Image Packaging System Man Pages

Part No: E21796
March 2012, E21796-02

ORACLE

120305@25097

Contents

Preface

This document provides the man pages for the Oracle Solaris 11 system installation tools.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes commands available with the operating system.
- Section 1M describes commands that are used chiefly for system maintenance and administration purposes.
- Section 5 contains miscellaneous documentation such as character-set tables.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `man` command for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . .	Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, *filename* . . . .
|	Separator. Only one of the arguments separated by this character can be specified at a time.
{ }	Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL	This section occurs only in subsection 3R to indicate the protocol description file.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.
OPTIONS	This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists

	alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.
USAGE	This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.
FILES	This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
ATTRIBUTES	This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See the `attributes`(5) man page for more information.

SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

REFERENCE

User Commands

Name packagemanager – GUI for the Image Packaging System

Synopsis

```
/usr/bin/packagemanager [options]

/usr/bin/packagemanager [-hiRu] [--help]
    [--info-install file] [--update-all]
    [--image-dir dir]

/usr/bin/packagemanager [file]
```

Description `packagemanager` is the graphical user interface for `pkg`(5), the Image Packaging System software.

The Package Manager enables you to perform the following tasks:

- Search, install, and remove packages.
- Add, remove, and modify publishers.
- Create, remove, and manage boot environments.

If the *file* operand is specified and its suffix is `.p5i`, `packagemanager` launches in Web Install mode, which adds one or more publishers and a number of packages for each publisher.

Options The following options are supported:

`-h` or `--help`
Displays a usage message.

`-i` or `--info-install` *file*
Allows you to specify a `.p5i` file to run `packagemanager` in Web Install mode. The *file* must have the suffix `.p5i`.

`-R` or `--image-dir` *dir*
Operate on the image rooted at *dir*, rather than the image discovered automatically.

`-U` or `--update-all`
Update all installed packages that have updates available.

Note – If the `package/pkg`, `package/pkg/package-manager`, or `package/pkg/update-manager` packages need to be updated, `packagemanager` first updates these packages and then restarts to carry out any remaining updates.

Operands *file* A Web Install file. This file must have the suffix `.p5i`. See the Package Manager online help for more information about Web Install.

Examples **EXAMPLE 1** Operate on the Current Image

Invoke `packagemanager` on the current image.

```
$ packagemanager
```

EXAMPLE 2 Operate on a Specified Image

Invoke `packagemanager` in the image stored at `/aux0/example_root`.

```
$ packagemanager -R /aux0/example_root
```

EXAMPLE 3 Invoke in Web Install Mode

Invoke `packagemanager` in Web Install mode.

```
$ packagemanager ~/test.p5i
```

Exit Status

The following exit values are returned:

`0` Everything worked.

`1` An error occurred.

`2` Invalid command line options were specified.

Files

Since `pkg`(5) images can be located arbitrarily within a larger file system, the token `$IMAGE_ROOT` is used to distinguish relative paths. For a typical system installation, `$IMAGE_ROOT` is equivalent to /.

`$IMAGE_ROOT/var/pkg`	Metadata directory for a full or partial image.
`$IMAGE_ROOT/.org.opensolaris,pkg`	Metadata directory for a user image.
	Within the metadata of a particular image, certain files and directories contain information useful during repair and recovery. The token `$IMAGE_META` is used to refer to the top-level directory that contains the metadata. `$IMAGE_META` is typically one of the two paths given above.
`$IMAGE_META/gui-cache`	Location for cached metadata maintained by `packagemanager` to speed up program startup and switching between publishers.

Other paths within the `$IMAGE_META` directory hierarchy are Private, and are subject to change.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg/package-manager`
Interface Stability	Uncommitted

See Also pm-updatemanager(1), pkg(1), pkg(5)

Package Manager online help

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Notes `packagemanager` needs to be invoked with sufficient privilege to operate on an image's files and directories.

Name pkg – Image Packaging System retrieval client

Synopsis

```
/usr/bin/pkg [options] command [cmd_options] [operands]

/usr/bin/pkg refresh [--full] [publisher ...]

/usr/bin/pkg install [-nvq] [-g path_or_uri ...] [--accept]
    [--licenses] [--no-be-activate] [--no-index] [--no-refresh]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    [--reject pkg_fmri_pattern ...] pkg_fmri_pattern ...

/usr/bin/pkg uninstall [-nvq] [--no-be-activate] [--no-index]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    pkg_fmri_pattern ...

/usr/bin/pkg update [-fnvq] [-g path_or_uri ...] [--accept]
    [--licenses] [--no-be-activate] [--no-index] [--no-refresh]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    [--reject pkg_fmri_pattern ...] [pkg_fmri_pattern ...]

/usr/bin/pkg list [-Hafnsuv] [-g path_or_uri ...]
    [--no-refresh] [pkg_fmri_pattern ...]

/usr/bin/pkg info [-lr] [-g path_or_uri ...] [--license]
    [pkg_fmri_pattern ...]

/usr/bin/pkg contents [-Hmr] [-a attribute=pattern ...]
    [-g path_or_uri ...] [-o attribute ...] [-s sort_key]
    [-t action_type ...] [pkg_fmri_pattern ...]

/usr/bin/pkg search [-HIaflpr] [-o attribute ...]
    [-s repo_uri] query

/usr/bin/pkg verify [-Hqv] [pkg_fmri_pattern ...]

/usr/bin/pkg fix [--accept] [--licenses] [pkg_fmri_pattern ...]

/usr/bin/pkg revert [-nv] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    (--tagged tag-name ... | path-to-file ...)

/usr/bin/pkg mediator [-aH] [-F format] [mediator ...]

usr/bin/pkg set-mediator [-nv] [-I implementation]
    [-V version] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    mediator ...

/usr/bin/pkg unset-mediator [-nvIV] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
```

```
    [--deny-new-be | --require-new-be] [--be-name name]
    mediator ...

/usr/bin/pkg variant [-H] [variant.variant_name ...]

/usr/bin/pkg change-variant [-nvq] [-g path_or_uri ...]
    [--accept] [--licenses] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    variant_name=value ...

/usr/bin/pkg facet [-H] [facet_name ...]

/usr/bin/pkg change-facet [-nvq] [-g path_or_uri ...]
    [--accept] [--licenses] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    facet_name=[True|False|None] ...

/usr/bin/pkg avoid [pkg_fmri_pattern ...]

/usr/bin/pkg unavoid [pkg_fmri_pattern ...]

/usr/bin/pkg freeze [-n] [-c reason] [pkg_fmri_pattern] ...

/usr/bin/pkg unfreeze [-n] [pkg_name_pattern] ...

/usr/bin/pkg property [-H] [propname ...]

/usr/bin/pkg set-property propname propvalue

/usr/bin/pkg add-property-value propname propvalue

/usr/bin/pkg remove-property-value propname propvalue

/usr/bin/pkg unset-property propname ...

/usr/bin/pkg publisher [-HPn] [publisher ...]

/usr/bin/pkg set-publisher [-Ped] [-k ssl_key] [-c ssl_cert]
    [-g origin_to_add | --add-origin origin_to_add ...]
    [-G origin_to_remove | --remove-origin origin_to_remove ...]
    [-m mirror_to_add | --add-mirror mirror_to_add ...]
    [-M mirror_to_remove | --remove-mirror mirror_to_remove ...]
    [--enable] [--disable] [--no-refresh]
    [--reset-uuid] [--non-sticky] [--sticky]
    [--search-after publisher] [--search-before publisher]
    [--search-first]
    [--approve-ca-cert path_to_CA]
    [--revoke-ca-cert hash_of_CA_to_remove]
    [--unset-ca-cert hash_of_CA_to_remove]
    [--set-property name_of_property=value]
    [--add-property-value name_of_property=value_to_add]
    [--remove-property-value name_of_property=value_to_remove]
    [--unset-property name_of_property_to_delete]
    publisher
```

```
/usr/bin/pkg set-publisher -p repo_uri
    [-Ped] [-k ssl_key] [-c ssl_cert]
    [--non-sticky] [--sticky]
    [--search-after publisher] [--search-before publisher]
    [--search-first]
    [--approve-ca-cert path_to_CA]
    [--revoke-ca-cert hash_of_CA_to_remove]
    [--unset-ca-cert hash_of_CA_to_remove]
    [--set-property name_of_property=value]
    [--add-property-value name_of_property=value_to_add]
    [--remove-property-value name_of_property=value_to_remove]
    [--unset-property name_of_property_to_delete]
    [publisher]

/usr/bin/pkg unset-publisher publisher ...

/usr/bin/pkg history [-Hl] [-t [time | time-time],...]
    [-o column,...] [-n number]

/usr/bin/pkg purge-history

/usr/bin/pkg rebuild-index

/usr/bin/pkg update-format

/usr/bin/pkg version

/usr/bin/pkg help

/usr/bin/pkg image-create [-FPUfz] [--force]
    [--full | --partial | --user] [--zone]
    [-k ssl_key] [-c ssl_cert]
    [--no-refresh] [--variant variant_name=value ...]
    [-g path_or_uri | --origin path_or_uri ...]
    [-m uri | --mirror uri ...]
    [--set-property name_of_property=value]
    [--facet facet_name=(True|False) ...]
    [(-p | --publisher) [name=]repo_uri] dir
```

Description `pkg` is the retrieval client for the Image Packaging System. With a valid configuration, `pkg` can be invoked to create locations for packages to be installed, called images, and install packages into those images. Packages are published by publishers, who can make their packages available at one or more repositories, or in package archives. `pkg` retrieves packages from a publisher's repository or package archives and installs the packages into an image.

A publisher name identifies a person, group of persons, or an organization as the source of one or more packages. To avoid publisher name collisions and help identify the publisher, a best practice is to use a domain name that represents the entity publishing the packages as a publisher name.

A repository is a location where clients can publish and retrieve package content (files contained within the package such as programs and documents) and metadata (information

about the package such as its name and description). As an example, a publisher named `example.org` might have their repository located at the URI `http://example.org/repository`.

