *Product_Option***.unpost_u** File
- The *Product_Option***.unpre_i** File
- The *Product_Option***.unpre_u** File

An executable file can be either a shell script or an executable file in the **a.out** file format. If an executable file is a shell script written in a shell other than Bourne shell, then the shell indicator must be placed into the script (e.g., **!/usr/bin/ksh**).

2.3.4.2.1 The instal and update Files

A software product must either provide an executable named **instal** for installation packages or an executable named **update** for update packages, or it must be able to use the default script **/usr/lib/instl/instal** (which is linked to **/usr/lib/instl/update**). Use of the default script is recommended.

The **instal** or **update** executable is required if the default script is not used for a software product. The **instal** or **update** executable must process all of the options of the software product. It can invoke other executables, which can also be delivered with the installation package.

The **instal** or **update** executable for a software product must do any processing necessary to facilitate the installation of the software product. It can check requisites, restore all files associated with the software product, perform any configuration necessary for the installation of the software product, and can add error and trace templates, if desired.

The following commands are available to help with installation processing:

inucp Copies files from the **/usr/lpp/***Product_Name***/inst_root** directory to the / (**root**) file tree when installing the **root** part.

inurecv Recovers saved files for installation failure or software rejection (**installp −r**).

inurest Restores files from the distribution medium onto the system using an apply list as input.

inusave Saves all files specified by an apply list into the save directory belonging to the software product.

inuumsg Issues messages from the **inuumsg.cat** message catalog file for the software product being installed.

ckprereq Verifies compatibility of the software product with any dependencies using requisite information supplied in the **lpp_name** file and information about already installed products found in the SWVPD.

sysck Checks the inventory information during installation and update procedures.

The **sysck** command is in the **/usr/bin** directory. Other commands listed previously are in the **/usr/sbin** directory.

For examples of their use, refer to the default installation script, **/usr/lib/instl/instal**.

The **installp** command invokes the **instal** or **update** executable and passes two arguments to it. The first argument is the name of the device from which the installation is being done. This can be a hardware device, such as a tape drive, or it can be the path name of an installation image file or the path name of a directory that contains the installation image file. The second argument is the name of the file that contains a list of the product options to install. After the **installp** command invokes the installation or update script, it waits for a return code from it.

For example, a user chooses to install the **tcp.client** and **tcp.server** options of the **bos.net** software product from the **/dev/rmt0** device. The **installp** command invokes the installation script, passing to it the device name, **/dev/rmt0**, and the name of a file that contains the following option names:

```
bos.net.tcp.client 03.02.0000.0000
bos.net.tcp.server 03.02.0000.0000
```

To ensure compatibility with the **installp** command, the **instal** or **update** executable provided with the software product or product update must perform the following procedures:

1. Use the **inusave** command to save the current level of any files being installed.
2. Restore all required files for the **usr** part from the distribution medium with the **inurest** command.
3. Copy all required files for the **root** part from the **/usr/lpp/***Product_Name***/inst_root** directory with the **inucp** command.
4. Create a status file in the temporary directory indicating success or failure for each software product option being installed or updated.
5. Return an exit code indicating the status of the installation.

The **instal** or **update** executable, if provided by the software product owner, must process all of the options of the software product. It can either process the installation for all the options or invoke other executables for each option. If options fail, status information must be written to a file named **status**. The **installp** command sets an environment variable named **INUTEMPDIR** that contains the path of a temporary directory used during the installation. The **status** file must be placed in this directory (that is, status must be written to

$INUTEMPDIR/status). If the **instal** or **update** executable returns a nonzero return code and no **status** file is found, the installation process assumes that all options failed.

The **status** file must contain a one line entry for each product option that was to be installed or updated where each line contains the following:

- A one-character status code. Valid values are:
  - s Success, update SWVPD
  - f Failure, perform cleanup procedure
  - b Bypass, failed, cleanup not needed
  - i Requisite failure, cleanup not needed
  - v **sysck** verification failed
- The name of the product option that was to be installed or updated.

When a software product installs successfully, information about it is entered into the SWVPD database. If a software product option does not install successfully, there are two possible resolutions. If an **lpp.cleanup** executable has been provided in the **liblpp.a** file, then it is executed; otherwise, the default **/usr/lib/instl/cleanup** script is executed.

