HEWLETT
PACKARD

HP 9000
Computers

Creating Product Packages for HP-UX

Creating Product Packages for HP-UX

HEWLETT
PACKARD

HP Part No. B2355-90031
Printed in USA 8/92

E0892

Legal Notices

Printing History

This is the first edition of this manual.

New editions of this manual will incorporate all material updated since this first edition. The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

Typeface Conventions

Unless otherwise noted in the text, this manual uses the following typeface conventions.

Table 0-1. Typeface Conventions

term	**Boldface** text indicates a term that is being introduced for the first time. See the Glossary for the term's definition.
variable_info	*Italic* text in commands or messages represents values you must supply. For example: `mkdir` *directory_name* indicates that you should supply the *directory_name* component of the command.
emphasis	*Italic* text is also sometimes used for emphasis (for example: *never* remove this file ...).
`command`	`Typewriter` text denotes command and file names, examples of source code, and information displayed by the computer.
fpkg(1M)	Refers to a page in the *HP-UX Reference*. This example says that the topic *fpkg* is found in section 1M of the *HP-UX Reference*.
Return	Used for graphic representations of the key caps on a keyboard.
`Help`	Shaded text is used to represent function keys or menu items that appear online.
[ ]	Square brackets enclose optional items in formats and command descriptions.
|	A vertical bar separates items in a list of choices.
. . .	Horizontal ellipsis indicate that you can repeat the preceding item one or more times.
⋮	Vertical ellipsis mean that irrelevant parts of a figure or example have been omitted.

Scope and Organization of This Manual

This manual is for vendors or customers who want to package software in a form that can be used by other HP-UX commands such as *update*(1M), *updist*(1M), *netdistd*(1M), and *rmfn*(1M). Each chapter and appendix addresses a particular aspect of the process for packaging software.

Table 0-2. Manual Organization

Chapter 1: Introduction	Contains a general description of the **fpkg** tool, including supported media types, media format versions, and systems. Also gives a brief overview of the process of packaging software products.
Chapter 2: Making a Package of Software Products	Describes the prerequisites and conditions that must exist when creating a software package, as well as a detailed explanation of all the options available in the **fpkg** command.
Chapter 3: Defining the Structure of the Product Package	Explains how to use the Product Specification File to structure the software product(s) you want to package. It also contains information about how to modify the structure of an existing software package.
Chapter 4: An Example of the Packaging Process	Shows the packaging process, giving examples of the Product Specification File, the log file, the format of the package on the install media, and the files created on the destination host.
Appendix A: Guidelines for Installation Control Scripts	Contains guidelines for writing and testing fileset customization and check scripts.
Appendix B: Re-Creating a Product Specification File	Contains detailed information about how to re-create a Product Specification File.
Glossary	Defines the critical terms used in the packaging process.

Related Documentation

In addition to this manual, you may find the following documentation useful:

- *HP-UX Reference* manual.
- The online manpages of the following related HP-UX commands:

fpkg(1M)	Command that packages a set of source files (a software product or application) into a format that is acceptable to the `update` command.
update(1M)	Command that installs or updates software from a host system to a destination system.
netdistd(1M)	Command that supports the installation of software across the network, from a server host to one or more destination hosts.
rmfn(1M)	Command that removes software products interactively or non-interactively from a system in units of filesets and/or partitions.
updist(1M)	Command that installs or updates the HP-UX system or application files as "fileset packages" in a special directory. This allows the system to be a network file distribution (netdist) server. The network server daemon, `netdistd`, finds the files in this special directory and supplies them to a remote `update` process on request.
update(4)	Describes the format of the update media.
CDFinfo(4)	Describes the CDFinfo file format and rule syntax.

- *Installing and Updating HP-UX*—explains how to install/update software on a local or remote host, remove software from a host, and manage network servers.

Contents

Tables

1

Introduction

The `fpkg` command allows you to package a collection of source files (a software product) into a format that is acceptable to the HP-UX `update` command. The `update` command is a convenient tool that lets you install new products or patches to existing products.

The media produced by `fpkg` can be either tape archive format (see *tar*(1)) or in a format usable by `netdistd` (see *netdistd*(1M)). The `fpkg` command cannot be used to make CD-ROM media.

The `fpkg` command takes information about what is to be packaged from two sources; command line arguments and a file called the **Product Specification File**. These two sources tell `fpkg` what files are to be packaged, where they are to be placed when loaded, what machine series (300/400, 700, or 600/800) are allowed to use these files, and how the source files are organized into logical groups called **filesets** and **partitions**.

The Product Specification File allows the source files to be scattered throughout a developer's file system and yet be pulled together by `fpkg` to be more suitable for loading on another system. The Product Specification File can also be used to specify permissions of each individual file or groups of files. This gives you an easy way to guarantee that the files will be packaged with the correct attributes, and not necessarily with those of the source file.

Supported Media Types

Using `fpkg`, you can package software products for distribution on the following media:

- **Tape Media**, which uses *tar*(1) to store software products and control files needed by `update` to use the media (i.e., all of the product files and the control files reside in a `tar` archive). Such an archive usually resides on a serial media such as a DDS, cartridge, or nine-track tape, though a Tape Media can be a simple, regular file that contains the `tar` archive. All tape devices currently supported by `update` are also supported by `fpkg`.
- **Network Media**, which is used by the *netdistd*(1M) network distribution daemon. Software packages can be added to an existing database, or a new `netdistd` database will be created if it does not already exist. The default netdist database is `/netdist`. This cannot be an NFS mounted directory. If a fileset being added to a network database already exists with the same fileset name, it will first be deleted from the database before adding.

  The `netdistd` daemon cannot handle multiple media format versions in the same database. The `fpkg` command will detect this and refuse to mix media formats.

Supported Media Format Versions

All HP-UX media is stamped with a media format version number, which `update` uses to determine if the media is in a format that can be understood. The `update` tool refuses to read media that is stamped with a media format version higher (newer) than the `update` version. However, `update` can read some media with a version number lower (older) than the `update` version (in other words, it is backward-compatible but not forward-compatible). The current versions of `update` and corresponding supported media format versions appear in Table 1-1.

Table 1-1. Supported Media Format Versions

`update` Release	Supported Media Format Versions
7.0	A.B7.00
8.0	A.B7.00 A.B8.00
8.05 and beyond	A.B7.00 A.B8.00 A.B8.05

Table 1-1 helps determine which media format version to use when creating product packages. The features introduced by each media format version are summarized below:

A.B7.00	Provides basic functionality, usable by a large number of HP-UX releases.
A.B8.00	Allows for fileset versioning, which means that a fileset can depend on a particular version of a different fileset. This allows `update` to determine if a depended-on fileset needs to be (re)loaded or if the version on the system is sufficient.
A.B8.05	This is the first version that fully supports the Series 700. In earlier versions, media must be marked for the Series 800, which implies Series 700 also. This version is like the A.B8.00 version but has two new fileset keywords (`sys` and `is`) that allow for greater Series and instruction set specification.

Supported Systems

The `fpkg` tool supports all HP-UX architectures currently supported by `update`. This includes the Series 300/400, 700, and 600/800, depending on the media format version (see "Supported Media Format Versions").

When packaging a product with `fpkg`, you have the option of specifying which machine series the product is to be loaded on.

- Media format versions A.B7.00 and A.B8.00 allow only one series to be specified (or none, which implies that all series can load it).
- With media format version A.B8.05, the keywords `sys` and `is` in the Product Specification File allow media to be made for any combination of series (although all filesets must agree on these keywords).

The `-S` option on the command line can help in setting the correct architecture flags.

The `fpkg` command does not need to run on the same series or release as the type of media that is being produced. This means that if your source/build machine is a Series 800, you can still make media to be loaded on a Series 300. Also, if your source/build machine is running 9.0 HP-UX, you can still make media loadable by a machine running 7.0 (by setting the `-V` *media-format-version* option).

An Overview of the Packaging Process

The `fpkg` command is all you need to create a software package that you can then install or update onto other destination hosts. In a nutshell, the process of creating a software package consists of the following steps:

1. Satisfy the necessary prerequisites and conditions before running the `fpkg` command (described in *Chapter 2: Making a Package of Software Products*).
2. Decide which options of the `fpkg` command are appropriate for your package (described in *Chapter 2: Making a Package of Software Products*).
3. Define the structure of your package using the Product Specification File (described in *Chapter 3: Defining the Structure of the Product Package*).
4. Create the package using the `fpkg` command, using the information gathered from steps 2 and 3 above. Once invoked, the `fpkg` command does the following:
   a. The `fpkg` command first parses the Product Specification File, flagging all errors and warnings it finds.
   b. If errors are found, `fpkg` exits, having listed these errors to `stderr` and the log file (if open).
   c. If no errors (or only warnings) are found, `fpkg` builds the media. Any warnings are listed to `stderr` and the log file (if open).

It is important to keep in mind that the files that are being packaged will reside in three different places during the process of packaging and installation:

- First, the files will reside in the original place(s) specified in the Product Specification File. These files can be scattered throughout the developer's file system.
- The `fpkg` command interprets the Product Specification File, translates file path names as required, then writes them to the specified media using the modified paths.
- Finally, the `update` command extracts the files from the media and loads them into their final destination.

2

Making a Package of Software Products

The `fpkg` command collects the necessary information to build the software package from:

1. The options to the command.
2. The Product Specification File that contains both data attribute and data location information.

This chapter describes the requirements for running the `fpkg` command, as well as the features provided by each of the commands' options.

Prerequisites and Conditions

- Packages must be made on the machine on which `fpkg` is executing.
- The `fpkg` command will not build packages on remote systems.
- You must be superuser to make Network Media using `fpkg`. This is to enable the setting of file permissions and for allowing access to the `netdistd` distribution tree. The `fpkg` command does not touch the permissions on the source files, but still needs to be able to set the appropriate permissions on the files it creates. Making Tape Media does not require superuser privileges unless the source files cannot be otherwise accessed.
- Normally, a media package cannot mix filesets destined for different architectures (machines series). This means that if one fileset is marked to be loaded on one specific series (Series 300 for example), all other filesets on that media must also be for that series. However, you can make filesets loadable by different series as long as all the filesets on the media are consistent.

The HP-UX 9.0 version of `update` allows a limited amount of architecture mixing of filesets. If you want to mix architectures and the package will only be installed on HP-UX 9.0, the `-M` option can be used to allow packaging of mixed architecture filesets.

- The media created by `fpkg` must have a version associated with it which correlates to the version of `update` (or `updist`) that is intended to read it. This is because throughout the history of the `update` command, certain enhancements were made to the media format that made it no longer compatible with the previous version. A media package can only contain one version of `update` media. It is important to know what version of `update` will be reading the media being created and use `fpkg` to create the appropriate media format version (see Table 1-1.
- When making `tape` media, the `fpkg` command tries to determine the capacity of the device. However, `fpkg` cannot do this for all devices, so the size of the device may have to be specified on the command line (using the `-s` *device-size* option). This information helps `fpkg` determine if the product will fit on the device, and how to arrange filesets on multi-volume packages.
- No interrupts are allowed if `network` media is being built, but if `tape` media is being built, `fpkg` can be interrupted. When an interrupt occurs, `fpkg` removes the work done so far, then restores the system to its original state.
- The `fpkg` command supports the following product file types: regular files, directories, symbolic links, and hard links. If a recognized but unsupported type or an unrecognized type is given, an error message is given.