`pkg` can also uninstall packages, refresh publisher metadata (such as the list of available packages), validate package installation in an image, and query the image for various tokens. These queries can also be made of `pkg`(5) repositories.

Images can be of three types: full images, capable of providing a complete system; partial images, which are linked to a full image (parent image), but do not provide a complete system on their own; and user images.

Options The following options are supported:

`-R` *dir*
Operate on the image rooted at *dir*. If no directory was specified or determined based on environment, the default is /. See the "Environment Variables" section for more information.

`--help` or `-?`
Display a usage message.

Sub-commands The following subcommands are supported:

`refresh [--full] [`*publisher* `...]`
Updates the client's list of available packages and publisher metadata for each publisher specified. If no publishers are specified, the operation is performed for all publishers.

With `--full`, force a full retrieval of all publisher metadata, instead of attempting an incremental update, and request that any proxies used during the operation ignore cached data. This option exists for troubleshooting purposes and should not be used on a regular basis.

`install [-nvq] [-g` *path_or_uri* `...] [--accept] [--licenses] [--no-be-activate] [--no-index] [--no-refresh] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`] [--reject` *pkg_fmri_pattern* `...]` *pkg_fmri_pattern* `...`
Installs and updates packages to the newest version that match *pkg_fmri_pattern* allowed by the packages installed in the image. To explicitly request the latest version of a package, use `latest` for the version portion of *pkg_fmri_pattern*. For example, specify `vim@latest`.

Some configuration files might be renamed or replaced during the installation process. For more information on how the package system determines which files to preserve, and how they are preserved during package operations, see "File Actions" in the `pkg`(5) man page.

If a package is on the avoid list, installing it removes it from that list.

With `-g`, temporarily add the specified package repository or archive to the list of sources in the image from which to retrieve package data. If packages from the specified sources are

also available from configured publishers in the image, the client will retrieve content for those packages from the specified sources only. When deciding which version of a package to use, publishers configured in the image, but not found in the given sources take precedence. After install or update, any packages provided by publishers not found in the image are added to the image configuration without an origin. This option can be specified multiple times.

With `-n`, perform a trial run of the operation with no package changes made.

With `-q`, hide progress messages during the requested operation.

With `-v`, issue verbose progress messages during the requested operation, and display detailed planning information (such as changing facets, mediators, and variants). This option can be specified multiple times to increase the amount of planning information displayed.

With `--accept`, you indicate that you agree to and accept the terms of the licenses of the packages that are updated or installed. If you do not provide this option, and any package licenses require acceptance, the installation operation fails.

With `--licenses`, display all of the licenses for the packages that are installed or updated as part of this operation.

With `--no-backup-be`, do not create a backup boot environment.

With `--no-be-activate`, if a boot environment is created, do not set it as the active BE on the next boot. See `beadm`(1M) for more information.

With `--no-index`, do not update the search indexes after the operation has completed successfully.

With `--no-refresh`, do not attempt to contact the repositories for the image's publishers to retrieve the newest list of available packages and other metadata.

With `--backup-be-name`, name the created backup boot environment using the given argument. Use of `--backup-be-name` implies `--require-backup-be`. See also `beadm`(1M).

With `--be-name`, rename the newly created boot environment to be the argument given. Use of `--be-name` implies `--require-new-be`. See also `beadm`(1M).

With `--require-backup-be`, always create a backup boot environment if a new boot environment will not be created. Without this option, a backup boot environment is created based on image policy. See `be-policy` in "Image Properties" below for an explanation of when backup boot environments are created automatically.

With `--require-new-be`, always create a new boot environment. Without this option, a boot environment is created based on image policy. See `be-policy` in "Image Properties" below for an explanation of when boot environments are created automatically. This option cannot be combined with `--require-backup-be`.

With `--deny-new-be`, do not create a new boot environment. This operation is not performed if a new boot environment is required.

With `--reject`, prevent packages with names matching the given pattern from being installed. If matching packages are already installed, they are removed as part of this operation. Rejected packages that are the target of group dependencies are placed on the avoid list.

`uninstall [-nvq] [--no-be-activate] [--no-index] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`]` *pkg_fmri_pattern* `...`

Removes installed packages that match *pkg_fmri_pattern*.

If a package is the subject of a group dependency, uninstalling it places it on the avoid list. See the `avoid` subcommand below.

For all other options, refer to the `install` command above for usage and their effects.

`update [-fnvq] [-g` *path_or_uri* `...] [--accept] [--licenses] [--no-be-activate] [--no-index] [--no-refresh] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`] [--reject` *pkg_fmri_pattern* `...] [`*pkg_fmri_pattern* `...]`

With no arguments, or if asterisk (*) is one of the patterns provided, update all installed packages in the current image to the newest version allowed by the constraints imposed on the system by installed packages and publisher configuration. To explicitly request the latest version of a package, use `latest` for the version portion of *pkg_fmri_pattern*. For example, specify `vim@latest`.

If *pkg_fmri_pattern* is provided, `update` replaces packages that are installed, and that match *pkg_fmri_pattern*, with the newest version allowed by the patterns and the constraints imposed on the system by installed packages and publisher configuration. Versions older or newer than what is already installed can be specified to perform in-place downgrades or upgrades of specific packages. Updating specific packages across package rename or obsolete boundaries is not supported.

Any preserved configuration files that are part of packages to be downgraded by `update` and that have been changed since the original version was installed are renamed using the extension `.update`. For more information about how the package system determines which files to preserve, and how these files are preserved during package upgrades, see "File Actions" in the `pkg`(5) man page.

With the `-f` option, do not execute the client up-to-date check when updating all installed packages.

For all other options, refer to the `install` command above for usage and their effects.

`list [-Hafnsuv] [-g` *path_or_uri* `...] [--no-refresh] [`*pkg_fmri_pattern* `...]`

Without arguments, display a list of packages in the current image, including information such as version and installed state. With arguments, display information for the specified packages. By default, package variants for a different architecture or zone type are excluded. The usual output is in three columns:

```
NAME (PUBLISHER)            VERSION           IFO
system/core-os              0.5.11-0.169      i--
x11/wm/fvwm (fvwm.org)      2.6.1-3           i--
```

The first column contains the name of the package. If the publisher from which the package is installed (or available, if not installed) is not the first in the publisher search order, then the publisher name is listed in parentheses after the package name. The second column contains the release and branch versions of the package. See the `pkg`(5) man page for information about release and branch versions and about variants.

The last column contains a set of flags that show the status of the package:

- An `i` in the `I` column shows that the package is installed.
- An `f` in the `F` column shows that the package is frozen.
- An `o` in the `O` column shows that the package is obsolete. An `r` in the `O` column shows that the package has been renamed (a form of obsoletion).

With `-H`, omit the headers from the listing.

With `-a`, list installed packages and the newest version of packages that are available for installation. Packages are considered to be available for installation if they are allowed by the installed incorporations and by the image's variants. If one or more patterns are specified, then the newest version matching the specified pattern and allowed by any installed incorporations and the image's variants is listed. Without `-a`, list only installed packages.

With `-f` and `-a`, list all versions of all packages for all variants regardless of incorporation constraints or installed state. To explicitly list the latest version of a package when using these options, use `latest` for the version portion of *pkg_fmri_pattern*. For example, specify `vim@latest`.

With `-g`, use the specified package repository or archive as the source of package data for the operation. This option can be specified multiple times. Use of `-g` implies `-a` if `-n` is not specified.

With `-n`, display the newest versions of all known packages, regardless of installed state.

With `-s`, display a one-line short-form giving the package name and summary. This option can be used with `-a`, `-n`, `-u` or `-v`.

With `-u`, list only packages with newer versions available. This option cannot be used with `-g`.

With -v, show full package FMRIs, including publisher and complete version, all in the first column (the VERSION column disappears). This option can be used with -a, -n, or -u.

With --no-refresh, do not attempt to contact the repositories for the image's publishers to retrieve the newest list of available packages.

info [-lr] [-g *path_or_uri* ...] [--license] [*pkg_fmri_pattern* ...]
Display information about packages in a human-readable form. Multiple FMRI patterns can be specified. With no patterns, display information on all installed packages in the image.

With -g, use the specified package repository or archive as the source of package data for the operation. This option can be specified multiple times. Use of -g implies -r.

With -l, only display information for installed packages. This is the default.

With -r, match packages based on the newest available versions, retrieving information for packages not currently installed (if necessary) from the repositories of the image's configured publishers. At least one package must be specified when using this option. Without -r, only installed packages are displayed by default.

With --license, display the license texts for the packages. This option can be combined with -l or -r.

contents [-Hmr] [-a *attribute=pattern* ...] [-g *path_or_uri* ...] [-o *attribute*,...] [-s *sort_key*] [-t *action_type* ...] [*pkg_fmri_pattern* ...]
Display the contents (action attributes) of packages. With no options or operands, display the value of the path attribute for actions installed in the current image, sorted alphabetically by attribute value. For information about actions and their attributes, see "Actions" in the pkg(5) man page. See also the list of pseudo attribute names below.

With -a, limit the output to those actions that have an attribute named in the option argument with a value that matches the (glob) pattern in the option argument (following the attribute name with an equals sign). If multiple -a options are given, then actions matching any of them are displayed.

With -g, display information for packages that could be installed in this image from the specified package repository or archive. Packages that could be installed include packages that are currently installed and other packages that satisfy criteria for installation in this image, such as variant and facet restrictions. This option can be specified multiple times. Use of -g implies -r.

With -m, display all attributes of all actions in the specified packages, including actions that could not be installed in this image.

With -o, display the listed attributes, sorted according to the values of the first attribute listed. The -o option can be specified multiple times, or multiple attributes can be specified as the argument to one -o option by separating the attribute names with commas. Only actions that have the requested attributes are displayed.

With `-r`, display information for the newest available versions of packages that could be installed in this image from the repositories of the publishers configured in this image. Packages that could be installed include packages that are currently installed and other packages that satisfy criteria for installation in this image, such as variant and facet restrictions. At least one package must be specified when using this option.

With `-s`, sort actions by the specified action attribute. If not provided, the default is to sort by path or by the first attribute specified by the `-o` option. The `-s` option can be specified multiple times.