The following example of a **status** file indicates to the **installp** command that the installations for the **tcp.client** and **tcp.server** options of **bos.net** are successful and the installation for the **nfs.client** option is not successful.

```
s bos.net.tcp.client
s bos.net.tcp.server
f bos.net.nfs.client
```

2.3.4.2.2 The lpp.cleanup File

The **lpp.cleanup** file provides a procedure to recover from a failed software product installation or update. This procedure is optional. The **installp** command executes the cleanup procedure when an error occurs or when the **−C** flag is specified for the **installp** command. This cleanup procedure does not perform cleanup for previously installed software that is in the BROKEN state; software gets in the BROKEN state when, for a failed installation, the cleanup that is performed by the cleanup executable is itself unsuccessful.

The software product owner should provide this executable only if the default cleanup script, **/usr/lib/instl/cleanup**, is not sufficient. If this file is provided, then it must undo any actions performed in the **instal/update** executable; in general, it should perform the basic functions that are provided in the default **cleanup** script.

The contents of the **lpp.cleanup** file depend on the procedures implemented by the software product during installation. At the very minimum, the cleanup procedure must remove any files that were restored. If the software product has a configuration procedure, then the cleanup procedure must be able to undo the effects of the configuration procedure. If the software product has separate options to be installed, the cleanup procedure must be able to clean up separate options or carry out a separate cleanup procedure for each option.

The **installp** command calls the software product cleanup procedure if the software product installation or update fails. Two arguments are passed to the cleanup procedure. The first argument is the device name, which can generally be ignored. The second argument is the name of the file containing a list of the options that failed to install.

2.3.4.2.3 The lpp.deinstal File

The **lpp.deinstal** file is the deinstallation procedure. This procedure is optional. If a **lpp.deinstal** file does not exist, the default shell script **/usr/lib/instl/deinstal** is used.

Software product developers should provide the **lpp.deinstal** executable only if the default script is not sufficient.

2.3.4.2.4 The lpp.reject File

The **lpp.reject** file is the rejection procedure. This procedure is optional. If a **lpp.reject** file does not exist, the default shell script **/usr/lib/instl/reject** is used. The **/usr/lib/instl/reject** is a link to the **/usr/lib/instl/cleanup** script. Software product developers should provide the **lpp.reject** executable only if the default script is not sufficient.

2.3.4.2.5 The *Product_Option*.config File

The *Product_Option***.config** file contains the configuration procedure of the software product option for the installation package. The default **instal/update** script executes this procedure at the end of its installation processing (after all the files have been restored, the error and trace templates updated, and the inventory information checked and updated). Some steps that might be included in a configuration file are:

- Add a user ID or group ID that your software product requires.
- Add a daemon.

2.3.4.2.6 The *Product_Option*.config_u File

The *Product_Option***.config_u** file contains the configuration of the software product option for the update package. The default **instal/update** script executes this procedure at the end of its update processing (after all the files have been restored, the error and trace templates updated, and the inventory information checked and updated).

2.3.4.2.7 The *Product_Option*.odmdel and *Product_Option.**.odmdel Files

The *Product_Option***.odmdel** and *Product_Option.****.odmdel** procedures save and delete stanzas that are being replaced from any ODM databases. The saved stanzas must be saved into files named *Product_Option***.rodmadd** and *Product_Option.****.rodmadd** for use in the **cleanup/reject** procedure. If they exist the *Product_Option***.odmdel** and *Product_Option.****.odmdel** procedures are executed from the default **instal/update** script before any stanzas in the *Product_Option***.odmadd** and *Product_Option.****.odmadd** data files are added to the relevant ODM databases.

2.3.4.2.8 The *Product_Option*.pre_d File

The purpose of the *Product_Option***.pre_d** file is to provide a method by which product owners can control when deinstallation of a product option is allowed. The *Product_Option***.pre_d** is executed before removing any files or data for the product option; if this executable exits with a non-zero return code, then an error will be given, and the product option will not be deinstalled. Any specific error messages concerning why the product option is not allowed to be deinstalled or messages concerning actions that can be taken to change this status must be provided by the *Product_Option***.pre_d** executable.