  In creating hard links, `fpkg` assumes that the first occurrence of the file in the `Files` list for this fileset is the "primary" to which all other occurrences of the inode (operating system structure that contains file information like number of links) are linked.
- Attributes of files being packaged are preserved as closely as possible:
  - □ File permissions may be overridden by the Product Specification File.
  - □ Hard links to files are preserved as long as both file elements of the file are included in the package.
  - □ Symbolic links are preserved (or not, depending on the `-h` option), even if the target file is not part of the package. The `update` tool requires that all symbolic links be relative to root (`/`), which means that "dot-relative" links such as "`foo -> ../foo`" are not acceptable to `update`. The `fpkg`

tool eases this restriction by converting any "dot-relative" symlinks to be relative to root (`/`). If any such conversions are made, `fpkg` issues a notice.

- Files created by `fpkg` are organized into filesets, and `fpkg` requires that these filesets be given a name. This name must be unique with respect to any other fileset that may be loaded onto the system. Since there is no way to know what other filesets may exist now and in the future, it is a good idea to create fileset names that have a low probability of being used by another `fpkg` user (or used on the base HP-UX system). Using a unique acronym (like your company's initials) as a prefix is a good example.
- Typically, only one invocation of `fpkg` is allowed at a time. If `fpkg` detects that another `fpkg` process is running and using the same resources, it will exit with a message stating this fact. To run more than one `fpkg` process at a time, do this for each invocation of `fpkg`:
  - Specify a different log file (with the `-L` *logfile* option) for each invocation.
  - If creating `network` media, specify a different netdist directory (with the `-d` *directory* option) for each invocation.
  - If creating `tape` media, specify a different tape device (with the `-a` *archive-file* option) for each invocation.
- When the destination directories are created, their attributes will be made to reflect the source, if possible. If there is no corresponding source directory and the directory permissions were not set in the Product Specification File, they will be set to these defaults:

mode	0755
owner	bin
group	bin

Options Available With the fpkg Command

For creating new packages, the `fpkg` command has the following syntax:

```
fpkg [-m media-type] [-d destination-directory] [-a archive-file]
[-s device-size] [-V media-format-version] [-S machine-series]
[-L logfile] [-h] [-v] [-c comment-string] [-M]  Product-Specification-File
```

For re-creating media (creating a Product Specification File from a CD-ROM or `netdist`, the syntax of the `fpkg` command is this:

```
fpkg [-v] [-L logfile] -r media-directory > Product-Specification-File
```

Here is a brief explanation of each of the `fpkg` options. More detailed explanations of the options appear after this table.

Table 2-1. Options of the fpkg Command

`-?`	Displays a general usage message.
`-m` *media-type*	Defines the type of product media to create (either `network` or `tape`). The default *media-type* is `network`.
`-d` *directory*	If creating `network` media, this option defines the destination directory where the media will be located. The default *directory* is `/netdist`.
`-a` *archive-file*	If creating `tape` media, this option names the archive file on which to write the `tar` archive. If the file does not exist, `fpkg` will create it as a regular disk file. The default *archive-file* is `/dev/rmt/0m`.
`-s` *device-size*	If creating `tape` media, this option specifies the size of the device, in megabytes. Otherwise, `fpkg` sets the size to: Cartridge tape — 63 MBytes 9-track tape — 40 MBytes DDS-format tape — 1330 MBytes Disk file — Size of free file system space

Table 2-1. Options of the fpkg Command (continued)

`-V` *media-format-version*	Specifies the media format version number for the product(s). The media format version number is used by **`update`** to compare itself against the media, to insure that it supports the format created by **`fpkg`**. The default *media-format-version* is **`A.B8.00`**.
`-S` *machine-series*	Specifies which series of machines will be able to read the media produced. The default *machine-series* for **`tape`** media is "all series". For **`network`** media, the default is the machine series on which **`fpkg`** is executed. You can give multiple **`-S`** options if the media format version supports it.
`-L` *logfile*	Writes log information to *logfile* instead of the default log file **`/tmp/fpkg.log`**.
`-h`	Follows symbolic links and treat them as regular files. Without this option, **`fpkg`** makes a literal copy of a symbolic link.
`-v`	Turns on verbose output.
`-c` *comment-string*	Overrides the default comment string placed in the **`MAIN.pkg`** file used by **`netdistd`**.
`-M`	Lets **`fpkg`** produce media that contains filesets destined for a mixture of architectures (HP-UX 9.0 systems or later).
`-r` *media-directory*	Creates a Product Specification File from a physical media (CD-ROM or **`netdist`**).

Defining the Type of Media to Create (`-m` *media-type*)

The `-m` *media-type* option defines the type of product media to create. The recognized media types are:

`network`	Builds the software package for distribution on a network server (via the `netdistd` daemon). The package is created in the format used by `netdistd` and then loaded into the destination directory (`/netdist` or whatever is set by the `-d` option).
`tape`	Builds the software package as a single `tar` archive so that it can be put on a DDS, cartridge, or nine-track tape. The package is written to the specified *archive-file* (`/dev/rmt/0m` or whatever is set by the `-a` option) in a `tar` format directly suitable for use by `update`.

If the media type named is not one of the above, an error message is given.

The default *media-type* is `network` media.

Naming the Network Media Destination (`-d` *directory*)

If creating `network` media, the `-d` *directory* option defines the destination directory where the media will be located. This directory is also referred to as the **Network Media**.

Specifying the `-d` option implies the `-m network` option, meaning that if you use the `-d` option, you do not have to use the `-m` option, since `fpkg` will assume that the media type is `network`.

The directory given must be an absolute pathname, not equal to `/`. Otherwise an error message is given.

The default destination directory is `/netdist`.

Naming the Tape Media Device (`-a` *archive-file*)

If creating `tape` media, the `-a` *archive-file* option lets you specify the output device file (or regular disk file) to which `fpkg` writes the package archive.

You can also use a dash (`-`) in place of the *archive-file* (i.e. `-a -`) to cause `fpkg` to write to `stdout`. This allows, among other things, the output to be piped to a tape device on a remote host.

Specifying the `-a` option implies the `-m tape` option, meaning that if you use the `-a` option, you do not have to use the `-m` option, since `fpkg` will assume that the media type is `tape`.

The package is written in `tar` format, meaning that all operations valid for working on a tar-archive are valid for the *archive-file*. One common operation is to create the *archive-file* as a regular disk file before transferring it to a tape device. The transfer can be done by using `dd` with a block size (`bs`) of 10K.

To determine the contents of the *archive-file* created by `fpkg`, you can view the *archive-file* using the following command:

```
tar -tvf archive-file
```

If the archive file does not exist, `fpkg` will create it as a regular disk file. If the archive file does not exist and cannot be created, or if it is not one of the supported serial types, an error message is given.

The default *archive-file* is `/dev/rmt/0m`.

Specifying the Tape Device Size (`-s` *device-size*)

If creating `tape` media, the `-s` *device-size* option lets you specify the size of the tape media (*archive-file*) in megabytes. This size information is used to determine how much of the package will fit on one tape. This is necessary information, especially when the package will span more than one volume. For some tape devices, `fpkg` can automatically determine the capacity.

This option is required if the tape media is anything other than a DDS-format tape or a disk file. If the option is not included when required, an error message is given.

The *device-size* has to be greater than zero. Otherwise, an error message is given.

The devices recognized by `fpkg` and the default *device-size* for each are:

Cartridge tape	63 MBytes
9-track tape	40 MBytes
DDS-format tape	1330 MBytes
Disk file	Size of free file system space

Information about default values get appended to the file `/tmp/fpkg.log` (or the log file set by the `-L` option). If the `-v` (verbose) option is used, the default values also appear on the screen.

Specifying the Media Format Version (`-V` *media-format-version*)

The `-V` *media-format-version* option specifies the version number of the media produced by `fpkg`. The *media-format-version* number determines which versions of `update` will be able to read the media. This allows `fpkg` to support multiple versions of `update`. In general, `update` can read media older than the time of its release, but not newer. For more details, see "Supported Media Format Versions" in Chapter 1.

Acceptable values for *media-format-version* are `A.B7.00`, `A.B8.00`, and `A.B8.05`.

If *media-format-version* is not one of the acceptable values, `fpkg` will round the given number to the next lowest acceptable value (for example `A.B8.01` -> `A.B8.00`).

The default value for *media-format-version* is `A.B8.00`.

Determining Machine Series To Use Media (`-S` *machine-series*)

The `-S` *machine-series* option lets you specify which series of machines will be able to read the media produced by `fpkg`. Acceptable values are `300`, `700`, and `800`.

Certain characteristics of this option differ depending on whether you are creating Tape Media (`-m tape`) or Network Media (`-m network`).

Tape Media

- You can allow all series to read the media by omitting the `-S` option and not specifying any architecture series specifiers in the Product Specification File.
- For tape media of version A.B8.05, you can specify a mixture of machines that can load this media by using multiple `-S` options (e.g., `-S 800 -S700`).
- Default *machine-series* is "all series". This default value will be overridden if the Product Specification File contains any machine architecture specifiers (`fpkg` issues a notice in this case).

Network Media

- The *machine-series* information is necessary because of the structure of the `netdistd` database, which keeps a separate `netdistd` subdirectory for each machine series. Thus, `fpkg` needs to know in which subdirectory to place the package.
- For network media of version A.B8.05, `fpkg` uses the first `-S` option to determine where to place the package. Therefore it must be called multiple times rearranging the `-S` options so that the package can be placed in each of the appropriate subdirectories.
- The default value for *machine-series* is whatever machine series the `fpkg` command is executed from. This default value will be overridden if the Product Specification File contains any machine architecture specifiers (`fpkg` issues a notice in this case).

Naming an Alternate Log File (`-L` *logfile*)

The `-L` *logfile* option lets you choose an alternate name for the log file. The `fpkg` tool appends a log of messages, errors, and other information to this file.

The *logfile* name must be an absolute pathname, not equal to `/`. Otherwise, an error message is given.

The default *logfile* name is `/tmp/fpkg.log`.

Treating Symbolic Links (`-h`)

The `-h` option tells `fpkg` to ignore files that are symbolic links, and to treat the linked-to-file as the file to be placed into the package instead of the link. Without this option, `fpkg` makes a literal copy of a symbolic link, which is then restored by `update` when the media is loaded. This option should not be set if you intend to ship symbolic links.

Turning On Verbose Output (`-v`)

The `-v` option turns on verbose output. This can be useful for determining what defaults were chosen for the package, and for a step-by-step progress report. Without this option set, `fpkg` issues some status information, notices, and errors. A log of more detailed information is appended to the file `/tmp/fpkg.log` (or the log file set by the `-L` option).

Changing the Comment String in MAIN.pkg (`-c` *comment-string*)

The `-c` option lets you override the default comment string that is placed in the `MAIN.pkg` file used by `netdistd`. The default string is: "Fileset packages for use by update(1m)".