With `-t`, only list actions of the type specified. Multiple types can be specified in a comma-separated list. This option can be specified multiple times.

With `-H`, omit the headers from the listing.

With *pkg_fmri_pattern*, display information only for those named packages.

Several special pseudo attribute names are available for convenience:

`action.hash`	The value of the action's hash, if the action carries a payload.
`action.key`	The value of the action's key attribute. For example, for a `file` action, the key attribute is the path to the file. Some actions do not have a key attribute.
`action.name`	The name of the action. For example, for a file action, this is `file`.
`action.raw`	All attributes of matching actions.
`pkg.fmri`	The full FMRI of the package containing the action, such as `pkg://solaris/web/amp@0.5.11,5.11-0.169:20110705T153434Z`.
`pkg.name`	The name of the package containing the action, such as `web/amp`.
`pkg.publisher`	The publisher of the package containing the action, such as `solaris`.
`pkg.shortfmri`	The short form FMRI of the package containing the action, such as `pkg://solaris/web/amp@0.5.11,5.11-0.169`

The `contents` and `search` subcommands are related: Both query the system for the contents of packages. The `contents` subcommand displays actions in one or more installed or installable packages, filtering the output based on the specified options. The `search` subcommand approaches the query from the other direction, displaying the names of all packages that contain a user-supplied token.

Each subcommand is capable of formulating some queries of which the other is capable. Care should be taken in choosing the subcommand, as a given query can be more naturally formulated in one than in the other.

`search [-HIaflpr] [-o` *attribute*`,...] [-s` *repo_uri*`]` *query*

Search for matches to the *query*, and display the results. Which tokens are indexed are action-dependent, but can include content hashes and path names. For information about actions and their attributes, see "Actions" in the `pkg`(5) man page. See also the list of pseudo attribute names in `pkg contents` above and in `-o` below.

By default, queries are interpreted as a series of terms to be matched exactly. The `?` and `*` characters can be used as `glob`(3C)-style wildcards, allowing more flexible query matches.

With `-H`, omit the headers.

With `-I`, use a case-sensitive search.

By default, and with `-a`, perform the search and display information about the matching actions.

By default, `search` prunes results from packages older than the currently installed version and from package versions excluded by current incorporations. Use `-f` to show all results, regardless of package version.

With `-l`, search the image's installed packages.

With `-o`, the columns of the results can be controlled. The `-o` option can be specified multiple times, or multiple attributes can be specified as the argument to one `-o` option by separating the attribute names with commas. In addition to the pseudo attributes outlined above, the following attributes are defined for search results:

`search.match`	Corresponds to the string that matched the search query.
`search.match_type`	Corresponds to the attribute that contained the string that matched the search query.

With `-p`, display packages that have some actions that match each query term. Using this option is equivalent to putting angle brackets (<>) around each term in the query. See below for more description of the <> operator.

By default, and with `-r`, search the repositories corresponding to the image's publishers.

With `-s`, search the `pkg`(5) repository located at the given URI. This can be specified multiple times. Package archives are not supported.

Both `-l` and `-r` (or `-s`) can be specified together, in which case both local and remote searches are performed.

In addition to simple token matching and wildcard search, a more complicated query language is supported. Phrases can be searched for by using single or double quotation marks (`'` or "). Be sure to take your shell into account so that `pkg` actually sees the `'` or ".

Boolean search using AND and OR is supported. Field, or structured, queries are supported. The syntax for these is *pkg_name*:*action_type*:*key*:*token*. Missing fields are implicitly wildcarded. A search for `:basename:pkg` matches all action types in all packages with a key of `basename` and that match the token `pkg`. Explicit wildcards are supported in the `pkg_name` and `token` fields. The `action_type` and `key` must match exactly.

To convert actions to the packages that contain those actions, use `<>`. With the `-a` option, searching for `token` results in information about the actions matching `token`, while searching for `<token>` results in a list of packages containing actions that matched `token`.

`verify [-Hqv] [`*pkg_fmri_pattern* `...]`
Validate the installation of packages in the current image. If current signature policy for related publishers is not `ignore`, the signatures of each package are validated based on policy. See `signature-policy` in "Image Properties" below for an explanation of how signature policies are applied.

With `-H`, omit the headers from the verification output.

With `-q`, print nothing, but return failure if there are any fatal errors.

With `-v`, include informational messages regarding packages.

`fix [--accept] [--licenses] [`*pkg_fmri_pattern* `...]`
Fix any errors reported by `pkg verify`. Verification of installed package content is based on a custom content analysis that might return different results than those of other programs.

With `--accept`, you indicate that you agree to and accept the terms of the licenses of the packages that are updated or installed. If you do not provide this option, and any package licenses require acceptance, the operation fails.

With `--licenses`, display all of the licenses for the packages to be installed or updated as part of this operation.

`revert [-nv] [--no-be-activate] [--no-backup-be | --require-backup-be]`
`[--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`]`
`(--tagged` *tag-name* `... |` *path-to-file* `...)`
Revert files to their as-delivered condition. Either all files tagged with a particular value, or individual files can be reverted. File ownership and protections are also restored.

Caution – Reverting some editable files to their default values can make the system unbootable, or cause other malfunctions.

For all other options, refer to the `install` command above for usage and their effects.

`mediator [-aH] [-F` *format*`] [`*mediator* `...]`
Display the current selected version and/or implementation of all mediators, or with arguments, only the mediators specified.

With `-a`, list the mediations that can be set for currently installed packages.

With `-F`, specify an alternative output format. Currently, only `tsv` (Tab Separated Values) is valid.

With `-H`, omit the headers from the listing.

`set-mediator [-nv] [-I` *implementation*`] [-V` *version*`] [--no-be-activate] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`]` *mediator* `...`

Set the version and/or implementation for the specified mediators in the current image.

With `-I`, set the implementation of the mediated interface to use. By default, if no version is specified, all implementation versions are allowed. To specify an implementation with no version, append an at sign (@).

With `-V`, set the version of the mediated interface to use.

If the specified mediator version and/or implementation is not currently available, any links using the specified mediators are removed.

For all other options, refer to the `install` command above for their usage and effects.

`unset-mediator [-nvIV] [--no-be-activate] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`]` *mediator* `...`

Revert the version and/or implementation of the specified mediators to the system default.

With `-I`, revert only the implementation of the mediated interface.

With `-V`, revert only the version of the mediated interface.

For all other options, refer to the `install` command above for their usage and effects.

`variant [-H] [variant.`*variant_name* `...]`

Without arguments, display the current values of all variants set in this image. With arguments, display the value of each specified `variant.`*variant_name* set in this image.

With `-H`, omit the headers from the listing.

See "Facets and Variants" in the `pkg`(5) man page for more information about variants.

`change-variant [-nvq] [-g` *path_or_uri* `...] [--accept] [--licenses] [--no-be-activate] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`]` *variant_name*`=`*value* `...`

Change the values of the specified variants set in the current image.

For option usage and effects, refer to the `install` command above.

Changing the value of a variant can cause package content to be removed, updated, or installed. Changing a variant value can also cause entire packages to be installed, updated, or removed to satisfy the new image configuration. See "Facets and Variants" in the pkg(5) man page for more information about variants.

`facet [-H] [`*facet_name* `...]`
Without arguments, display the current values of all facets that have been explicitly set in this image by using the `pkg change-facet` command. With arguments, display the value of each specified *facet_name* set in this image.

With `-H`, omit the headers from the listing.

See "Facets and Variants" in the pkg(5) man page for more information about facets.

`change-facet [-nvq] [-g` *path_or_uri* `...] [--accept] [--licenses] [--no-be-activate] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`]` *facet_name*`=[True|False|None] ...`
Change the values of the specified facets set in the current image.

Facets can be set to `True` or `False`. Setting a facet to `None` applies the default value of `True` to that facet; thus, any actions subject to the facet will be installed. See "Actions" in the pkg(5) man page for information about actions..

For option usage and effects, refer to the `install` command above.

Changing the value of a facet can cause package content to be removed, updated, or installed. Changing a facet value can also cause entire packages to be installed, updated, or removed to satisfy the new image configuration. See "Facets and Variants" in the pkg(5) man page for more information about facets.

`avoid [`*pkg_fmri_pattern* `...]`
Avoids the specified packages if they are the target of a group dependency by placing the package names that currently match the specified patterns on the avoid list. Only packages that are not currently installed can be avoided. If a package is currently the target of a group dependency, uninstalling the package places it on the avoid list.

Without any arguments, display each avoided package along with any packages that have a group dependency on that package.

Packages that are on the avoid list are installed if needed to satisfy a required dependency. If that dependency is removed, the package is uninstalled.

`unavoid [`*pkg_fmri_pattern* `...]`
Remove the specified packages from the avoid list. Packages on the avoid list that match an installed package's group dependency cannot be removed using this subcommand. To remove a package from the avoid list that matches a group dependency, install the package.

Without any arguments, display the list of avoided packages.

`freeze [-n] [-c` *reason*`] [`*pkg_fmri_pattern*`] ...`
Freeze the specified packages to the versions specified. If no version is given, the package must be installed and is frozen at that installed version. When a package that is frozen is installed or updated, it must end up at a version that matches the version at which it was frozen. For example, if a package was frozen at 1.2, then it could be updated to 1.2.1, 1.2.9, 1.2.0.0.1, and so on. That package could not end up at 1.3, or 1.1. A publisher presented in the *pkg_fmri_pattern* is used to find matching packages. However, publisher information is not recorded as part of the freeze. A package is frozen with respect to its version only, not its publisher. Freezing a package that is already frozen replaces the freeze version with the newly specified version.

If no packages are provided, information about currently frozen packages is displayed: package names, versions, when the package was frozen, and any associated reasons.

Freezing a package does not prevent removal of the package. No warning is displayed if the package is removed.

With `-c`, record the *reason* with the packages that are frozen. The reason is shown if a freeze prevents an installation or update from succeeding.

With `-n`, perform a trial run of the operation, displaying the list of packages that would be frozen without freezing any packages.