Note: If the force option (−F flag) is used with the deinstall option, then the *Product_Option***.pre_d** file will not be executed, and the customer will be allowed to force the deinstallation of the product option.

2.3.4.2.9 The *Product_Option*.pre_rm File

The purpose of the *Product_Option***.pre_rm** file is to provide a method by which product owners can perform any work that is necessary before the removal of any files and software vital product data belonging to a previously installed version of a product option.

If the *Product_Option***.pre_rm** executable is found in the **liblpp.a** file, then it will be called before files and SWVPD information are removed from the system when removing that *Product_Option*. This provides a way for product owners to do any necessary work before the removal of files (other than saving off configuration files listed in *Product_Option***.cfgfiles**, which is done by **installp**).

The *Product_Option***.pre_rm** executable is called from the **rminstal** script that is used to remove the files for a product option that is to be removed. When an installation level of a product option is installed, any currently existing files of that product option are not saved as they might be when an update is installed; instead, the **rminstal** script is used to remove those files. File removal is based on the list of files in *Product_Option***.rm_inv** or on the file inventory in the SWVPD for the installed product option.

The product owner can provide a *Product_Option***.pre_rm** executable in **usr, root,** and **share** part **liblpp.a** files. Actions that would need to be part of a **root** part executable are those that need to be performed for each individual client root in a diskless and/or dataless environment.

An example of where a *Product_Option***.pre_rm** file might be used is if a product option has a need to query SWVPD information before the product option is removed.

2.3.4.2.10 The *Product_Option*.post_i File

The *Product_Option***.post_i** file contains the post-installation procedure of the software product option. The default **instal/update** script executes this procedure if it exists after doing the restore of the files of the software product option from the installation medium, but before updating the error and trace templates and executing the configuration procedure. A software product requires a post-installation procedure only if the software product option requires special steps to be performed at this point in the installation procedure. Many software product options do not require this file.

2.3.4.2.11 The *Product_Option*.post_u File

The *Product_Option***.post_u** file contains the software product option's post-update procedure. The default **instal/update** script executes this procedure if it exists after doing the restore of the software product option's files from the update medium, but before updating the error and trace templates and executing the configuration procedure. A software product requires a post-update procedure only if the software product option requires special steps to be performed at this point in the update procedure. Many software product options do not require this file.

2.3.4.2.12 The *Product_Option*.pre_i File

The *Product_Option***.pre_i** file contains the pre-installation procedure of the software product option. The default **instal/update** script executes this procedure if it exists after doing prerequisite checking but before saving any previous versions of the files of the product option. A software product option requires a pre-installation procedure only if the software product option requires special steps to be performed at this point in the installation procedure. Many software product options do not require this file.

2.3.4.2.13 The *Product_Option*.pre_u File

The *Product_Option***.pre_u** file contains the pre-update procedure of the software product option. The default **instal/update** script executes this procedure if it exists before saving any previous versions of the files of the product option. A software product requires a pre-update procedure only if the software product option requires special steps to be performed at this point in the update procedure. Many software product options do not require this file.

2.3.4.2.14 The *Product_Option*.unconfig File

The *Product_Option*.**unconfig** file backs out any procedures done with the *Product_Option*.**config** file. This procedure is executed at the beginning of the default **cleanup/reject** script if it exists.

2.3.4.2.15 The *Product_Option*.unconfig_d File

The purpose of the *Product_Option***.unconfig_d** file is to undo during the deinstallation of the product option any product-specific configuration that was done during the installation of the product option. This file is required for version 1 PowerOpen-formatted product packages if it is necessary for undoing any product-specific configuration.

The *Product_Option***.unconfig_d** file is an executable file that is applied with the installation level of the product option. It is not included inside the **liblpp.a** file as other controls files are. Including this file as a regular file inside the product package allows the file to be updated, if necessary.