Mixing Architectures (`-M`)

The `-M` option allows `fpkg` to produce media that contains filesets destined for a mixture of architectures. However, until HP-UX release 9.0, the `update` command will refuse to load media that contains filesets with mixed architecture specifiers. Using the `-M` option may cause the media to be loadable only by HP-UX release 9.0 or later systems. A warning will be given if this is the case.

Re-Creating Media (`-r` *media-directory*)

The `-r` option lets you transfer filesets from either CD-ROM or `netdistd` media to tape media. With this option, `fpkg` reads the media specified by *media-directory* and writes (to standard out) a Product Specification File that can be used in a second invocation of `fpkg` to re-create the desired media. The argument *media-directory* is the pathname of a mounted CD-ROM (e.g. `/UPDATE_CDROM`) or that of the architecture level of a `netdistd` directory (e.g. `/netdist/300` or `/netdist/800`).

Note The `-r` *media-directory* option is not intended as a replacement for `updist`, which should still be used when transferring media to a `netdistd` database.

The `fpkg` command will skip any filesets on CD-ROM media that are secured (encrypted). If the verbose output (`-v`) option is given, `fpkg` notifies you each time it skips a secured fileset. To transfer secured filesets to the tape media, do this:

1. Use `updist` to transfer the fileset(s) to a `netdistd` directory (they are decrypted by the `updist` process).
2. Invoke `fpkg` with the `-r` option naming the `netdistd` directory just created by `updist`. This creates a new Product Specification File.

For more details, see *Appendix B: Re-Creating a Product Specification File.*

Some Examples of Command Lines

Here are some example command line usages of **fpkg** (and some other tools). These examples assume that the Product Specification File has already been created, and that it does not contain any architecture specifiers (**sys**, **is**, **ff H**, or **ff M** keywords).

- To make tape media for a Series 800 machine (default A.B8.00 media format version), use this series of commands:

`fpkg -m tape -a update.image -S 800 psf.file`	*Write the image to a disk file*
`tar -tvf update.image`	*Use* **tar** *to look at the contents of the file*
`/etc/update -cs $PWD/update.image -S800`	*Use* **update** *to look at the contents of the file*
`dd if=update.image of=/dev/rmt/0m bs=10k`	*Use* **dd** *to transfer the image to a DDS tape device*
`cat update.image | tcio -o -z -v -V -S8 /dev/update.src`	*Write the same image to a cartridge tape. The* **-z** *option is required.*

- To make network media with a version of A.B7.00 for a Series 300, under a non-default netdist tree, use these commands:

`fpkg -S300 -VA.B7.00 -d /netdist/7.0 psf.file`	
`/etc/netdistd -f /netdistd/7.0/MAIN.pkg`	*Start the* **netdistd** *daemon on default port (2106)*

- To make tape media for all architectures, use this command:

  ```
  fpkg -m tape -a update.image psf.file
  ```

- To make a tape using a device on a remote host, use this command (size must be specified):

  ```
  fpkg -a - -S300 -s1330 -v /tmp/psf | remsh host dd obs=10k of=/dev/rmt/0m
  ```

- To make DDS-format tape media for Series 700 and Series 800 of 1300 MBytes (1.3 GBytes) capacity, (only `A.B8.05` media format version supports this), use this command:

  ```
  fpkg -m tape -a /dev/rmt/0m -s1330 -S700 -S800 -VA.B8.05 psf.file
  ```

3

Defining the Structure of the Product Package

This chapter describes the structure of HP-UX update media and how to use the Product Specification File to define the structure of your software package.

General Structure of HP-UX Update Media

HP-UX update media appears as a three layer hierarchy:

1. The top layer contains **partitions**. Partitions are a named collection of related filesets.
2. Under each partition is a collection of related **filesets**. A **fileset** is a named collection (grouping) of directories and files that divide a partition into manageable units. Each fileset belongs to exactly one partition.
3. The actual directories and files make up the third layer.

The `update` tool allows you to pick and load functional groups of files either at the partition level or the fileset level. The fileset is the smallest group of files that `update` will load.

The Product Specification File defines partitions, the filesets in partitions, and the files in each fileset. There is no limit to the number of partitions, filesets, and files that you can define in a Product Specification File.

Creating the Product Specification File

The **Product Specification File** is the mechanism for defining the structure for the products being packaged.

There are attributes associated with each level of the product structure, and each attribute has a **keyword** that tells `fpkg` about that attribute of the product. The same keywords for attributes are used for both supported media types (`network` and `tape`).

In general, the structure of the Product Specification File looks like this:

```
Partition-name-and-description
    Fileset-name-and-description
    Fileset-attributes
        Files-in-fileset
    Another-fileset
    Attributes
        Files-in-this-fileset

Next-Partition-name-and-description
    Next-Fileset
    Attributes
        Files
...
```

The Product Specification File is made up of a list of keywords usually followed by an argument. The syntax for a keyword entry is:

keyword (or short notation) argument [# *comments*]

Most attributes are optional and are not required entries in the Product Specification File. Each attribute is specified by either its full keyword or a short notation. For example, a partition name called can be specified by either of the following entries:

```
partition_name  ALLBASE
pn  ALLBASE
```

Table 3-1. Keywords Used in the Product Specification File

Full Keyword	Short Notation	Argument(s)	Use
`partition_name`	`pn`	string	*recommended*
`partition_description`	`pd`	string	*recommended*
`fileset_name`	`fn`	string	*required*
`fileset_description`	`fd`	string	*recommended*
`fileset_flags`	`ff`	characters	*optional*
`instruction_set`	`is`	instruction id	*not recommended*
`system_architecture_type`	`sys`	Series list	*not recommended*
`fileset_dependency`	`dep`	string(s)	*optional*
`fileset_version`	`fv`	version string	*optional*
`fileset_file_permission`	`ffperm`	*owner group mode*	*optional*
`fileset_directory_permission`	`fdperm`	*owner group mode*	*optional*
`customize`		file name	*optional*
`decustomize`		file name	*optional*
`copyright`		file name	*optional*
`CDFinfo`		file name	*optional*
`systemfile`		file name	*optional*
`media_order`		number	*optional*
`media_format`		format version	*optional*
`pseudo_root`	`pr`	path=path	*optional*
`Files`	`F`	* or none	*required*

Prerequisites and Conditions

- The files contained in a fileset are listed one per line, the list is terminated by any recognized keyword. This means that no file name can exactly match that of any keyword. To work around file name conflicts, use the full pathname of the file, or prefix the path `./` to the filename.
- All keywords except the `Files` (`F`) keyword have arguments. If the argument is missing, an error message is given.
- Some keywords have an argument that is a character string value. Strings containing embedded white space (e.g. description fields) do not need to be enclosed in quotes, unless the comment character (#) is to be used in the string.
- Comments (designated by a preceding #) can be placed on a line by themselves or after the *keyword argument* syntax.
- Most attribute keywords should be used only once for each partition or fileset group. For example, in a given fileset, there can be only one instance of `fileset_name` and `fileset_description`. The exceptions to this rule are four of the keywords dealing with fileset attributes (`fileset_dependency`, `fileset_file_permission`, `fileset_directory_permission`, and `systemfile`) and the two file location keywords (`pseudo_root` and `Files`).

  A warning message is given for all duplicate keywords given after the first.

The following sections describe each keyword (the short notation appears in parentheses after the full keyword).

Defining Partition Attributes

Two keywords describe the attributes of the partition(s) being packaged. Because they are defined at the partition level, these two keywords apply to all the levels defined for that partition (i.e., filesets and/or files).

partition_name (pn)

The keyword `pn` establishes its argument as the partition name, specifying the partition to which any following filesets will belong. The argument for `pn` can be up to 14 characters long. It is an optional (but recommended) keyword. This keyword must precede any `fileset_name` (`fn`) keyword. It is usually the first keyword to appear in the Product Specification File. Each time `fpkg` finds a new `pn` keyword, a new partition is started, and the filesets that follow are placed in that partition.

If no partition name is given, the default will be `UNKNOWN` and `fpkg` will give a warning.

Any character usable in a directory name is allowed as part of the partition name. The following characters are not allowed:

```
. * / ? " [
```

Embedded white space is also not allowed. If an illegal character (or embedded white space) is used in the partition name, an error message is given.

An example of a `partition_name` entry is:

```
pn  ALLBASE
```

Note The partition name has no physical representation on the media (unlike the fileset name). It only appears in the control files on the media.

---

partition_description (pd)

The keyword `pd` lets you attach a descriptive text field up to 32 characters long to the partition name that was given with the 'pn' keyword that preceded this keyword in the Product Specification File. This description can be helpful when making installation selections, since it is visible when running `update` to load the media interactively.

The description may contain spaces and may be surrounded by double-quotes ("). If no description is provided, `fpkg` will use `"no description given"`.

An example of a `partition_description` entry is:

```
pd Database Products
```

Defining Fileset Attributes

The keywords in this section apply to the fileset(s) in a partition.

fileset_name (fn)

The keyword `fn` establishes its argument as the fileset name. It is a required field in the Product Specification File. The other fileset attribute keywords, all optional, provide additional information for the fileset named with `fn`. Each time `fpkg` finds a new `fn` keyword, a new fileset is started.

The argument for `fn` can be up to 14 characters long.

For every other fileset keyword that is used, `fpkg` checks for the existence of a fileset name. If not found, an error message is given.

A fileset without a partition causes the partition to be set to `UNKNOWN`, and a warning is given.

All printable characters are allowed except for the following:

```
. * / ? " [
```

Embedded white space is also not allowed. If an illegal character (or embedded white space) is used in the fileset name, an error message is given.

Note that as mentioned in "Prerequisites and Conditions" in Chapter 2, fileset names must be unique with respect to any other filesets that may be loaded onto the system. This is because the fileset name is used as a directory and file name in the `update` database. Having a conflicting fileset name will cause problems and confusion. For examples of fileset names already in use, look in the `/etc/filesets` directory or `/system` directory on any HP-UX system. Since there is no way of knowing what fileset names are currently in use or what will be used in the future, choosing a unique and meaningful name is a challenge. One recommendation is to use your company's initials as a prefix of the fileset name.

An example of a `fileset_name` entry is:

```
fn  ALLBASE1
```

fileset_description (fd)

The keyword `fd` lets you attach a descriptive text field up to 32 characters long to the fileset name that was given with the `fn` keyword that preceded this keyword in the Product Specification File. This description can be helpful when making installation selections, since it is visible when running `update` to load the media interactively.

The description may contain spaces and may be surrounded by double-quotes ("). If no description is provided, the fileset will have a blank description.

An example of a `fileset_description` entry is:

```
fd  ALLBASE Run-time System
```

fileset_flags (ff)

The keyword `ff` allows you to assign special conditions to a fileset. The argument for `ff` is a list of up to 7 characters, each with a special meaning (order is not important). No embedded spaces are allowed.

The possible flags are:

`B` The "reboot" or **Rebuild and Reboot Fileset Flag** causes `update` to rebuild the kernel and reboot after the fileset is loaded and its `customize` script is run. All filesets marked with a `B` flag will be loaded and customize scripts executed before the kernel is rebuilt.

`C` The "no change destination" or **Nonlocatable Fileset Flag** states that the fileset cannot be installed into any destination directory other than root (/).