`unfreeze [-n] [`*pkg_name_pattern*`] ...`
Remove the constraints that freezing imposes from the specified packages. Any versions provided are ignored.

With `-n`, perform a trial run of the unfreeze, displaying the list of packages that would be unfrozen without unfreezing any packages.

`property [-H] [`*propname* `...]`
Display image property information. With no argument, display the names and values for all image properties. If a specific list of property names is requested, display the names and values for those properties. See "Image Properties" below for descriptions of image properties.

With `-H`, omit the headers from the listing.

`set-property` *propname propvalue*
Update an existing image property or add a new image property.

`add-property-value` *propname propvalue*
Add a value to an existing image property or add a new image property.

`remove-property-value` *propname propvalue*
Remove a value from an existing image property.

`unset-property` *propname* `...`
Remove an existing image property or properties.

`publisher [-HPn] [`*publisher* `...]`

Display publisher information. With no arguments, display the list of all publishers, their origin URIs, and mirrors in order of search preference. If specific publishers are requested, display detailed configuration for those publishers.

With `-H`, omit the headers from the listing.

With `-P`, display only the first publisher in the publisher search order.

With `-n`, display only enabled publishers.

`set-publisher [-Ped] [-k` *ssl_key*`] [-c` *ssl_cert*`] [-g` *origin_to_add* `| --add-origin` *origin_to_add* `...] [-G` *origin_to_remove* `| --remove-origin` *origin_to_remove* `...] [-m` *mirror_to_add* `| --add-mirror` *mirror_to_add* `...] [-M` *mirror_to_remove* `| --remove-mirror` *mirror_to_remove* `...] [--enable] [--disable] [--no-refresh] [--reset-uuid] [--non-sticky] [--sticky] [--search-after` *publisher*`] [--search-before` *publisher*`] [--search-first] [--approve-ca-cert` *path_to_CA*`] [--revoke-ca-cert` *hash_of_CA_to_remove*`] [--unset-ca-cert` *hash_of_CA_to_remove*`] [--set-property` *name_of_property=value*`] [--add-property-value` *name_of_property=value_to_add*`] [--remove-property-value` *name_of_property=value_to_remove*`] [--unset-property` *name_of_property_to_delete*`]` *publisher*

Update an existing publisher or add a package publisher. If no options affecting search order are specified, new publishers are appended to the search order and are thus searched last.

With `-P` or `--search-first`, set the specified publisher first in the search order. When installing new packages, this publisher is searched first. Updates to already installed packages come from the same publisher that originally provided the package as long as that publisher remains sticky. When `-P` or `--search-first` is used with `-p`, only added publishers are placed first in search order.

With `--non-sticky`, specify that higher ranked publishers than this one can provide updates to packages originally installed from this publisher.

With `--sticky`, specify that updates to packages that were installed from this publisher must also come from this publisher. This is the default behavior.

With `--search-before`, alter the publisher search order so that the publisher being modified is searched before the specified publisher. When used with `-p`, `--search-before` only applies to added publishers.

With `--search-after`, alter the publisher search order so that the publisher being modified is searched after the specified publisher. When used with `-p`, `--search-after` only applies to added publishers.

With `--approve-ca-cert`, add the given certificate as a CA certificate that is trusted. The hashes of the PEM representation of the user-approved CA certificates are listed in the detailed output of the `pkg publisher` command.

With `--revoke-ca-cert`, treat the certificate with the given hash of its PEM representation as revoked. The hashes of the user-revoked CA certificates are listed in the detailed output of the `pkg publisher` command.

With `--unset-ca-cert`, remove the certificate with the given hash from the list of approved certificates and the list of revoked certificates.

With `--set-property`, update an existing publisher property or add a new publisher property.

With `--add-property-value`, add a value to an existing publisher property or add a new publisher property.

With `--remove-property-value`, remove a value from an existing publisher property.

With `--unset-property`, remove an existing publisher property.

With `-c` and `-k`, specify client SSL certificate and key respectively.

With `-g` (`--add-origin`), add the specified URI or path as an origin for the given publisher. This should be the location of a package repository or archive.

With `-G` (`--remove-origin`), remove the URI or path from the list of origins for the given publisher. The special value * can be used to remove all origins.

With `--no-refresh`, do not attempt to contact the repositories for the image's publishers to retrieve the newest list of available packages and other metadata.

With `--reset-uuid`, choose a new unique identifier that identifies this image to its publisher.

With `-m` (`--add-mirror`), add the URI as a mirror for the given publisher.

With `-M` (`--remove-mirror`), remove the URI from the list of mirrors for the given publisher. The special value * can be used to remove all mirrors.

With `-p`, retrieve publisher configuration information from the specified repository URI. If a publisher is specified, then only the matching publisher is added or updated. If no publisher is specified, all publishers are added or updated as appropriate. This option cannot be combined with the `-g`, `--add-origin`, `--G`, `--remove-origin`, `-m`, `--add-mirror`, `-M`, `--remove-mirror`, `--disable`, `--enable`, `--no-refresh`, or `--reset-uuid` options.

With `-e` (`--enable`), enable the publisher. With `-d` (`--disable`), disable the publisher. A disabled publisher is not used when populating the package list or in certain package operations (install, uninstall, and update). However, the properties for a disabled publisher can still be set and viewed. If only one publisher exists, it cannot be disabled.

`/usr/bin/pkg set-publisher -p` *repo_uri* `[-Ped] [-k` *ssl_key*`] [-c` *ssl_cert*`] [--non-sticky] [--sticky] [--search-after` *publisher*`] [--search-before` *publisher*`] [--search-first] [--approve-ca-cert` *path_to_CA*`] [--revoke-ca-cert` *hash_of_CA_to_remove*`] [--unset-ca-cert` *hash_of_CA_to_remove*`] [--set-property` *name_of_property*`=`*value*`] [--add-property-value` *name_of_property*`=`*value_to_add*`] [--remove-property-value` *name_of_property*`=`*value_to_remove*`] [--unset-property` *name_of_property_to_delete*`] [`*publisher*`]`

With `-p`, retrieve publisher configuration information from the specified repository URI. If a publisher is specified, then only the matching publisher is added or updated. If no publisher is specified, all publishers are added or updated as appropriate. See `pkg set-publisher` above for descriptions of other options that can be used with the `-p` option. The `-p` option cannot be combined with the `-g`, `--add-origin`, `-G`, `--remove-origin`, `-m`, `--add-mirror`, `-M`, `--remove-mirror`, `--disable`, `--enable`, `--no-refresh`, or `--reset-uuid` options.

`unset-publisher` *publisher* `...`

Remove the configuration associated with the given publisher or publishers.

`history [-Hl] [-t [`*time* `|` *time*`-`*time*`],...] [-o` *column*`,...] [-n` *number*`]`

Display the command history of the applicable image.

With `-H`, omit the headers from the listing.

With `-t`, display log records for a comma-separated list of timestamps, formatted with `%Y-%m-%dT%H:%M:%S` (see `strftime`(3C)). To specify a range of times, use a hyphen (-) between a start and finish timestamp. The keyword `now` can be used as an alias for the current time. If the timestamps specified contain duplicate timestamps or overlapping date ranges, only a single instance of each duplicate history event is printed.

With `-l`, display log records in long format, which, in addition to the standard format, includes the outcome of the command, the time the command completed, the version and name of the client used, the name of the user who performed the operation, and any errors that were encountered while executing the command.

With `-n`, display only the specified number of most recent entries.

With `-o`, display output using the specified comma-separated list of column names. Valid column names are:

`be`	The name of the boot environment this operation was started on.
`be_uuid`	The `uuid` of the boot environment this operation was started on.
`client`	The name of the client.

`client_ver`	The version of the client.
`command`	The command line used for this operation.
`finish`	The time that this operation finished.
`id`	The user id that started this operation.
`new_be`	The new boot environment created by this operation.
`new_be_uuid`	The `uuid` of the new boot environment created by this operation.
`operation`	The name of the operation.
`outcome`	A summary of the outcome of this operation.
`reason`	Additional information on the outcome of this operation.
`snapshot`	The snapshot taken during this operation. This is only recorded if the snapshot was not automatically removed after successful operation completion.
`start`	The time that this operation started.
`time`	The total time taken to perform this operation. For operations that take less than one second, 0:00:00 is shown.
`user`	The username that started this operation.

If the `command` or `reason` columns are specified, they must be the last item in the `-o` list, in order to preserve output field separation. These two columns cannot be shown in the same `history` command.

An asterisk (*) is shown after the values for `be` or `new_be` if the boot environment is no longer present on the system.

The values for `be` and `new_be` are obtained by looking up the current boot environment name, using the `be_uuid` or `new_be_uuid` fields. If a boot environment was subsequently renamed, and later deleted, the values displayed for `be` and `new_be` are the values recorded at the time of the `pkg` operation.

`purge-history`
Deletes all existing history information.

`rebuild-index`
Rebuilds the index used by `pkg search`. This is a recovery operation not intended for general use.

`update-format`
Updates the format of the image to the current version. Once this operation has completed, the image can no longer be used with older versions of the `pkg`(5) system.

`version`
Display a unique string identifying the version of `pkg`(1). This string is not guaranteed to be comparable in any fashion between versions.

`image-create [-FPUfz] [--force] [--full | --partial | --user] [--zone] [-k` *ssl_key*`] [-c` *ssl_cert*`] [--no-refresh] [--variant` *variant_name*`=`*value* `...] [-g` *path_or_uri* `| --origin` *path_or_uri* `...] [-m` *uri* `| --mirror` *uri* `...] [--set-property` *name_of_property*`=`*value*`][--facet` *facet_name*`=(True|False) ...] [(-p | --publisher) [`*name*`=]`*repo_uri*`]` *dir*
At the location given by *dir*, create an image suitable for package operations. The default image type is user, as given by the `-U` (`--user`) option. The image type can be set to a full image (`--F` or `--full`) or to a partial image (`-P` or `--partial`) linked to the full image enclosing the given *dir* path. Additional origins can be specified using `-g` or `--origin`. Additional mirrors can be specified using `--m` or `--mirror`.