Product options can have a **share** part, a **usr** part, or both **usr** and **root** parts. Product configuration can take place during installation of any part. Likewise, during deinstallation it can be necessary to undo configuration for any part of a product option. Undoing of configuration for **usr**, **root**, and **share** parts will be provided in the following files, respectively:

1. /**usr**/**lpp**/*Product_Name*/*Product_Option***.unconfig_d**
2. /**usr**/**lpp**/*Product_Name*/**inst_root**/*Product_Option***.unconfig_d**
3. /**usr**/**share**/**lpp**/*Product_Name*/*Product_Option***.unconfig_d**

The *Product_Option***.unconfig_d** file must be generic enough to undo the configuration of the product option during deinstallation no matter what updates are installed for that version/release level of the product option. If the configuration actions performed for a product option are changed because of an update, then it might be necessary to change the executable that undoes the configuration. If the *Product_Option***.unconfig_d** file is changed with an update, then it will probably be necessary for all subsequent updates to either prereq that update or to explicitly ship the updated *Product_Option***.unconfig_d** file.

If it is necessary to undo any product-specific configuration of a product during deinstallation, then the *Product_Option***.unconfig_d** file must be provided unless execution of the following files (in the order listed) will correctly undo the configuration:

1. *Product_Option***.unconfig**
2. *Product_Option***.unpost_i**
3. *Product_Option***.unpre_i**

The above files are those that are provided in the **liblpp.a** file of the installation level of a product for undoing product configuration, if necessary, for the reject of the product option. Since the actions performed in the above files are often the same as the actions necessary to undo the configuration during deinstallation, it is allowed for a product not to supply a *Product_Option***.unconfig_d** file if execution of the above performs the correct actions. If neither the *Product_Option***.unconfig_d** file nor any of the above three files exist, then it is assumed that it is not necessary to undo any product-specific configuration for deinstallation of the product option.

2.3.4.2.16 The *Product_Option*.unconfig_u File

The *Product_Option*.**unconfig_u** file backs out any procedures done with the *Product_Option*.**config_u** file. This procedure is executed at the beginning of the default **cleanup/reject** script if it exists.

2.3.4.2.17 The *Product_Option*.unodmadd and *Product_Option.**.unodmadd Files

The *Product_Option***.unodmadd** and *Product_Option.****.unodmadd** procedures delete any stanzas that were added to ODM databases during the installation or update procedure from the *Product_Option***.odmadd** and *Product_Option.****.odmadd** data files. If they exist, the *Product_Option***.unodmadd** and *Product_Option.****.unodmadd** procedures are executed after re-configuration in the default **cleanup/reject** script.

2.3.4.2.18 The *Product_Option*.unpost_i File

The *Product_Option*.**unpost_i** file backs out any procedures done with the *Product_Option*.**post_i** file. The default **cleanup/reject** script executes this procedure if it exists after restoring ODM databases to the correct level and before removing files applied for this level.

2.3.4.2.19 The *Product_Option*.unpost_u File

The *Product_Option*.**unpost_u** file backs out any procedures done with the *Product_Option*.**post_u** file. The default **cleanup/reject** script executes this procedure if it exists after restoring ODM databases to the correct level and before removing files applied for this level.

2.3.4.2.20 The *Product_Option*.unpre_i File

The *Product_Option*.**unpre_i** file backs out any procedures done with the *Product_Option*.**pre_i** file. The default **cleanup/reject** script executes this procedure after files are removed for the present level and before saved files for the previous level are recovered.

2.3.4.2.21 The *Product_Option*.unpre_u File

The *Product_Option*.**unpre_u** file backs out any procedures done with the *Product_Option*.**pre_u** file. The default **cleanup/reject** script executes this procedure after files are removed for the present level and before saved files for the previous level are recovered.

2.4 Format of Distribution Media

The following types of media can be used to distribute software product installation packages.

- tape
- CDROM
- diskette

This section describes the format that must be used to distribute multiple product packages on each of these media.

2.4.1 Tape

In order to stack multiple product package images onto either a single tape or a set of tapes, the files on each tape in the set must conform to the following format.

- File 1 is empty. (Reserved for bootable tapes.)
- File 2 is empty. (Reserved for bootable tapes.)
- File 3 contains a table of contents file that describes all product package images on the entire set of tapes. Therefore, each tape in the set contains a copy of the same table of contents file except for the difference of the tape volume number in a multi-volume set.
- Files 4 through (n+3) contain the backup / restore file images for product packages 1 through n.
- A product package image file cannot span across two tapes.
- Each file is followed by an end-of-file tape mark.