`Y` Indicates to `update` that it should run the `sysrm` or `rmfn` command to remove any old fileset by this name prior to loading. This can slow the update process considerably and is not normally done. It is best to remove specific unwanted files in the `customize` script.

`D` Specifies that the fileset's `customize` script should run only after all filesets selected are loaded (as opposed to running after each fileset is loaded). This is the default action for filesets loaded with a version of `update` that is version 8.05 or later. Consequently, this flag is obsolete (but can still be used) for loading on 8.05 or later systems. This flag is not compatible with the `B` flag.

`H`	For A.B7.00 and A.B8.00 media format versions, this flag is used to indicate that this fileset is loadable only onto PA-RISC architecture machines, namely Series 700 and 800. Use of this flag is NOT recommended (see Note below).
`M`	Similar to the `H` flag, but is used to indicate MC-680x0 architecture machines, namely the Series 300 and 400. Use of this flag is NOT recommended (see Note below).

Note The `H` and `M` flags are used on A.B7.00 and A.B8.00 media format versions. For A.B8.05 media, use the `sys` and `is` keywords instead. The `H` and `M` flags are used to specify the type of machines that can load a fileset. These flags may be left off to indicate that the fileset is loadable by all series machines. Or they may be left off, and later specified by using the `-S` *machine-series* command line option, in which case `fpkg` will automatically supply these flags. (That is if the media format version is A.B7.00 or A.B8.05. If the media format version is A.B8.05, it will use the appropriate `sys` and `is` keywords.) All filesets on the media must have the same architecture-specific flags. It is recommended that you NOT use the `H` and `M` flag, but instead use the `-S` *machine-series* command line option (and let `fpkg` supply the appropriate flags).

`S`	Used only if this media is later transferred to a CD-ROM through HP's internal integration and manufacturing process. When this flag is set, the fileset is encrypted upon transfer to the CD. When encrypted, the fileset cannot be loaded without first obtaining a codeword (password). Note that `fpkg` cannot make CD-ROM media.

The fileset flags given in the Product Specification File are checked against the list above. If a flag does not match one in the list, an error message is given.

An example of a `fileset_flags` entry is:

```
ff  BC
```

instruction_set (is)

The keyword `is` specifies the instruction set of the systems that are allowed to load a fileset. This keyword is only valid with `A.B8.05` version media.

The argument string for this keyword can contain up to 11 characters. Valid instruction set identifiers are:

`MC68020`	for Series 300 and 400 machines
`PA_RISC_1_0`	for Series 700 and 800 machines
`PA_RISC_1_1`	for Series 700 machines only
*	indicates that any instruction set machine may load this fileset

If this keyword is used, the `sys` keyword must also be specified. This keyword cannot be used in conjunction with the `H` or `M` flag to the `fileset_flags` (`ff`) keyword.

It is recommended that you NOT use the `sys` or `is` keywords, but instead use the command line option `-S` *machine-series* (which allows `fpkg` to automatically generate the `is` keyword, if appropriate).

An example of an `instruction_set` entry is:

```
is PA_RISC_1_0
```

system_architecture_type (sys)

The keyword `sys` describes the machines or architectures on which this software will execute. This keyword is only valid with `A.B8.05` version media.

The argument string for this keyword can be up to 27 characters long. Valid system types are:

- `S300`
- `S400` (translated to be `S300`).
- `S600` (translated to be `S800`).
- `S700`
- `S800`
- `*` (translated as "any series machine")

To specify more than one system type, separate them by a comma.

If this keyword is used, the `is` keyword must also be specified. This keyword cannot be used in conjunction with the `H` or `M` flag to the **`fileset_flags`** (**`ff`**) keyword.

It is recommended that you NOT use the **`sys`** or **`is`** keywords, but instead use the command line option `-S` *machine-series* (which allows **`fpkg`** to automatically generate the **`sys`** keyword, if appropriate).

An example of a `system_architecture_type` entry is:

```
    sys S700,S800
```

fileset_dependency (dep)

The keyword `dep` allows you to specify any fileset (and fileset version) that must be loaded before, or along with this fileset for the product to function properly.

There are two argument strings associated with this keyword:

1. The first string is the name of one dependent fileset. It can be up to 14 characters long.
2. The second (optional) string is the version number of the dependent fileset. It can be up to 11 characters long. During an installation, selection of a fileset with dependencies causes the automatic selection of the dependent filesets if they are not already present on the destination host with a version number equal to or greater than that required. This version feature is not supported when making `A.B7.00` media (`fpkg` will give a warning in this case). See section "fileset"version (fv)" for more details.

Most fileset attribute keywords should be used only once for each fileset. However, this keyword can be used more than once for the same fileset (when a fileset has more than one dependent fileset).

The `fpkg` tool will give a warning if the depended on fileset is not contained in the same package (this is because `update` cannot enforce this dependency if it is not on the same media, it can only give a warning during loading).

An example of a `fileset_dependency` entry is:

```
dep  ALLBASE-MAN A.B8.05A
```

fileset_version (fv)

The keyword `fv` sets the version string for this fileset. The version string is used by `update` in calculating fileset dependencies (refer to the `dep` keyword). Giving a fileset a "version" allows other filesets to depend on a particular version of this fileset. For example, if this fileset is loaded onto a system, and it has a fileset version of `A.B8.07.0A`, and later another fileset is loaded that has a dependency on version `A.B8.05.0A` of this fileset (as in the example above), `update` will proceed with the load because it knows that the system holds a fileset equal or greater than the version required.

The concept of giving a fileset a version number was introduced at HP-UX release 8.0, so if you are making media for HP-UX 7.0, the fileset version will be ignored, and `fpkg` will give a warning (if the media format version set by the `-V` option is `A.B7.00`). Giving a fileset version `A.B7.00` (the default) indicates to `update` that it should not use the version number in its calculations, and it will always reload the fileset if another selected fileset depends on it.

Both `update` and `fpkg` require that the fileset version be at least `A.B7.00`, thus a version of `A.B6.5` will be rejected.

When a fileset is being loaded, either as a primary selection or as a dependency, its version number is checked against the version of the fileset on the destination (if the fileset already exists on the destination).

- The fileset is updated if its version is greater (newer) than the version on the destination host.
- If the version on the media is the same as the one on the destination host, the software is installed if and only if the fileset was manually selected for loading.
- If the software is less (older) than the version on the destination host, the interactive version of `update` will just give a warning and allow it. however, the command line version of `update` will give an error, so the version on the destination host must be removed first (manually by the user).

The argument string for this keyword can be up to 11 characters long, and the syntax of the argument string is a sequence of dot-separated letters and digits. When `update` compares two version strings, it compares each corresponding sub-string between the dots. So a version of `B6` is greater than `A.B7.00`. Version strings are truncated at 11 characters.

The default value for `fileset_version` is `A.B7.00`.

An example of a `fileset_version` entry is:

```
fv  A.B7.00
```

fileset_file_permission (ffperm)

By default, a destination file inherits the permissions of the source file. The keyword `ffperm` allows you to override this default by specifying a new *owner*, *group*, and *mode* for all files following this keyword in this fileset.

There are three arguments to this keyword; *owner*, *group*, and *mode*. The arguments *owner* and *group* are given as strings and represent an owner and group name on the destination host. The strings are looked up in the password file on the package-creation machine and the uid/gid (user id/group id) stored. The *mode* argument is expected to be in octal (unless it has a leading `0x` to indicate hexadecimal). You cannot specify the *mode* in decimal form.

This keyword only applies to the fileset in which it is defined. Most fileset attribute keywords should be used only once for each fileset group. However, this keyword can be used more than once when describing a fileset.

This keyword is most useful when a group of files will all have the same permissions. To set the permissions on a per-file basis, or to override the default permissions, the `-o -g -m` file flags may be used (see section "Files (F)" for more details).

Note *owner*, *group*, and *mode* can only be changed for regular files and hard links, not symbolic links. If the permissions are changed on a hard link, they are changed for all other links because they share the inode. If permissions are changed on a symbolic link, they are changed on the source. The permissions are not changed on the link itself.

The syntax for the `ffperm` keyword is:

`ffperm` *owner group mode*

These permissions apply globally in the fileset until a file level override is used or a new `ffperm` keyword overrides them. The three arguments are position dependent, so if any one of the arguments is not wanted, use an asterisk (*) to indicate that no override for that permission should be applied. You can also use the following line to terminate the effects of the previous `ffperm`:

```
ffperm * * *
```

An example of the `fileset_file_permission` keyword is:

```
ffperm  root bin 0644
```

fileset_directory_permission (fdperm)

By default, a destination directory inherits the permissions of the source directory (otherwise the default is `bin bin 0775`). The keyword `fdperm` allows you to override this default by specifying a new owner, group, and mode for all directories following this keyword in this fileset.

There are three arguments to this keyword; *owner*, *group*, and *mode*. The *owner* and *group* arguments are given as strings and represent an owner and group name on the destination host. The strings are looked up in the password file on the package-creation machine and the uid/gid (user id/group id) stored. The *mode* argument is expected to be in octal (unless it has a leading `0x` to indicate hexadecimal). You cannot specify the *mode* in decimal form.

This keyword only applies to the fileset in which it is defined. Most fileset attribute keywords should be used only once for each fileset group. However, this keyword can be used more than once when describing a fileset.

The syntax for the `fdperm` keyword is:

`fdperm` *owner group mode*

These permissions apply globally in the fileset until a file level override is used or new `fdperm` keyword overrides it. The three arguments are position dependent, so if any one of the arguments is not wanted, use an asterisk (*) to indicate that no override for that permission should be applied. You can also use the following line to terminate the effects of the previous `ffperm`:

```
fdperm * * *
```

An example of the `fileset_directory_permission` keyword is:

```
fdperm  root bin 0755
```

customize

The `customize` keyword allows a *customize* script to be placed on the media and associated with the current fileset. This script will be executed after the file set has been successfully loaded. The *customize* script will be executed with the current working directory set to the directory where the fileset is loaded (usually `/`), but you can specify that it be relocated if the `fileset_flags` (`ff`) keyword allows it.

The actual *customize* script will be passed one argument, either `HP-MC68020` for (Series 300/400 machines) or `HP-PA` (for Series 600, 700 and 800 machines) depending on which type of machine the fileset is loaded (this is useful when loading on a mixed architecture cluster). See *Appendix A: Guidelines for Installation Control Scripts* for more details on writing a *customize* script.

The `customize` keyword has an argument that is the pathname for the actual *customize* script, telling `fpkg` where to get the file. The *customize* script provided will be renamed as it is loaded on the media to allow `update` to find and execute it.

If you do not use the `customize` keyword, `fpkg` supplies a nearly empty default *customize* script to overwrite a possible older one left on the system.

A symbolic link may not be given for the pathname for this keyword. If it is, an error message is given.

The syntax of the `customize` keyword is:

`customize` *filename*

The *filename* must include an absolute pathname. The *filename* itself is not important, since it will be renamed `customize` when the package is loaded by `update`. For instance, in the example below, `customize.UX-CORE` will be renamed `customize`.

Here is an example of the use of the `customize` keyword:

```
customize /build/scripts/customize.UX-CORE
```

decustomize

The `decustomize` keyword allows a *decustomize* script to be placed on the media and associated with the current fileset. This script will be executed when the fileset is removed using `rmfn`.