A package repository URI must be provided using the `-p` or `--publisher` option. If a publisher name is also provided, then only that publisher is added when the image is created. If a publisher name is not provided, then all publishers known by the specified repository are added to the image. An attempt to retrieve the catalog associated with this publisher is made following the initial creation operations.

For publishers using client SSL authentication, a client key and client certificate can be registered via the `-c` and `-k` options. This key and certificate are used for all publishers added during image creation.

If the image is to be run within non-global zone context, then the `-z` (`--zone`) option can be used to set an appropriate variant.

With `-f` (`--force`), force the creation of an image over an existing image. This option should be used with care.

With `--no-refresh`, do not attempt to contact the repositories for the image's publishers to retrieve the newest list of available packages and other metadata.

With `--variant`, set the specified variant to the indicated value. See “Facets and Variants” in the `pkg`(5) man page for more information about variants.

With `--facet`, set the specified facet to the indicated value. See “Facets and Variants” in the `pkg`(5) man page for more information about facets.

With `--set-property`, set the specified image property to the indicated value. See “Image Properties” below for descriptions of image properties.

Image Properties The following properties define characteristics of the image. These properties store information about the purpose, content, and behavior of the image. To view the current values of these properties in the image, use the `pkg property` command. To modify the values of these properties, use the `pkg set-property` and `pkg unset-property` commands.

`be-policy`
(string) Specify when a boot environment is created during packaging operations. The following values are allowed:

`default` Apply the default BE creation policy, `create-backup`.

`always-new` Requires a reboot for all package operations by performing them in a new BE set as active on the next boot. A backup BE is not created unless explicitly requested.

This policy is the safest, but is more strict than most sites need since no packages can be added without a reboot.

`create-backup`
For package operations that require a reboot, a new BE is created and set as active on the next boot. If packages are modified or content that could affect the kernel is installed and the operation affects the live BE, a backup BE is created but not set as active. A backup BE can also be explicitly requested.

This policy is potentially risky only if newly installed software causes system instability, which is possible but relatively rare.

`when-required`
For package operations that require a reboot, a new BE is created and set as active on the next boot. A backup BE is not created unless explicitly requested.

This policy carries the greatest risk since if a packaging change to the live BE makes further changes impossible, there might be no recent BE to which one can fallback.

`ca-path`
(string) A path name that points to a directory where CA certificates are kept for SSL operations. The format of this directory is specific to the underlying SSL implementation. To use an alternate location for trusted CA certificates, change this value to point to a different directory. See the `CApath` portions of `SSL_CTX_load_verify_locations`(3openssl) for requirements for the CA directory.

Default value: `/etc/openssl/certs`

`check-certificate-revocation`
(boolean) If this is set to `True`, the package client attempts to contact any CRL distribution points in the certificates used for signature verification to determine whether the certificate has been revoked since being issued.

Default value: `False`

`flush-content-cache-on-success`
(boolean) If this is set to `True`, the package client removes the files in its content-cache when install or update operations complete. For update operations, the content is removed

only from the source BE. When a packaging operation next occurs in the destination BE, the package client flushes its content cache if this option has not been changed.

This property can be used to keep the content-cache small on systems with limited disk space. This property can cause operations to take longer to complete.

Default value: `True`

`mirror-discovery`
(boolean) This property tells the client to discover link-local content mirrors using mDNS and DNS-SD. If this property is set to `True`, the client attempts to download package content from mirrors it dynamically discovers. To run a mirror that advertises its content via mDNS, see `pkg.depotd`(1M).

Default value: `False`

`send-uuid`
(boolean) Send the image's Universally Unique Identifier (UUID) when performing network operations. Although users can disable this option, some network repositories might refuse to talk to clients that do not supply a UUID.

Default value: `True`

`signature-policy`
(string) Determine what checks will be performed on manifests when installing, updating, modifying, or verifying packages in the image. The final policy applied to a package depends on the combination of image policy and publisher policy. The combination will be at least as strict as the stricter of the two policies taken individually. By default, the package client does not check whether certificates have been revoked. To enable those checks, which might require the client to contact external web sites, set the `check-certificate-revocation` image property to `True`. The following values are allowed:

`ignore`	Ignore signatures for all manifests.
`verify`	Verify that all manifests with signatures are validly signed, but do not require all installed packages to be signed. This is the default value.
`require-signatures`	Require that all newly installed packages have at least one valid signature. The `pkg fix` and `pkg verify` commands also warn if an installed package does not have a valid signature.
`require-names`	Follow the same requirements as `require-signatures` but also require that the strings listed in the `signature-required-names` property appear as a common name of the certificates used to verify the chains of trust of the signatures.

`signature-required-names`
(list of strings) A list of names that must be seen as common names of certificates while validating the signatures of a package.

`trust-anchor-directory`
(string) The path name of the directory that contains the trust anchors for the image. This path is relative to the image. The default value is `ignore`.

`use-system-repo`
(boolean) This property indicates whether the image should use the system repository as a source for image and publisher configuration and as a proxy for communicating with the publishers provided. The default value is `False`. See `pkg.sysrepo`(1M) for information about system repositories.

Publisher Properties

The following properties define signature policy for a particular publisher. The image properties of the same name define signature policy for the image. To view the current values of these properties for a particular publisher, use the `pkg publisher` *publisher_name* command. To modify the values of these publisher signature policy properties, use the `--set-property` and `--unset-property` options of the `pkg set-publisher` command.

`signature-policy`
(string) This property functions identically to the image property of the same name except that it only applies to packages from the particular publisher.

`signature-required-names`
(list of strings) This property functions identically to the image property of the same name except that it only applies to packages from the particular publisher.

Examples

EXAMPLE 1 Create an Image With Publisher Configured

Create a new, full image, with publisher `example.com`, stored at `/aux0/example_root`.

```
$ pkg image-create -F -p example.com=http://pkg.example.com:10000 \
/aux0/example_root
```

EXAMPLE 2 Create an Image, Specifying Additional Origins and Mirror

Create a new, full image, with publisher `example.com`, that also has an additional mirror, two additional origins, and is stored at `/aux0/example_root`.

```
$ pkg image-create -F -p example.com=http://pkg.example.com:10000 \
-g http://alternate1.example.com:10000/ \
-g http://alternate2.example.com:10000/ \
-m http://mirror.example.com:10000/ \
/aux0/example_root
```

EXAMPLE 3 Create an Image With No Publisher Configured

Create a new, full image with no publishers configured at `/aux0/example_root`.

```
$ pkg image-create -F /aux0/example_root
```

EXAMPLE 4 Install a Package

Install the latest version of the `widget` package in the current image.

```
$ pkg install application/widget
```

EXAMPLE 5 List Specified Contents of a Package

List the contents of the `system/file-system/zfs` package. Display the action name, the mode of the file (if defined), the size (if defined), the path, and the target (if a link). Limit the action to types `dir`, `file`, `link`, and `hardlink`, since specifying the `action.name` attribute, which is available for all actions, displays a line for all actions, which is not desired here.

```
$ pkg contents -t dir,file,link,hardlink \
-o action.name,mode,pkg.size,path,target system/file-system/zfs
ACTION.NAME MODE PKG.SIZE PATH                TARGET
dir         0755          etc
dir         0755          etc/fs
dir         0755          etc/fs/zfs
link                      etc/fs/zfs/mount    ../../../usr/sbin/zfs
link                      etc/fs/zfs/umount   ../../../usr/sbin/zfs
dir         0755          etc/zfs
dir         0755          kernel
dir         0755          kernel/drv
dir         0755          kernel/drv/amd64
file        0755  1706744 kernel/drv/amd64/zfs
file        0644      980 kernel/drv/zfs.conf
dir         0755          kernel/fs
dir         0755          kernel/fs/amd64
hardlink                  kernel/fs/amd64/zfs  ../../../kernel/drv/amd64/zfs
...
```

EXAMPLE 6 List Specified Contents of Two Packages

List the contents of `web/browser/firefox` and `mail/thunderbird`, limiting the display to just the package name and path attributes of actions whose `path` attribute ends in `.desktop` or `.png`.

```
$ pkg contents -o pkg.name,path -a path=\*.desktop \
-a path=\*.png web/browser/firefox mail/thunderbird
PKG.NAME            PATH
web/browser/firefox usr/share/applications/firefox.desktop
mail/thunderbird    usr/share/applications/thunderbird.desktop
web/browser/firefox usr/share/pixmaps/firefox-icon.png
mail/thunderbird    usr/share/pixmaps/thunderbird-icon.png
...
```

EXAMPLE 7 Search for a Package

Search the package database for the token `bge`.

EXAMPLE 7 Search for a Package *(Continued)*

```
$ pkg search bge
INDEX       ACTION VALUE                      PACKAGE
driver_name driver bge                        pkg:/driver/network/bge@0.5.11-0.169
basename    file   kernel/drv/sparcv9/bge     pkg:/driver/network/bge@0.5.11-0.169
basename    file   kernel/drv/amd64/bge       pkg:/driver/network/bge@0.5.11-0.169
pkg.fmri    set    solaris/driver/network/bge pkg:/driver/network/bge@0.5.11-0.169
```

The token is in the package `driver/network/bge` both as the basename for the file action representing `/kernel/drv/`*arch*`/bge` and as a driver name.