For a description of the table of contents, see The Table of Contents File on page 2−46.

2.4.2 CDROM

A CDROM that is to contain multiple product package images must be in Rockridge format. The directory used for storing product packages is **/usr/sys/inst.images**, and it must contain the following:

- The backup/restore file images of the product packages.
- A table of contents file named **.toc** that describes all the product package images in the directory.

2.4.3 Diskette

In order to stack multiple product package images onto a set of diskettes, the following files must be written to the set of diskettes.

- A table of contents file that describes all product package images to be included in the set.
- Each product package image file that is to be included in the set.

The files are written to the set of diskettes using the following rules:

- The data is written as a stream to the diskette set with a volume ID sector inserted as block 0 of each diskette in the set. The data from the last block of one volume will be treated as logically adjacent to the data from block 1 of the next volume (the volume ID sector will be verified and discarded when reading).
- Each file will begin on a 512-byte block boundary.
- The table of contents file is written first. This file is padded to fill its last sector with null characters (x'00'). At least one null character is required to mark the end of the table of contents file; thus, it is possible that a full sector of null characters could occur.

- Following the table of contents, each of the product package image files is written to successive sectors. Each file is padded to fill its last sector using null characters; no null characters are required if the file ends on the block boundary.
- Block 0 of each diskette in the set contains a volume ID sector. The format of this sector is:

Bytes 0:3 A magic number for identification. This is a binary integer with a value of decimal 3609823513=x'D7298918'.

Bytes 4:15 A date/time stamp, in ASCII, that serves as the identification for the set of diskettes. The format is *mmddhhMMssyy* where
mm is month,
dd is day,
hh is hour (0−23),
MM is minutes,
ss is seconds, and
yy is year.

Bytes 16:19 A binary integer volume number within this set of diskettes. The first diskette in the set is x'00000001'.

Bytes 20:511 Binary zeroes.

2.4.4 The Table of Contents File

Table 2−2 describes the table of contents file. Some fields are different for the different types of media.

Table 2−2. The Table of Contents File.

Field Name	Format	Separator	Description
1. Volume	Character	White space	For the tape and diskette table of contents file, this is the number of the volume containing this data. For a table of contents file on disk or CD, the volume number will be 0.
2. Date / time stamp	*mmddhhMMssyy*	White space	For the tape or diskette this is the time stamp of when volume 1 was created. For disk or CD this is the time stamp of when the **.toc** file was created.
3. Header format	Character	New line	A number to indicate the format of the table of contents file. Valid entries are: 1 — Pre PowerOpen version 2 — Pre PowerOpen version 3 — Version 1 PowerOpen B — **mksysb** tape; invalid for use by **installp**
			Items 4 and 5 below are repeated for each product package image contained on the media.

Field Name	Format	Separator	Description
4. Location of product package image	Character	White space	For the tape or diskette this is a character string in the form *vvv*:*bbbbb*:*sssssss* where these fields are described in detail below. For disk or CDROM this is the filename of the product package image file. This is the filename only and that it must not be preceded by any part of the path name.
5. Package specific information	**lpp_name** file format	New line	The contents of the **lpp_name** file contained in this product package image. See The lpp_name File on page 2–19.

2.4.4.1 Date/Time Stamp Format

The date / time stamp is in the format of *mmddhhMMssyy* where each letter represents a digit and has the following meaning:

mm is the month.
dd is the day.
hh is the hour (0–23).
MM is the minutes.
ss is the seconds.
yy is the year.

2.4.4.2 Location Format for Tape and Diskette

The location is in the format of *vvv*:*bbbbb*:*ssssssss* where each letter represents a digit and has the following meaning:

- for tape

  vvv is the volume number of the tape.
  : is the colon character.
  bbbbb is the file number on the tape of the product package image.
  : is the colon character.
  ssssssss is the size of the file in bytes.

- for diskette

  vvv is the volume number of the diskette.
  : is the colon character.
  bbbbb is the block number on diskette where the product package image file begins.
  : is the colon character.
  ssssssss is the size of the file in bytes (this includes any padding to the end of the block boundary).