It is important to remember that the *decustomize* script is executed twice. The first time, `rmfn` runs the script just to check if the fileset is removable. The second time, `rmfn` runs the script just prior to removing all files loaded with this fileset. The first invocation of the script is given 2 arguments, the machine architecture (`HP-MC68020` for Series 300/400 machines or `HP-PA` for Series 600, 700 and 800 machines), and the word `check` (meaning don't do anything yet, just checking). The second invocation of the script is given just 1 argument, the machine architecture (`HP-MC68020` or `HP-PA`).

The script should exit with a return code of `0` if no problems are encountered, and with a value `1` if an error occurred. The first invocation of the script is the only chance it has to stop the removal process (by returning a value of `1`). See *Appendix A: Guidelines for Installation Control Scripts* for more details on writing a *decustomize* script.

The `decustomize` keyword has an argument that is the pathname for the actual *decustomize* script, telling `fpkg` where to get the file. The *customize* script provided will be renamed as it is loaded on the media to allow `update` to find and execute it.

A symbolic link may not be given for the pathname for this keyword. If it is, an error message is given.

The syntax of the `decustomize` keyword is:

`decustomize` *filename*

The *filename* must be an absolute pathname. The *filename* itself is not important, since it will be renamed `decustomize`. For instance, in the example below, `decustomize.UX-CORE` will be renamed `decustomize`.

Here is an example of the use of the `decustomize` keyword:

```
decustomize /build/scripts/decustomize.UX-CORE
```

copyright

The `copyright` keyword places a file on the system called:

`/system/`*fileset-name*`/copyright.`

This is where most HP applications place copyright information about the product contained in that fileset.

The syntax of the `copyright` keyword is:

`copyright` *filename*

Here is an example of the use of the `copyright` keyword:

```
copyright /build/rights
```

CDFinfo

The `CDFinfo` keyword allows a CDFinfo file to be placed on the media and associated with the current fileset. The CDFinfo file contains rules that `update` uses when loading the fileset onto a clustered system. These rules specify which files should be loaded as context dependent files (or CDFs). The rules in this file also apply to the *sam*(1M) utility when a system is turned into a cluster server, or when adding a cnode. For more details on creating CDFs, see the *CDFinfo*(4) entry in the *HP-UX Reference* manual.

A CDFinfo file is not necessary if the application will not be supported on a HP-UX cluster system, or if all the files are system independent (i.e. can be shared by all systems in a cluster).

The syntax of the `CDFinfo` keyword is:

`CDFinfo` *filename*

Here is an example of the use of the `CDFinfo` keyword:

```
CDFinfo /build/cdfs/UX-CORE
```

systemfile

The `systemfile` keyword is used if a file needs to be loaded in the `/system/`*fileset* directory but has no specific keyword to place it there (i.e. it is not a `customize`, `decustomize`, `copyright`, or `CDFinfo` file). The file will be loaded under the fileset directory associated with the current fileset, and will be named the same as the basename of the source file.

Do not place files called `index` in this directory, since an `index` file is created by `fpkg` and used by `update` and other utilities. Also, if the filesets are to be loaded into a system running 8.0 HP-UX, the `update` utility will remove the obsoleted files called `revlist`, `pif`, and `customize.old`, so you should avoid using these names for system files.

The syntax of the `systemfile` keyword is:

`systemfile` *filename*

Here is an example of the use of the `systemfile` keyword:

```
systemfile /build/UX-CORE/pdf
```

media_order

The `media_order` keyword is used to control the order in which the filesets are written to the (tape) media. All filesets with a `media_order 1` will be processed first, then those with `media_order 2`, etc.

However, all filesets that are marked with the `fileset_flag B` will be placed on the media first, because `update` loads all those filesets first so that the new kernel can be built. The `media_order` keyword can still be used to order the set of filesets marked with the `B` flag.

Filesets with the same `media_order` number are placed on the media as they appear in the Product Specification File.

The default value for `media_order` is `1`. The maximum value is `10`.

The syntax of the `media_order` keyword is:

```
media_order number
```

Here is an example of the use of the `media_order` keyword:

```
media_order 2
```

media_format

The `media_format` keyword is used to specify the media format version from within the Product Specification File.

The syntax of the `media_format` keyword is:

```
media_format format-version
```

The *format-version* value must agree with the value supplied with the `-V` *media-format-version* command line option.

Here is an example of the use of the `media_format` keyword:

```
media_format A.B8.00
```

Describing the Location of Files

Two keywords describe where the files you want to package into a fileset are located, and where they should be installed. Most attribute keywords should be used only once for each partition or fileset group. However, these two keywords are an exception.

pseudo_root (pr)

The `pseudo_root` keyword specifies a directory where the source files are to be found on the system. In addition, this keyword can also specify a destination directory where those files will be placed when loaded by `update`.

The syntax of the `pseudo_root` keyword is:

`pr` *source-directory* [=*destination-directory*]

Both *source* and *destination* must be absolute pathnames. If these checks fail, an error messages are given.

Here is an example of the use of the `pseudo_root` keyword:

```
pr /users/joe/build
```

The example above will cause `fpkg` to look for the source files in the directory `/users/joe/build`. Any files specified with the `Files` keyword (and not beginning with `/`) will have their path prefixed with the path `/users/joe/build` and included in the current fileset. If the `Files *` keyword is used, all files in the directory `/users/joe/build` will be included in the current fileset.

Another example of how the `pseudo_root` keyword can be used is this:

```
pr /users/joe/build=/usr/bin
```

This example will also cause `fpkg` to look for files in the directory `/users/joe/build`, but the files will have the path `/users/joe/build` replaced with the path `/usr/bin` as it is loaded on the media. This is very useful if the directory that holds the source files is different than where they should be when loaded by `update`. See section "Example of Product Specification File" in Chapter 4 for more ideas on how this can be used.

The fpkg command does not enforce the absolute location of a fileset. All files are placed on the media with relative pathnames. The update command normally loads files relative to the root directory (/) on the destination host's file system, but filesets that do not have the C fileset flag set can be installed to a destination other than root.

Files (F)

The Files (or F) keyword is used to begin specifying the files that are to be included in the current fileset. Each fileset definition MUST include at least one F keyword.

The syntax for the F keyword depends on whether you want to include ALL files and directories under the specified source directory or just specific files and directories.

1. To include ALL files and directories, the syntax for the F keyword is:

   ```
   Files *
   ```

   If the pseudo_root keyword is defined, F * includes all files and directories under this directory in the fileset. Partial wildcarding is not supported, such as F dm* (to indicate all files starting with dm....). If F * is used without the pr keyword, an error message is given.

   Before processing a directory recursively, fpkg changes to the directory given by the pr keyword. Before the chdir is done, the current working directory is saved. It will be restored after directory processing is finished. If either chdir fail, an error message is given.

   When processing the directory recursively, several problems may be encountered. An unreadable or un-statable directory causes an error message.

2. If you do not want to do a recursive directory search, use the `F` keyword followed by an explicit list of files and/or directories to include in the fileset. All following lines that do not match a reserved `fpkg` keyword are assumed to be file names.

   In this case, the syntax for the `F` keyword is:

   ```
   Files
   source  [destination] [-o owner] [-g group] [-m mode]
   ...
   ```

   The field separator is white space or a tab. The list is ended by any keyword or EOF. The *source* pathname is used for *destination* if no mapping (using the `pr` keyword) has been defined and *destination* is not given. If a *source* directory has been defined (using `pr`), then the *source* files can be relative pathnames. Otherwise, full pathnames are required.

   Here are some examples of how files can be specified:

sourcefile	*Specifies a single file.*
sourcefile destination	*Specifies where to get the file and what to name it on the media.*
sourcefile `-o root -m 0755`	*Specify a file, and override permissions.*

   Make sure you indicate the destination directory with the `pr` keyword or give absolute pathnames when specifying the files. If this is not done, an error message is given.

More Information About File Location Keywords

By default, a destination file or directory inherits the permissions of the source file or directory. The keywords `ffperm` or `fdperm` allow you to override this default by specifying a new owner, group, and mode for all files/directories in the fileset. The options (`-o`, `-g`, and `-m`) are used to override either of the above choices and support specifying file or directory permissions at the file level.

When `fpkg` puts together a source name or destination name, it prefixes any directories defined by the `pseudo_root` keyword to it and treats the pathname as a whole.

For instance, given the following syntax:

```
pseudo_root   /users/mode.data/database/1/bin=/database/bin
Files
db1_file1  -o bin  -g bin  -m 0644
```

`db1_file1` has:

source	`/users/mode.data/database/1/bin/db_file1`
destination	`/database/bin/db_file1`

This means that when you set

```
db1_file1 -o bin  -g bin  -m 0644
```

you are asking `fpkg` to set only `db1_file1` with these permissions. You have to be sure that the directories in the path also get their permissions set. To do this, you have three options:

1. Rely on the permissions that the directories have in the source.
2. Use the `fdperm` keyword.
3. Give a line for each directory.

For example, rather than this use of the `F` keyword:

```
pseudo_root   /users/mode.data/database/1/bin=/database/bin
Files
db1_file1  -o bin  -g bin  -m 0644
db1_file2  -o bin  -g bin  -m 0644
1          -o bin  -g bin  -m 0644
11         -o bin  -g bin  -m 0644
```

Adding extra lines (the third and fourth lines in the example below) sets separate permissions for directories and files.

```
pseudo_root   /users/mode.data/database/1/bin=/database/bin
Files
/users/mode.data/database/       /database/     -o root -g other -m 0755
/users/mode.data/database/1/bin/ /database/bin/ -o root -g other -m 0755
db1_file1  -o bin  -g bin  -m 0644
db1_file2  -o bin  -g bin  -m 0644
1          -o bin  -g bin  -m 0644
11         -o bin  -g bin  -m 0644
```

The `pseudo_root` and `Files` keywords can be used more than once in a fileset.

When processing the files in a directory, several problems may be encountered. Inability to open or stat a file found causes an error message.

Examples of the Use of File Location Keywords

The following examples illustrate the use of `pseudo_root` and `Files` keywords.

1. All files under `/mfg/softbench/hp/files` to be rooted under `softbench`:

   ```
   pr /mfg/softbench/hp/files=/softbench
   F *
   ```

2. All files under `/develop/bin`, to be rooted under `usr/bin`:

   ```
   pr /develop/bin/=/usr/bin
   F *
   ```

3. Certain files under `/develop/bin`, to be rooted under `usr/bin`:

   ```
   pr /develop/bin=/usr/bin
   F
   bdf
   more
   vi
   ...
   ```

4. No `pr` keyword given, just name each file explicitly:

   ```
   F
   /develop/bin/bdf          /usr/bin/bdf
   /develop/bin/vi           /usr/local/bin/vi
   /usr/local/bin/find       /bin/find
   ...
   ```

5. No `pr` keyword given, name only the source explicitly:

   ```
   F
   /usr/bin/bdf
   /usr/bin/vi
   /bin/find
   ...
   ```

   These files will have the same destination (e.g. `/usr/bin/vi/` as a source will also have `/usr/bin/vi/` as a destination).