EXAMPLE 8 Search for Packages that Depend on the Specified Package

Search for installed packages that depend on `package/pkg`.

```
$ pkg search -l 'depend::package/pkg'
INDEX       ACTION VALUE                    PACKAGE
incorporate depend package/pkg@0.5.11-0.169 pkg:/consolidation/ips/ips-incorporation@0.5.11-0.169
require     depend package/pkg@0.5.11-0.169 pkg:/system/install@0.5.11-0.169
require     depend package/pkg@0.5.11-0.169 pkg:/package/pkg/system-repository@0.5.11-0.169
```

EXAMPLE 9 Search for Dependencies

Search for all `incorporate` dependencies in installed packages.

```
$ pkg search -l 'depend:incorporate:'
INDEX       ACTION VALUE                            PACKAGE
incorporate depend pkg:/BRCMbnx@0.5.11,5.11-0.133  pkg:/consolidation/osnet/osnet-incorporation@0.5
incorporate depend pkg:/BRCMbnxe@0.5.11,5.11-0.133 pkg:/consolidation/osnet/osnet-incorporation@0.5
...
```

EXAMPLE 10 Add a Publisher

Add a new publisher `example.com`, with a repository located at `http://www.example.com/repo`.

```
$ pkg set-publisher -g http://www.example.com/repo example.com
```

EXAMPLE 11 Add a Publisher With Key and Certificate

Add a new publisher `example.com`, with a secure repository located at `https://secure.example.com/repo`, and a key and certificate stored in the directory `/root/creds`.

```
$ pkg set-publisher -k /root/creds/example.key \
-c /root/creds/example.cert -g https://secure.example.com/repo \
example.com
```

EXAMPLE 12 Add and Automatically Configure a Publisher

Add a new publisher with a repository located at `/export/repo` using automatic configuration.

```
$ pkg set-publisher -p /export/repo
```

EXAMPLE 13 Add and Manually Configure a Publisher

Add a new publisher `example.com` with a repository located at `/export/repo/example.com` using manual configuration.

```
$ pkg set-publisher -g /export/repo example.com
```

EXAMPLE 14 Verify All Signed Packages

Configure an image to verify all signed packages.

```
$ pkg set-property signature-policy verify
```

EXAMPLE 15 Require All Packages To Be Signed

Configure an image to require all packages to be signed, and require the string `example.com` to be seen as a common name for one of the certificates in the chain of trust.

```
$ pkg set-property signature-policy require-names example.com
```

EXAMPLE 16 Require All Packages From a Specified Publisher To Be Signed

Configure an image so that all packages installed from publisher `example.com` must be signed.

```
$ pkg set-publisher --set-property signature-policy=require-signatures \
example.com
```

EXAMPLE 17 Require a Specified String in the Chain of Trust

Add the string `foo` to the image's list of common names that must be seen in a signature's chain of trust to be considered valid.

```
$ pkg add-property-value signature-require-names foo
```

EXAMPLE 18 Remove a String From the Chain of Trust for a Specified Publisher

Remove the string `foo` from the list of common names that must be seen to validate a signature for the publisher `example.com`.

```
$ pkg set-publisher --remove-property-value signature-require-names=foo \
example.com
```

EXAMPLE 19 Add a Trusted CA Certificate

Add the certificate stored in `/tmp/example_file.pem` as a trusted CA certificate for the publisher `example.com`.

EXAMPLE 19 Add a Trusted CA Certificate *(Continued)*

```
$ pkg set-publisher --approve-ca-cert /tmp/example_file.pem \
example.com
```

EXAMPLE 20 Revoke a Certificate

Revoke the certificate with the hash `a12345` for publisher `example.com`, preventing the certificate from validating any signatures for packages from `example.com`.

```
$ pkg set-publisher --revoke-ca-cert a12345 example.com
```

EXAMPLE 21 Forget About a Certificate

Make `pkg` forget that the certificate `a12345` was ever added or revoked by the user.

```
$ pkg set-publisher --unset-ca-cert a12345 example.com
```

EXAMPLE 22 Downgrade a Package

Downgrade the installed package `foo@1.1` to an older version.

```
$ pkg update foo@1.0
```

EXAMPLE 23 Switch Conflicting Package Installation

In the case of two conflicting packages, change which package is installed. Suppose package A depends on either package B or package C, and B and C are mutually exclusive. If A and B are installed, use the following command to switch to using C instead of B without uninstalling A:

```
$ pkg install --reject B C
```

EXAMPLE 24 List Packages in a Package Archive

List all versions of all packages in a package archive.

```
$ pkg list -f -g /my/archive.p5p
```

EXAMPLE 25 List Packages in a Package Repository

List all versions of all packages in a repository.

```
$ pkg list -f -g http://example.com:10000
```

EXAMPLE 26 Display Information About a Package in a Package Archive

Display the package information for the latest version of a package in a package archive. The package might or might not be currently installed.

```
$ pkg info -g /my/archive.p5p pkg_name
```

EXAMPLE 27 Display Contents of a Package in a Package Archive

Display the contents of a package in a package archive. The package is not currently installed.

```
$ pkg contents -g /my/archive.p5p pkg_name
```

EXAMPLE 28 Remove All Publisher Origins and Mirrors

Remove all of the origins and mirrors for a publisher and add a new origin.

```
$ pkg set-publisher -G '*' -M '*' -g http://example.com:10000 \
example.com
```

Environment Variables

`PKG_IMAGE`
The directory containing the image to use for package operations. Ignored if `-R` is specified.

`PKG_CLIENT_CONNECT_TIMEOUT`
Seconds to wait trying to connect during transport operations (for each attempt) before the client aborts the operation. A value of 0 means wait indefinitely.

Default value: 60

`PKG_CLIENT_LOWSPEED_TIMEOUT`
Seconds below the `lowspeed` limit (1024 bytes/sec) during transport operations before the client aborts the operation. A value of 0 means do not abort the operation.

Default value: 30

`PKG_CLIENT_MAX_CONSECUTIVE_ERROR`
Maximum number of transient transport errors before the client aborts the operation. A value of 0 means do not abort the operation.

Default value: 4

`PKG_CLIENT_MAX_REDIRECT`
Maximum number of HTTP or HTTPS redirects allowed during transport operations before a connection is aborted. A value of 0 means do not abort the operation.

Default value: 5

`PKG_CLIENT_MAX_TIMEOUT`
Maximum number of transport attempts per host before the client aborts the operation. A value of 0 means do not abort the operation.

Default value: 4

`http_proxy`, `https_proxy`
HTTP or HTTPS proxy server.

Exit Status The following exit values are returned:

`0` Command succeeded.

`1` An error occurred.

`2` Invalid command line options were specified.

`3` Multiple operations were requested, but only some of them succeeded.

`4` No changes were made - nothing to do.

`5` The requested operation cannot be performed on a live image.

`6` The requested operation cannot be completed because the licenses for the packages being installed or updated have not been accepted.

`7` The image is currently in use by another process and cannot be modified.

`99` An unanticipated exception occurred.

Files A `pkg`(5) image can be located arbitrarily within a larger file system. In the following file descriptions, the token `$IMAGE_ROOT` is used to distinguish relative paths. For a typical system installation, `$IMAGE_ROOT` is equivalent to `/`

`$IMAGE_ROOT/var/pkg` Metadata directory for a full or partial image.

`$IMAGE_ROOT/.org.opensolaris,pkg` Metadata directory for a user image.

Within a particular image's metadata, certain files and directories can contain information useful during repair and recovery. The token `$IMAGE_META` refers to the top-level directory containing the metadata. `$IMAGE_META` is typically one of the two paths given above.

`$IMAGE_META/lost+found` Location of conflicting directories and files moved during a package operation. Location of unpackaged contents of a removed directory.

`$IMAGE_META/publisher` Contains a directory for each publisher. Each directory stores publisher-specific metadata.

Other paths within the `$IMAGE_META` directory hierarchy are private and are subject to change.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also pkgsend(1), pkg.depotd(1m), glob(3C), pkg(5), beadm(1M)

http://hub.opensolaris.org/bin/view/Project+pkg/

Name pkgdepend – Image Packaging System dependency analyzer

Synopsis

```
/usr/bin/pkgdepend [options] command [cmd_options] [operands]

/usr/bin/pkgdepend generate [-IMm] -d dir [-d dir]
    [-D name=value] [-k path] manifest_file

/usr/bin/pkgdepend resolve [-moSv] [-d output_dir]
    [-s suffix] manifest_file ...
```

Description `pkgdepend` is used to generate and resolve dependencies for packages. A package might depend on files from other packages. `pkgdepend` is typically used in two passes: file dependency generation and file-to-package resolution.

The `generate` subcommand examines the content of a package and determines what external files the package needs.

The `resolve` subcommand takes the list of files from the `generate` step, and searches a reference set of packages to determine the names of the packages that contain those dependent files. The reference set of packages that are searched for the dependent files are the packages that are currently installed on the publisher's system.

Several components of delivered files are used as sources of dependency information:

ELF	ELF headers in delivered files are analyzed for dependency information, with the `-k` and `-D` options modifying the information obtained. For more details on ELF dependencies, see `ldd`(1) and the *Linker and Libraries Guide*.
Scripts	Shell scripts that contain `#!` lines referencing an interpreter result in a dependency on the package that delivers that interpreter.
Python	Python scripts are first analyzed as scripts. In addition, any imports the script declares might also serve as sources of dependency information.
Hard links	Hard links in manifests result in a dependency on the package that delivers the link target.
SMF	Delivered SMF service manifests that include `require_all` dependencies result in dependencies on the packages that deliver the SMF manifests that provide those FMRIs.

Options The following options are supported:

`-R` *dir*	Operate on the image rooted at *dir*. If no directory was specified or determined based on environment, the default is /. See the "Environment Variables" section for more information.
`--help` or `-?`	Displays a usage message.

Sub-commands The following subcommands are supported:

`generate [-IMm] -d` *dir* `[-d` *dir*`] [-D` *name*`=`*value*`] [-k` *path*`]` *manifest_file*
Produce the dependencies on files of the manifest specified by *manifest_file*.

With `-I`, show the dependencies that are satisfied within *manifest_file*. Do not use the result of `-I` with `pkgdepend resolve`.

With `-M`, display a list of file types that could not be analyzed.

With `-m`, repeat the original manifest with any discovered dependencies added after.

With `-d`, add *dir* to a list of directories to search for the manifest's files.

For each `-D`, add the *value* as a way to expand the token *name* in run paths for ELF file dependencies.

For each `-k`, add the *path* to the list of run paths to search for kernel modules. Using the `-k` argument removes the default paths, which are `/kernel` and `/usr/kernel`.

Run paths such as those specified by the `-k` option can also be specified per action or per manifest by using the action or manifest attribute `pkg.depend.runpath`. The value of the `pkg.depend.runpath` attribute is a colon-separated string of the paths to use.