4

An Example of the Packaging Process

This chapter shows an example of the packaging process, which requires the following four steps to complete:

1. Satisfy the necessary prerequisites and conditions before running the `fpkg` command.
2. Decide which options of the `fpkg` command are appropriate to use.
3. Define the structure of the software package using the product specification file.
4. Create the package using the `fpkg` command, using the information gathered from steps 2 and 3 above. Once invoked, the `fpkg` command does the following:
   a. The `fpkg` command first parses the Product Specification File, flagging all errors and warnings it finds.
   b. If errors are found, `fpkg` exits, having listed these errors to `stderr` and the log file (if open).
   c. If no errors (or only warnings) are found, `fpkg` builds the media. Any warnings are listed to `stderr` and the log file (if open).

For this example, we'll briefly go through these steps, and show listings of:

- The Product Specification File.
- The log file produced during the packaging process.

Step 1: Satisfy the Necessary Prerequisites and Conditions

In this example, we will be creating a software package on a Series 300 machine that will be put in a tape image on a regular disk file.

- The package must be made on the machine on which `fpkg` is executing.
- No interrupts of the `fpkg` command will be allowed.

Step 2: Decide Which Options to Use for fpkg

For this example, the following command will be used:

```
fpkg -v -m tape -a /tmp/tape.out -S 300 /tmp/psf
```

The options on the above command line set the following conditions:

`-v`	Verbose output is turned on.
`-m tape`	The type of media that will be created is `tape`.
`-a /tmp/tape.out`	The archive file that the package will be written to. In this case it is a regular disk file. Had it been written to a tape drive, the device file for the tape drive would be named here.
`-S 300`	The package will be read by Series 300 machines only.
`/tmp/psf`	The name of the Product Specification File.

In addition, the following default conditions exist (since the corresponding options were not specified):

- Literal copies will be made of symbolic links (since `-h` option was not used).
- The media format version number for the products created by this command is `A.B8.00`.
- Log information will be written to the log file `/tmp/fpkg.log`.
- The size of the output disk file will be calculated by `fpkg` from the free disk space.

Step 3: Define the Structure of the Software Package

The Product Specification File defines the structure of the software package. In this example, we are making a database package that contains two partitions:

DATABASE	This is the actual database application, which contains two filesets (DBASE-RUN and DBASE-DOC).
DBEXAMPLES	This is a set of database examples, all contained in a single fileset (DBASE-EXAMPLE).

Example of Product Specification File

```
# Product Specification File to package a database application
#
# Start of DATABASE partition information
 pn DATABASE                             # partition name
 pd "The Database"                       # partition description
  # Start of DBASE-RUN fileset information
   fn DBASE-RUN                          # fileset name
   fd "The database application"         # fileset description
   ff C                                  # flag to make update load under '/'
   customize /build/scripts/customize-DBASE     # customize script
   decustomize /build/scripts/decustomize-DBASE # decustomize script
   CDFinfo /build/scripts/CDFinfo-DBASE         # associated CDFinfo file
   copyright /build/misc/rights                 # copyright info file
  #
  # The DBASE-RUN fileset contains everything in /build/dbase/bin on the
  # source machine, and is loaded on the destination system under /usr/bin.
  # These are all executables so set the fileset permissions as such.
  #
   ffperm bin bin 0655                 # set default file permissions
   fdperm bin bin 0555                 # set default directory permissions
   pr /build/dbase/bin=/usr/bin        # specify source/dest dirs
   F *                                 # load all files from directory
  #
  # Now add the support files, setting permissions one by one
   pr /build=/usr                         # specify source/dest dirs
   F                                      # list files separately
   lib -o bin -g bin -m 755               # set directory permissions
   lib/dictionary -o root -g bin -m 0444  # set file permissions
   lib/library -o root -o bin -m 644      # set file permissions
```

```
    #
    # Now add some miscellaneous files in chunks.
      ffperm bin bin 666                      # set default file permissions
      pr /build/misc=/usr/local/misc          # specify source/dest dirs
      F                                       # list files separately
      file1
      file2
      ffperm bin bin 555                      # set new default permissions
      F                                       # list files separately
      file3
      file4
    #
    # Start of DBASE-DOC fileset information
      fn DBASE-DOC                            # fileset name
      fd "Documentation for DBASE"            # fileset description
      copyright /build/misc/rights            # copyright info file
      pr /usr/man/man1                        # same source/destination dirs
      F *                                     # load all files from source

# Start of DBEXAMPLES partition information
  pn DBEXAMPLES                               # partition name
  pd "Database examples"                      # partition description
    # Start of DBASE-EXAMPLE fileset information
      fn DBASE-EXAMPLE                        # fileset name
      fd "Example database's"                 # fileset description
        fdperm bin bin 555                    # specify directory permissions
        pr /build/examples=/usr/local/examples  # specify source/dest dirs
        F                                     # list files separately
        example1 -o bin -g bin -m 644         # set file permissions
        example2 -o bin -b bin -m 555         # set file permissions
```

Step 4: Invoke the fpkg Command

When the `fpkg` command is invoked, the following things occur:

1. The `fpkg` command parses the Product Specification File, flagging all errors and warnings it finds.
2. If errors are found, `fpkg` exits, and the errors are listed to `stderr` and the log file (if it is open).
3. If no errors (or only warnings) are found, `fpkg` builds the media. Any warnings are listed to `stderr` and the log file (if it is open).

In this example, the log file `/tmp/fpkg.log` captures the output from the `fpkg` session. If the message type is serious (error or warning), or if the log file is closed or writing to it fails, the message is written to `stderr`.

Example of Log File

```
=======  04/07/92 16:10:09 MDT  04/07/92  16:10:09 MDT  BEGINNING fpkg PROGRAM
         (command line)

       * The options used for this run are:
          - m (media type)              tape
          - a (archive file)            /tmp/tape.out
          - s (size of output device: MBs)  131
          - V (media format version)    A.B8.00
          - L (logfile)                 /tmp/fpkg.log
          - h (follow symbolic links)   no
          - v (verbose)                 yes
          - S (Series)                  300
         The product specification file is: /tmp/psf

       * Begin parsing the product specification file.
       * Fileset "DBASE-RUN":
          Source location:  "/build/dbase/bin"
          Destination location: "/usr/bin"
       * Fileset "DBASE-RUN":
          Source location:  "/build"
          Destination location: "/usr"
       * Fileset "DBASE-RUN":
          Source location:  "/build/misc"
          Destination location: "/usr/local/misc"
       * Fileset "DBASE-RUN":
          Source location:  "/build/misc"
          Destination location: "/usr/local/misc"
       * Fileset "DBASE-DOC":
           Source location:  "/usr/man/man1"
           Destination location: "/usr/man/man1"
        * Fileset "DBASE-EXAMPLE":
           Source location:  "/build/examples"
           Destination location: "/usr/local/examples"
        * Finished parsing the product specification file.
        * Begin building the software package for  tape media.
        * Total size for control files: 4608 bytes.
        * Fileset: "DBASE-RUN" occupies 16896 bytes on tape.
        * Fileset: "DBASE-DOC" occupies 8704 bytes on tape.
        * Fileset: "DBASE-EXAMPLE" occupies 6656 bytes on tape.
        * Begin building the tar tape.
        * a system/INDEX (mode 0000644) 1 blocks
        * a system/INFO (mode 0000644) 4 blocks
        * a system/CDFinfo (mode 0000644) 1 blocks
        * Begin building  fileset "DBASE-RUN".
        * a DBASE-RUN/../system/DBASE-RUN/customize (mode 0100544) 1 blocks
        * a DBASE-RUN/../system/DBASE-RUN/decustomize (mode 0100544) 1 blocks
        * a DBASE-RUN/../system/DBASE-RUN/CDFinfo (mode 0100444) 1 blocks
        * a DBASE-RUN/../system/DBASE-RUN/copyright (mode 0100444) 2 blocks
        * a DBASE-RUN/../usr/bin/x (mode 0100655) 0 blocks
        * a DBASE-RUN/../usr/bin/y (mode 0100655) 0 blocks
        * a DBASE-RUN/../usr/bin/z (mode 0100655) 0 blocks
        * a DBASE-RUN/../usr/lib/dictionary (mode 0100444) 1 blocks
        * a DBASE-RUN/../usr/lib/library (mode 0100644) 1 blocks
        * a DBASE-RUN/../usr/local/misc/f1 (mode 0100666) 1 blocks
        * a DBASE-RUN/../usr/local/misc/f2 (mode 0100666) 1 blocks
        * a DBASE-RUN/../usr/local/misc/f3 (mode 0100555) 1 blocks
        * a DBASE-RUN/../usr/local/misc/f4 (mode 0100555) 1 blocks
        * Begin building  fileset "DBASE-DOC".
        * a DBASE-DOC/../system/DBASE-DOC/copyright (mode 0100444) 2 blocks
        * a DBASE-DOC/../usr/man/man1/x.1 (mode 0100644) 0 blocks
        * a DBASE-DOC/../usr/man/man1/y.1 (mode 0100644) 0 blocks
        * a DBASE-DOC/../usr/man/man1/z.1 (mode 0100644) 0 blocks
        * a DBASE-DOC/../system/DBASE-DOC/CDFinfo (mode 0100444) 1 blocks
        * a DBASE-DOC/../system/DBASE-DOC/customize (mode 0100544) 1 blocks
        * Begin building  fileset "DBASE-EXAMPLE".
        * a DBASE-EXAMPLE/../usr/local/examples/example1 (mode 0100644) 0 blocks
        * a DBASE-EXAMPLE/../usr/local/examples/example2 (mode 0100555) 0 blocks
        * a DBASE-EXAMPLE/../system/DBASE-EXAMPLE/CDFinfo (mode 0100444) 1
          blocks
        * a DBASE-EXAMPLE/../system/DBASE-EXAMPLE/customize (mode 0100544) 1
          blocks
        * Success building the software package.  Review the log file,
          "/tmp/fpkg.log" for details.

=======   04/07/92 16:10:10 MDT  04/07/92  16:10:10 MDT  COMPLETED fpkg PROGRAM
          (command line)
```

Format of the Package on the Install Media

When the software package is built for distribution on a network server (`network` media type), it is created as **Network Media**, which exists as a tree of directories and files in the file system.

When the software package is built for distribution on **Tape Media** (`tape` media type), files are transferred to the `tar`-formatted archive directly from the source location. Names are translated during packaging.

If you are interested in a more detailed description of the format of the package on the install media, see *update*(4) in the *HP-UX Reference* manual.

A

Guidelines for Installation Control Scripts

This appendix contains guidelines for writing and testing installation control scripts. There are two types of Installation Control Scripts supported by `fpkg`:

1. `customize`.
2. `decustomize`.

The `customize` scripts are run by `update` during the install and update process. The `decustomize` scripts are run during the fileset removal process by the `rmfn` command.

Unless specifically noted, the use of the term **installation control script** applies to both types.

This appendix covers:

- Location and execution of Installation Control Scripts.
- Execution of other commands by the Installation Control Scripts.
- Input and output from the Installation Control Scripts.
- File management by the Installation Control Scripts.
- How to test your Installation Control Scripts.

All Installation Control Scripts perform product-specific, vendor-supplied operations:

`customize` script	Runs after the fileset is successfully loaded by `update` during an install or update.
`decustomize` script	Runs before fileset deletion by `rmfn` to perform removal operations specific to the particular fileset.

Installation control scripts perform a myriad of product-specific setup operations, such as:

- Performing product-specific requirements checks, such as prerequisites.
- Removing previously installed versions of the product.
- Removing obsolete files.
- Moving configuration files into place if absent.
- Modifying existing configuration files for new features.
- Rebuilding custom versions of configuration files.
- Creating device files or custom programs.
- Killing fileset-specific daemons as part of fileset removal.

General Guidelines for Installation Control Scripts

- Emphasize performance, even if it means a script must be written as a program. All Installation Control Scripts execute serially, and directly affect the total time required to complete an installation.
- Rebuilding the kernel in a postload script is strongly discouraged. It's a complex and trouble-prone process. The `update` command has the ability to rebuild the kernel for you by specifying the `B` fileset flag (`ff`) in the Product Specification File.
- The results of disk space analysis are only valid while the `update` code itself is running. Files copied or removed during script execution are not reflected in the disk space analysis results.
- Installation control scripts are left on customers' systems after installations. Hence they should be well-engineered and well-commented.

Location and Execution of Installation Control Scripts

This section details the location and execution of each type of installation control script.

Details Common to Both Types of Installation Control Scripts

- Installation control scripts are always run as superuser. Use appropriate caution.
- Installation control scripts must be executable.
- Each script must set its own `PATH` variable.
- Neither `update` nor `rmfn` require that the system be shut down. Hence, Installation Control Scripts must work correctly on both quiet single-user systems and active multi-user systems. They must deal properly with unremovable running programs. They might have to shut down or start up processes themselves to succeed.
- Installation control scripts should be re-runnable. If a script is run more than once, it should produce the same results each time. The second execution should not produce any error messages or leave the system in a state different than before it was run.

  For example, if you must move a file from `/etc/newconfig` to another location, use the `cpio -p` command to copy it rather than the `mv` command to move it, or check for the absence of the `/etc/newconfig` version before attempting the move.

  Note: Use the *cpio*(1) command rather than *cp*(1) because `cpio` copies permission bits (owner/group/mode).
- Installation control scripts must exit with return value zero (`exit 0`) if no serious errors are detected (no ERRORs emitted as described in the "Input and Output From Installation Control Scripts" section later in this appendix). They must return 1 (`exit 1`) in case of any serious ERRORs and WARNINGs.

Details Specific to customize Scripts

- After the product has loaded, `customize` script files are called as:

  `/system/`*fileset*`/customize` *architecture*

  where *fileset* is the name of the fileset the script acts on and *architecture* is either `HP-MC68020` or `HP-PA`.

- The current working directory when a `customize` script is executed is `update`'s destination directory. Some applications can be installed in directories other than the default destination (`/`), depending on whether the `C` fileset flag (`ff`) is set in the Product Specification File.

- If the fileset is relocatable (the `C` fileset flag is not set), then you must:
  - □ Ensure that the script uses relative pathnames for files it manipulates.
  - □ Test the fileset for correct loading and functionality when loaded to a non-root destination.

  The `update` command changes the working directory to the destination directory before running the `customize` script.

- The `update` command only runs Installation Control Scripts for filesets that load successfully. If a fileset fails to load correctly, `update` logs the following message:

  ```
  WARNING: Skipping customize script for fileset
           because the fileset did not load successfully.
  ```

- `customize` scripts always run after their filesets are completely and successfully loaded, either after all critical filesets are loaded, or after all filesets are loaded (and the system reboots, if appropriate). Which scenario applies depends on if the `B` (reboot) fileset flag is specified in the Product Specification File.

Details Specific to decustomize Scripts

- `decustomize` script files are executed twice. The first time, `rmfn` runs the script to check whether the fileset is removable. The syntax of this first invocation looks like this:

  `/system/`*fileset*`/decustomize` *architecture* `check`

  where *fileset* is the name of the fileset the script acts on, *architecture* is either `HP-MC68020` (for Series 300/400) or `HP-PA` (for Series 600, 700, and 800), and `check` means to just check, not to do anything.

  The second time, `rmfn` runs the script just prior to removing all files loaded with *fileset*. The syntax of the second invocation looks like this:

  `/system/`*fileset*`/decustomize` *architecture*

- The first invocation of the `decustomize` script is the only chance it has of preventing files from being removed. This happens if the exit status from the script is a value of `1`. If the exit status is a `0`, then the removal process continues.
- The `rmfn` command does not remove files on remotely mounted file systems.
- For release 9.0, if a file to be deleted is a symbolic link to another file, `rmfn` removes only the symbolic link, not the target. Any pathnames that contain symbolic links are followed and the appropriate file is removed. For releases earlier than 9.0, if a file to be deleted is a symbolic link to another file, `rmfn` removes the target and not the link.
- `decustomize` scripts must not shut down and reboot the system, even for critical filesets, since `rmfn` might be initiated during the installation process.

Execution of Other Commands by Installation Control Scripts

- Every command used by an installation control script is a potential source of failure due to:

  1. The fact that the command may not exist on the system.
  2. Command/kernel or command/library incompatibilities.

  Your script can use any command conditionally, if it checks first for its existence and executability, and if it does not fail when the command is unavailable.

  You can also deliver the command in your fileset if it is suspected it will not be on the system. That is, the fileset is self-contained.

- Do not use or depend on commands in any other fileset in a `customize` script, because fileset load order is not guaranteed. Specifying the `D` fileset flag in the Product Specification File will guarantee that all filesets are loaded before the script is run.
- Pathnames of commands run by the script should be absolute pathnames, or relative to the paths specified in the `PATH` variable. (This is not really a restriction, just a reminder.)

Input and Output From Installation Control Scripts

- Installation control scripts must not be interactive. This includes messages such as, `Press return to continue`. Once initiated, the installation process is designed to run to completion without intervention.
- Installation control scripts must write serious errors to standard error (`echo >&2`) and other messages to standard output. Installation control scripts must not write directly to `/dev/console` or attempt any other method of writing directly to the display. During an interactive installation process, the human interface has control of the screen. Also, `update` has a non-interactive mode—it can run from a command line or *cron*(1M).

  At this time, standard output and standard error from installation control scripts are appended to either the `update` log file (`/tmp/update.log`) or the remove log file (`/tmp/rmfn.log`). They are not handled separately.
- Only minimal, essential information should be emitted by installation control scripts. Ideally, no output is emitted if all goes well.
- Begin and end messages are logged around the execution of each script. Before an installation control script is run, a message is logged, for example:

  ```
          * Beginning customize script for fileset:
  ```

  Next appear any messages from the script itself. When the script completes, one of the following messages, depending on the return value, is logged:

  ```
          * customize script for fileset succeeded.

  ERROR:    Customize script for fileset <   > failed.  You might
            want to make appropriate corrections and re-invoke it
            manually later using the command line shown above.
  ```

- For easiest review of the log file, output from installation control scripts must conform to the following log file format conventions wherever possible.

  1. Never emit blank lines.
  2. All output lines must have one of these forms:

     ```
     ERROR:   text
     WARNING: text
     NOTE:    text
     blank    text
     ```

     In each case, the keyword must begin in column 1, and the *text* must begin in column 10 (indented nine blanks).
  3. Choose the keyword (`ERROR`, `WARNING`, `NOTE`, or blank) as follows:

`ERROR:`	Something happened which must grab the user's attention. Cannot proceed, or need corrective action (to be taken later).
`WARNING:`	Can continue, but it's important the user knows something went wrong or requires attention.
`NOTE:`	Something out of the ordinary or worth special attention; not just a status message.
blank	Generic progress and status messages (keep them to a necessary minimum).

     Do not start a line with an asterisk (*) character. This is reserved for standard operation messages, so that they can be easily distinguished from product messages, warnings and errors.
  4. If the message text requires more than one line (79 columns), break it into several lines. Begin each continuation line with nine blanks. For example:

     ```
     NOTE:    To install your new graphics package, it was
              necessary to turn on the lights in the next room.
              Turn them off when you leave.
     ```
  5. Do not use tabs for anything. Simply avoid them.

- Scripts execute other commands, which might unexpectedly fail and emit output not in the above format. Wherever you suspect a failure is possible or likely, and it is reasonable to do so, redirect the standard output and/or error of the executed command to `/dev/null` or to a temporary file. Then emit a proper-format message based on the return code or on output from the command. For example:

  ```
  if /bin/grep bletch /etc/bagel 2> /dev/null
  then echo "ERROR: Cannot find bletch in /etc/bagel." >&2
  fi
  ```

- The following are other suggested conventions, to help your script's output look compatible with the output from `update'` or `rmfn`.

  1. Use full sentences wherever possible. Avoid terseness.
  2. Start sentences and phrases with a capital letter and end with a period.
  3. Put two blanks after colons and periods; one after semicolons and commas.
  4. Uppercase first letters of phrases after colons. (This helps break up the message into digestible "bites" of information.)
  5. Surround product, fileset, directory, and file names, and other unpredictable variant strings with quotes. For example:

     ```
     echo "ERROR:  Cannot open file \"$file\"."
     ```

     Exception: When referring to an object name in a string owned and controlled by the script (such as `/etc/mnttab`), you can leave off the quotes.
  6. Speak in present tense. Avoid "would", "will", and so forth. Also avoid past tense except where necessary.
  7. Use "cannot" rather than "can't", "could not", "couldn't", "unable to", "failed to", and similar phrases.
  8. When reporting an internal error (unlikely in a shell script), start the message string with "Internal error:".
  9. Keep your messages simple, neutral, and direct.

File Management for Installation Control Scripts

- If any files in the previous release of your fileset changed names or became obsolete, the installation control script should remove the old versions. No other agent takes care of this.

Note It is necessary to handle cleanup of any previous release whose update to the new release is “supported”. Sometimes this is more than just the previous release.

Also, it is wise to leave old cleanup code from previous releases in a new version of an installation control script, if there is no significant risk of failure or spurious messages, nor significant time or space penalty for doing so.

- If your fileset's name changes between releases, your new installation control script should remove the old `/system/`*fileset* directory and the file `/etc/filesets/`*fileset* (using `rm -rf`).
- Any files created (built) by a `customize` script and left on the system when it completes should have their names added to the fileset's `/etc/filesets/`*fileset* file with the proper full, absolute pathnames. It is sufficient to append the names, one per line, to the end of the file. For relocatable filesets, if the installation destination (initial working directory) is other than `/`, prefix the new filename accordingly.
- Any files deleted by a `customize` script that were loaded by `update` from the install media should have their names removed from the fileset's `/etc/filesets/`*fileset* file. This might require passing a copy through *grep*(1) to a temporary file, for example:

```
cd /etc/filesets
grep -v '^/old/file/name'  < fileset > /tmp/live_files
mv /tmp/live_files fileset
```

- If an installation control script writes to the `/etc/inittab` file, it must do so only if the file already exists. Do not accidentally create an incomplete `inittab` file.

Testing Installation Control Scripts

Here are some steps to follow when testing Installation Control Scripts.

These steps do not cover all cases. There might still be some problems with your scripts even after doing this testing. For example, you will test loading/removing individual filesets. There might be some interactions that are discovered only after all the filesets are combined on/ deleted from the system.