The use of `-k` is overridden by any `pkg.depend.runpath` attributes set in the manifest or action.

The special token `$PKGDEPEND_RUNPATH` can be used as one component of the `pkg.depend.runpath` attribute value in order to include the standard system run path for the file being analyzed.

In some cases, you might want to prevent automatic generation of dependencies. For example, if a package delivers a sample Python script that imports a set of modules, those modules imported by the sample script are not dependencies for the package that delivers the sample script. Use the action or manifest attribute `pkg.depend.bypass-generate` to prevent generating dependencies against the specified files.

The `pkg.depend.bypass-generate` values are Python regular expressions that match file names. The regular expressions are implicitly anchored at the start and end of the file path. The value given in the following example matches `this/that` but does not match `something/this/that/the/other`.

```
pkg.depend.bypass-generate=this/that
```

For more information about Python regular expression syntax, use the command `pydoc re` or see more complete documentation at `http://docs.python.org/dev/howto/regex.html`.

When `pkgdepend generate` input manifests contain SMF manifest files, any SMF services or instances declared by those SMF manifest files are included in the `pkgdepend` output. These SMF services or instances are included in the form of a `set` action with the name `org.opensolaris.smf.fmri`.

`resolve [-moSv] [-d` *output_dir*`] [-s` *suffix*`]` *manifest_file* `...`

Transform dependencies on files into dependencies on the packages that deliver those files. Dependencies are first resolved against the manifests given on the command line and then against the packages installed on the system. By default, the dependencies for each manifest are placed in a file named *manifest_file*`.res`.

With `-m`, repeat the manifest, with any dependencies produced by the `generate` step removed, before adding the resolved dependencies.

With `-o`, write the results to standard output. This option is intended for human consumption. Appending this output to a file might result in an invalid manifest. The `-d` or `-s` options are strongly recommended instead of `-o` for use in a pipe line for manifest processing.

With `-d`, write the resolved dependencies for each manifest provided in a separate file in *output_dir*. By default, each file has the same base name as the manifest that was the source of the dependencies written to that file.

With `-s`, for each output file, append *suffix* to the base name of the file that was the source of the resolved dependencies. A "." is prepended to *suffix* if it is not provided.

With `-S`, only resolve against the manifests given on the command line and not against the manifests installed on the system.

With `-v`, include additional package dependency debugging metadata.

Examples

EXAMPLE 1 Generate Dependencies

Generate the dependencies for the manifest written in `foo`, whose content directory is in `./bar/baz`, and store the results in `foo.fdeps`.

```
$ pkgdepend generate -d ./bar/baz foo > foo.fdeps
```

EXAMPLE 2 Resolve Dependencies

Resolve the file dependencies in `foo.fdeps` and `bar.fdeps` against each other and against the packages currently installed on the system.

```
$ pkgdepend resolve foo.fdeps bar.fdeps
$ ls *.res
foo.fdeps.res    bar.fdeps.res
```

EXAMPLE 3 Generate and Resolve Dependencies For Two Manifests

Generate the file dependencies for two manifests (`foo` and `bar`) and retain all the information in the original manifests. Then resolve the file dependencies and place the resulting manifests in `./res`. These resulting manifests can be used with `pkgsend publish`.

```
$ pkgdepend generate -d /proto/foo -m foo > ./deps/foo
$ pkgdepend generate -d /proto/bar -m bar > ./deps/bar
$ pkgdepend resolve -m -d ./res ./deps/foo ./deps/bar
$ ls ./res
foo    bar
```

EXAMPLE 4 Add Values To Tokens For ELF File Dependencies

Replace all `PLATFORM` tokens in the run paths in ELF files with `sun4v` and `sun4u` while generating the dependencies for the manifest written in `foo` whose content directory is in /.

```
$ pkgdepend generate -d / -D 'PLATFORM=sun4v' -D 'PLATFORM=sun4u' foo
```

EXAMPLE 5 Specify a Kernel Module Directory

Specify `/kmod` as the directory in which to find kernel modules when generating the dependencies for the manifest written in `foo` whose content directory is in /.

```
$ pkgdepend generate -d / -k /kmod foo
```

EXAMPLE 6 Bypass Dependency Generation

Append `opt/python` to the standard Python run path for a given Python script, and bypass dependency generation against all Python modules called `test` for a file delivered as `opt/python/foo/file.py`.

Avoid generating dependencies against any file delivered in `usr/lib/python2.6/vendor-packages/xdg`.

```
$ cat manifest.py
set name=pkg.fmri value=pkg:/mypackage@1.0,1.0
set name=pkg.summary value="My test package"
dir path=opt mode=0755 group=sys owner=root
dir path=opt/python mode=0755 group=sys owner=root
dir path=opt/python/foo mode=0755 group=sys owner=root
file NOHASH path=opt/python/__init__.py mode=0644 group=sys owner=root
file NOHASH path=opt/python/foo/__init__.py mode=0644 group=sys owner=root
#
# Add runpath and bypass-generate attributes:
#
file NOHASH path=opt/python/foo/file.py mode=0644 group=sys owner=root \
    pkg.depend.bypass-generate=.*/test.py.* \
    pkg.depend.bypass-generate=.*/testmodule.so \
    pkg.depend.bypass-generate=.*/test.so \
    pkg.depend.bypass-generate=usr/lib/python2.6/vendor-packages/xdg/.* \
```

EXAMPLE 6 Bypass Dependency Generation *(Continued)*

```
    pkg.depend.runpath=$PKGDEPEND_RUNPATH:/opt/python

$ pkgdepend generate -d proto manifest.py
```

Environment Variables

`PKG_IMAGE` Specifies the directory that contains the image to use for package operations. This value is ignored if `-R` is specified.

Exit Status

The following exit values are returned:

`0` Everything worked.

`1` An error occurred.

`2` Invalid command line options were specified.

`99` An unanticipated exception occurred.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also

`pkg`(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Name pkgdiff – compare package manifests

Synopsis

```
/usr/bin/pkgdiff [-i attribute ...] [-o attribute]
    [-v name=value ...] file1 file2
```

Description `pkgdiff` compares two package manifests and reports differences. `pkgdiff` sorts each manifest and action into a consistent order before comparison.

Output is in the following form:

+ *complete_action* This action is in *file2* but not in *file1*.

- *complete_action* This action is in *file1* but not in *file2*.

actionname keyvalue [*variant values, if any*]

- *attribute1=value1* This *attribute*,*value* is in *file1* but not in *file2*.

+ *attribute2=value2* This *attribute*,*value* is in *file2* but not in *file1*.

Actions with different variants but the same type and key attribute value are treated as separate actions for purposes of comparison. Thus, actions that change attributes are shown in their complete form rather than as attribute changes.

Options The following options are supported:

`-i` *attribute* Ignore *attribute* if present during comparisons. File hash values can be ignored with `-i` *hash*. This option cannot be used with the `-o` option. This option can be repeated.

`-o` *attribute* Only report differences in *attribute*. This option cannot be used with the `-i` option. This option elides any action changes that do not affect *attribute* on an action.

`-v` *name=value* Only compute differences for this variant value. For example, only compute differences for `arch=sparc`. This variant tag is removed for all actions before comparison. Only one value can be specified per variant. This option can be repeated for different variants.

Exit Status The following exit values are returned:

`0` No differences were found.

`1` Differences were found.

`>1` An error occurred.

`99` An unanticipated exception occurred.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also `pkg`(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Name pkgfmt – format a package manifest

Synopsis `/usr/bin/pkgfmt [-c|-d|-u] [`*package-manifest-file*`]`

Description `pkgfmt` without the `-c` or `-d` options formats a package manifest in a consistent manner, including wrapping lines at 80 characters, sorting actions by type, and sorting attributes. Lines that do not parse into actions (such as macros, comments, or transforms) do not appear in sorted order.

If no arguments are given, `pkgfmt` reads `stdin` until EOF, and then writes the formatted manifest to `stdout`. Any manifests specified on the command line are formatted in place.

`pkgfmt` with the `-c` option checks whether the manifests are formatted in `pkgfmt` style. The `-d` option displays the differences if the file is not properly formatted.

Options The following options are supported:

`-c` Check whether the manifest is formatted in the `pkgfmt` style.

`-d` Display manifest differences from the formatted version in unified form.

`-u` Do not wrap lines at 80 characters. This option is useful for applying traditional text processing tools to package manifests.

Exit Status The following exit values are returned:

`0` Command succeeded.

`1` The `-c` or `-d` options were specified, and one or more manifests are not in `pkgfmt` normal form, or an error occurred.

`2` Invalid command line options were specified.

`99` An unanticipated exception occurred.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also pkg(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Name pkglint – Image Packaging System package lint

Synopsis

```
/usr/bin/pkglint [-c dir] [-r uri] [-p regexp]
    [-f rcfile] [-b build_no] [-v]
    [-l uri] | manifest ...

/usr/bin/pkglint -L [-v]
```

Description `pkglint` runs a series of checks on one or more package manifests, optionally referencing another repository.

`pkglint` should be used during the package authoring process, prior to package publication. `pkglint` performs exhaustive testing on the manifests that might be too expensive to perform during normal operation of `pkgsend`(1) or `pkg.depotd`(1M). `pkglint` checks include tests for duplicate actions, missing attributes, and unusual file permissions.

Manifests for linting can be passed as a space-separated list of local files on the command line, or manifests can be retrieved from a repository.

When retrieving manifests from repositories, on first run `pkglint` creates and populates `pkg`(5) user images in the specified cache directory. If the `-r` option is supplied, a user image named *cache_dir*`/ref_image` is created for the reference repository. If the `-l` option is supplied, a user image named *cache_dir*`/lint_image` is created for the lint repository. No content is installed in these images. These images are only used by `pkglint` to retrieve manifests from the repositories.

Subsequent invocations of `pkglint` can reuse the cache directory and can omit any `-r` or `-l` arguments.

`pkglint` provides limited support for configuring publishers in the cache directory. Use `pkg`(1) to perform more complex publisher configuration on these images.