Here are a couple of reasons, specific to the `update` process, why the steps below do not cover all cases.

- You will update your system from one version of a release to another version of the same release. You might miss a problem where your script uses a command that is new to one version of a release, and you don't execute the script on the old version of the release, before that new command is installed on the system.
- You might do your testing on a fully loaded system and miss a problem where you execute a command in your script that is not part of the base system. If the user chooses not to load the fileset containing that particular command, your script will fail.

Testing customize Scripts

Test the standalone case on all supported systems:

1. `rm /tmp/update.log`
2. Run `/etc/update` to get your fileset(s) and any others of interest.
3. After the installation completes, check the `/tmp/update.log` file for any problems, either in format or contents of the logged messages.
4. Study the resulting file system to see if the script did what you expected it to do. If you have a complex script, run the tests for your product that you feel will give you confidence your product has been installed correctly on the system.

Testing decustomize Scripts

Test the standalone case on all supported systems:

1. If you want to start with a fresh system, remove `/tmp/update.log` and run `/etc/update` to load the filesets of interest.
2. After the installation completes, check the log messages in the file `/tmp/update.log` for any problems.
3. `rm /tmp/rmfn.log`
4. Now run `/etc/rmfn` to get your function(s) removed.
5. Check the log messages in the file `/tmp/rmfn.log` for any problems during fileset removal.
6. Study the resulting file system to see if the `decustomize` script did what you expected it to do. If you have a complex `decustomize` script, run the tests for your product that give you confidence your product has been deleted correctly from the system.

B

Re-Creating a Product Specification File

There are occasions when you might have media available and you want to apply **fpkg** capabilities to it. For example, you have a CD-ROM media and want to make an update tape containing all or part of the filesets on the CD-ROM. The *fpkg*(1M) command is the one to use for creating an update tape, but that requires the Product Specification File that was used to create the CD-ROM media (or one functionally similar).

The `-r` option directs the output of the **fpkg** command to be a Product Specification File.

Prerequisites and Conditions

The `-r` option can only be used with the `-v` option to increase verbosity and the `-L` option to specify an alternate log file. Including any of the other valid **fpkg** options along with the `-r` option on the command line causes a fatal error (along with an error message telling why).

Using fpkg to Re-Create a Product Specification File

To re-create a Product Specification File, use the following syntax:

```
fpkg [-v][-L logfile]-r media-directory > Product-Specification-File
```

The argument *media-directory* is the directory under which the appropriate media (either netdist or CD-ROM) will be found.

Re-Creating a Product Specification File From CD-ROM Media

In the case of CD-ROM media, the *media-directory* command argument is the directory at which the CD-ROM drive is mounted. This directory is usually named `/UPDATE_CDROM`.

The `fpkg` command verifies the correct structure of this media by looking for a subdirectory named `system` which contains files named `INDEX` and `INFO`. A fatal error occurs if either of these files are not found.

Re-Creating a Product Specification File From netdist Media

For netdist media, the *media-directory* command argument points to a directory which has been created as a result of either the *updist*(1M) command or the *fpkg*(1M) command. A typical directory in this case might be named `/netdist/8.07/MR/700`.

The `fpkg` command verifies the correct structure of this media by looking in each immediate subdirectory for files named `netdist.index` and `netdist.info`, and for a directory named `product`. Any subdirectory is ignored as netdist media if it does not have any such structure. A fatal error occurs if no subdirectory under the argument directory has any such structure. Further testing by `fpkg` ensures the suitability of each subdirectory as netdist media. Failing any of these tests results in the directory being ignored as a fileset directory structure.

The Command Output

The result of using `fpkg` with the `-r` *media-directory* option is:

- A Product Specification File printed to `stdout` (default is the monitor). Output should be redirected to a permanent file.
- Error messages are directed to `stderr`.
- Log messages are sent to `/tmp/fpkg.log` (unless you named another log file with the `-L` option).

The Product Specification File resulting from each type of media is slightly different because of differences in the structure of CD-ROM media and netdist media.

Output From CD-ROM Media

The CD-ROM media has all of its filesets distributed under a single root directory. There is no separation of filesets on a CD, which is necessary on tape media. For this reason, `fpkg` must individually specify the source location of each file contained in a particular fileset. In addition, the mode of each file is explicitly specified. This makes for a lengthy Product Specification File.

Note Any filesets on CD-ROM media which are found to be secured will be omitted from the output. These filesets will be noted in the log file only if the verbose (`-v`) option was used.

Output From netdist Media

The directories under a netdist distribution hierarchy are organized by fileset. For the sake of consistency, each file is listed as in reading from CD-ROM media. The Product Specification File contains the filesets in the order in which they are read from the netdist directory. However, the sequence in which the subdirectories are read may not be identical to the sequence in which they must be put onto the tape media. The correct ordering is done by `fpkg` after reading the Product Specification File and prior to writing to the tape.

Example

The following command line will read the CD-ROM media mounted at /UPDATE_CDROM and will write the Product Specification File into the file /tmp/psf.

```
fpkg -rv /UPDATE_CDROM > /tmp/psf
```

You can edit the Product Specification File **/tmp/psf**, but deleting entire fileset structures is the only recommended action.

The following command line will result in an update tape which is the functional equivalent of the CD-ROM media (the default tape device /dev/rmt/0m is used here).

```
fpkg -m tape /tmp/psf
```

Glossary

The following terms, names, and acronyms are used when packaging and installing software applications on HP-UX.

Alternate Log File
You can override the default location of the `fpkg` log file with an alternate name using the `-L` *logfile* option of `fpkg`.

Critical Filesets
Critical filesets contain software that is critical to the correct operation of the destination host. Critical filesets are those marked with the rebuild and reboot (`B`) fileset flag. During the load phase, critical filesets are loaded and customized before other filesets.

customize Script
Optional, vendor-supplied script associated with a fileset that is executed before or after installing the corresponding fileset.

decustomize Script
An optional, vendor-supplied script associated with a fileset that is executed by `rmfn`.

Dependee Fileset
A fileset on which some other fileset depends. For example, if fileset A depends on fileset B, then B is a dependee of A.

Dependencies
Dependencies between filesets are used to enforce corequisites. A fileset that depends on another fileset requires that the other fileset be installed in order for the first fileset to be usable. For example, if fileset A depends on fileset B, then B must be installed in order for A to be usable (A won't work without B).

Depender

A fileset that depends on some other fileset. For example, if fileset A depends on fileset B, then A is a depender of B.

Destination Host

A host (local or remote) on which software is installed or copied.

fileset

HP-UX software is organized into **filesets** and **partitions**. Individual files are logically grouped into filesets, and filesets are logically grouped into partitions. For a description of existing HP-UX filesets and partitions, see *Installing and Updating HP-UX*.

fpkg

This command allows a software vendor to create software products and package them onto Tape Media or Network Media. Network Media can be accessed directly and can be served to other hosts by the `netdistd` command.

Host

Same as a "system".

INDEX File

An `INDEX` file provides attribute and organizational information about partitions and filesets.

INFO File

An `INFO` file provides information about the files within a fileset. This information includes file type, mode, size, and pathname.

Installation Control Scripts

Optional, vendor-supplied scripts that are run during `update` and `rmfn`. Includes the `customize` script for `update` and the `decustomize` script for `rmfn`.

Installed Product

A product that has been installed on a host so that its files can be used by end-users. Contrast with a product residing in Network Media on a host's file system, sometimes referred to as a "served product".

Keyword

A word (or phrase) that tells `fpkg` about an attribute of the product. Keywords also have a corresponding shorthand notation. Either the full keyword or the shorthand notation can be used in the Product Specification File.

Local Host

The host that `update` or `updist` is being run on. Essentially equivalent to the administrative host, though the term local host is used along with remote host when talking about destination hosts.

Locatable Fileset

The files in a locatable fileset can be installed relative to an arbitrary destination directory on a destination host. If a fileset is nonlocatable, then its files are always installed relative to root (`/`).

Locatable Software

Software that can be installed relative to an arbitrary destination directory on a destination host. If software is nonlocatable, its files are always installed relative to the root (`/`) directory.

Logging

The `fpkg` tool keeps a record of its actions (messages, errors, and other information) in a log file. The default location for the `fpkg` log file is `/tmp/fpkg.log`. This can be overridden by invoking `fpkg` with the `-L` *logfile* option.

Media Format

The organization of media based on the HP-UX data model.

netdistd

The network server command; it serves Network Media simultaneously to multiple `update` processes on remote hosts.

Network Media

One type of HP-UX Media. This media type uses a file system to store the software products and control files needed by `netdistd` to use the media (i.e., all of the files in the products and the various `netdistd` control files reside in a directory structure with a single, common root).

Network Server

An alternate source for software installation and updates. The `netdistd` command manages a network server and interfaces with the install agent.

Nonlocatable Fileset Flag

An attribute of a fileset (set by the keyword `ff C`) which states that the fileset cannot be installed into any non-root destination directory. It must be loaded relative to root (`/`). A fileset with this flag is a nonlocatable fileset.

Nonlocatable Software

Software that is always installed relative to the root (`/`) directory. If software is locatable, its files can be installed under an arbitrary destination directory on a destination host.

partition

HP-UX software is organized into **filesets** and **partitions**. Individual files are logically grouped into filesets, and filesets are logically grouped into partitions. For a description of existing HP-UX filesets and partitions, see *Installing and Updating HP-UX*.

Product Specification File

The input file used to define the structure and attributes of the products to be packaged by `fpkg`.

Rebuild and Reboot Fileset Flag

An attribute of a fileset (set by the keyword `ff B`) which states that a destination host must have the kernel rebuilt and be rebooted after the fileset is installed. All filesets marked with a `B` flag will be loaded before the kernel is rebuilt. A fileset with this flag is considered to be a critical fileset.

rmfn

This command can interactively or non-interactively remove products from a system in units of filesets or partitions.

HP-UX Media

The term for generically referring to media used by HP-UX. There are two types of HP-UX Media that `fpkg` can make: Network Media and Tape Media. A media contains the software product files and the catalog of product information used for control of the selection and installation of the products.

Serial Media
A synonym for Tape Media.

Served Product
A product contained in a Network Media which is provided to `update` or `updist` through a `netdistd` server. Products contained in Network Media can be served to other hosts.

Tape Media
One type of HP-UX Media. This media type uses `tar` to store software products and control files needed by `update` to use the media (i.e., all of the files in the products and the various control files reside in a single `tar` archive). Such an archive usually resides on a serial media such as a DDS, cartridge, or nine-track tape, though a Tape Media can be a simple, regular file that contains the `tar` archive.

Tar Media
A form of installation media that exists as a `tar`-archive file, usually resident on a tape. A synonym for Tape Media.

update
The command that you execute to install or update software.

updist
This command is similar to `update`, except that it installs or updates the HP-UX system or application files as "fileset packages" in a special directory. This allows the system to be a network file distribution (netdist) server. The network server daemon (`netdistd`) finds the files in this special directory and supplies them to a remote `update` process on request.

Index

HEWLETT PACKARD

Reorder No. or
Manual Part No.
B2355-90031

**Manufacturing
Part No.
B2355-90031**

B2355-90031