`pkglint` allows package authors to bypass checks for a given manifest or action. A manifest or action that contains the attribute `pkg.linted` set to `True` does not produce any lint output for that manifest or action.

More granular `pkg.linted` settings can be made using substrings of `pkglint` check names. For example, `pkg.linted.`*check*`.`*id* set to `True` bypasses all checks with the name *check*`.`*id* for the given manifest or action.

The behavior of `pkglint` can be configured by specifying a`pkglintrc` file. By default, `pkglint` searches in `/usr/share/lib/pkg/pkglintrc` and `$HOME/.pkglintrc` for configuration options. Use the `-f` option to specify a different configuration file.

During the lint run, any errors or warnings encountered are printed to `stderr`.

Options The following options are supported:

`-b` *build_no*	Specify a build number used to narrow the list of packages used during linting from lint and reference repositories. If no `-b` option is specified, the latest versions of packages are used. See also the `version.pattern` configuration property.
`-c` *cache_dir*	Specify a local directory used for caching package metadata from the lint and reference repositories.
`-l` *lint_uri*	Specify a URI representing the location of the lint repository. Both HTTP and file system based publication are supported. If you specify `-l`, then you must also specify `-c`.
`-L`	List the known and excluded lint checks and then exit. Display the short name and description of each check. When combined with the `-v` flag, display the method that implements the check instead of the description.
`-f` *config_file*	Configure the `pkglint` session using the *config_file* configuration file.
`-p` *regexp*	Specify a regular expression used to narrow the list of packages to be checked from the lint repository. All manifests from the reference repository are loaded (assuming they match the value for `-b`, if supplied), ignoring this pattern.
`-r` *repo_uri*	Specify a URI representing the location of the reference repository. If you specify `-r`, then you must also specify `-c`.
`-v`	Run `pkglint` in a verbose mode, overriding any `log_level` settings in the configuration file.
`--help` or `-?`	Display a usage message.

Files The `pkglintrc` configuration file takes the following key/value arguments:

`log_level`	The minimum level at which to emit lint messages. Lint messages lower than this level are discarded. The default value is `INFO`. Log levels in order of least to most severe are `DEBUG`, `INFO`, `WARNING`, `ERROR`, and `CRITICAL`.
`do_pub_checks`	If `True`, perform checks that might only make sense for published packages. The default value is `True`.
`pkglint.ext.*`	The plugin mechanism of `pkglint` allows for additional lint modules to be added at runtime. Any key that starts with `pkglint.ext.` takes a value that must be a fully-specified Python module. See the "Developers" section for more information.

`pkglint.exclude`	A space-separated list of fully specified Python modules, classes, or function names to omit from the set of checks performed.
`use_progress_tracker`	If `True`, use a progress tracker when iterating over manifests during lint runs. The default value is `True`.
`version.pattern`	A version pattern, used when specifying a build number to lint against (`-b`). If not specified in the configuration file, the `-b` option uses the pattern `*,5.11-0.`, matching all components of the 5.11 build, with a branch prefix of 0.

Developers To extend the set of checks performed by `pkglint`, subclass `pkg.lint.base.Checker` and its subclasses, `ManifestChecker`, `ActionChecker`, and `ContentChecker`. Add the Python module name that contains those classes to a new `pkglint.ext.` key in the configuration file.

Instances of those new subclasses are created by `pkglint` on startup. Methods inside each subclass with the special keyword argument `pkglint_id` are invoked during the course of the lint session. Those methods should have the same signature as the corresponding `check()` method in the super class. Methods should also be assigned a `pkglint_desc` attribute, which is used as the description printed by `pkglint -L`.

Parameters are available to `Checker` subclasses, allowing them to tune their behavior. The recommended parameter naming convention is *pkglint_id*.*name*. Parameter values can be stored in the configuration file, or accessed in manifests or actions retrieved using the `LintEngine.get_param()` method. When accessing parameters from the manifest, the prefix `pkg.lint` is prepended to the key name to ensure that `pkglint` parameters do not overlap with any existing action or manifest values.

Examples **EXAMPLE 1** First Run on a Particular Repository

Running a `pkglint` session for the first time on a given repository.

```
$ pkglint -c /space/cache -r http://localhost:10000 mymanifest.mf
```

EXAMPLE 2 Subsequent Run on the Same Repository

A subsequent run against the same repository used in Example 1.

```
$ pkglint -c /space/cache mymanifest-fixed.mf
```

EXAMPLE 3 Using a Lint Repository With a Narrowed Manifest Set

Running a `pkglint` session with a lint repository and specifying a subset of manifests to check.

```
$ pkglint -c /space/othercache -l http://localhost:10000 \
-p '.*firefox.*'
```

EXAMPLE 4 Specifying a Build

Running a `pkglint` session against a given build in verbose mode.

EXAMPLE 4 Specifying a Build *(Continued)*

```
$ pkglint -c /space/cache -r http://localhost:10000 \
-l http://localhost:12000 -b 147 -v
```

EXAMPLE 5 Modifying a Configuration File

A configuration file with a new lint module, excluding some checks.

```
$ cat ~/.pkglintrc
[pkglint]

log_level = DEBUG
# log_level = INFO

pkglint.ext.mycheck = org.timf.mychecks
pkglint.ext.opensolaris = pkg.lint.opensolaris
pkglint.exclude: pkg.lint.opensolaris.OpenSolarisActionChecker
pkg.lint.pkglint.PkgActionChecker.unusual_perms pkg.lint.pkglint.PkgManifestChecker
pkg.lint.opensolaris.OpenSolarisManifestChecker
```

Exit Status

The following exit values are returned:

`0` Command succeeded.

`1` One or more lint checks emitted output.

`2` Invalid command line options were specified.

`99` An unanticipated exception occurred.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also

`pkg`(1), `pkg.depotd`(1m), `pkgsend`(1), `pkg`(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Name pkgmerge – Image Packaging System package merging utility

Synopsis

```
/usr/bin/pkgmerge [-n] -d dest_repo
    -s variant=value[,...],src_repo ...
    [pkg_fmri_pattern ...]
```

Description `pkgmerge` is a package publication tool for creating multi-variant packages. It does this by merging packages with identical names and versions (excluding time stamp), tagging actions that are unique in the versions being merged with the specified variant name and value for the given source, and then publishing the new packages to the target repository. Only the newest version of every package from each source is used.

If an action has the attribute `pkg.merge.blend` set to the name of the variant being merged, that action is copied to the other manifests prior to merging so that the action appears without any added variant tags in the final output. Note that the attribute `pkg.merge.blend` itself is removed from any actions in the output manifest. This attribute can be repeated with different values for multiple pass merges.

Non-identical actions that deliver to the same path in an input manifest result in `pkgmerge` exiting with an error.

Options The following options are supported:

`-d` *dest_repo*
: The file system path or URI of the target repository to publish the merged packages to. The target repository must already exist. New repositories can be created using `pkgrepo`(1).

`-n`
: Perform a trial run with no changes made to the target repository.

`-s` *variant=value*[,...],*src_repo*
: The variant name and value to use for packages from this source, followed by the file system path or URI of the source repository or package archive to retrieve packages from. Multiple variants can be specified separated by commas. The same variants must be named for all sources. This option can be specified multiple times.

`--help` or `-?`
: Displays a usage message.

Environment Variables The following environment variable is supported:

`TMPDIR` The absolute path of the directory where temporary data should be stored during program execution. If not set, the default is to store temporary data in `/var/tmp`.

Examples **EXAMPLE 1** Specify Variant Name and Value

Tag each package found in the specified source with the given variant name and value specified for the source it was retrieved from:

```
$ pkgmerge -s arch=sparc,http://src.example.com \
-d http://dest.example.com
```

EXAMPLE 1 Specify Variant Name and Value *(Continued)*

Sample package:

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
dir group=sys mode=0755 owner=root path=usr
```

Sample package after operation:

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
set name=variant.arch value=sparc
dir group=sys mode=0755 owner=root path=usr
```

EXAMPLE 2 Merge and Publish Packages

Merge the newest version of each package from the given sources and publish the new packages to the target repository:

```
$ pkgmerge -s arch=sparc,http://src1.example.com \
-s arch=i386,http://src2.example.com \
-d /path/to/target/repository
```

Sample package from source 1 (SPARC):

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T121410Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

Sample package from source 2 (i386):

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

Merged package:

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
set name=variant.arch value=sparc value=i386
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=sparc
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=i386
dir group=sys mode=0755 owner=root path=usr
```

EXAMPLE 3 Merge Debug and Non-Debug Packages for i386 and SPARC Systems

Merge the newest version of each package in a set of debug and non-debug repositories for i386 and SPARC systems:

```
$ pkgmerge -s arch=sparc,debug=false,/repo/sparc-nondebug \
-s arch=sparc,debug=true,/repo/sparc-debug \
-s arch=i386,debug=false,/repo/i386-nondebug \
```

EXAMPLE 3 Merge Debug and Non-Debug Packages for i386 and SPARC Systems *(Continued)*

```
-s arch=i386,debug=true,/repo/i386-debug \
-d /path/to/target/repository
```

Sample package from source 1 (SPARC non-debug):

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T121410Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

Sample package from source 2 (SPARC debug):

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T121411Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

Sample package from source 3 (i386 non-debug):

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

Sample package from source 4 (i386 debug):

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163428Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

Merged package:

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163428Z
set name=variant.arch value=sparc value=i386
set name=variant.debug value=false value=true
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=sparc variant.debug=false
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=sparc variant.debug=true
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=i386 variant.debug=false
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=i386 variant.debug=true
dir group=sys mode=0755 owner=root path=usr
```

EXAMPLE 4 Merge Using pkg.merge.blend

Merge packages for two architectures that do not collide, using the `pkg.merge.blend` attribute.

```
$ pkgmerge -s arch=sparc,http://src1/example.com \
-s arch=i386,http://src2.example.com \
-d /path/to/target/repository
```

Sample package from source 1 (SPARC):

EXAMPLE 4 Merge Using pkg.merge.blend *(Continued)*

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T121410Z
file 1d5eac1aab628317f9c088d21e4afda9c754bb76 mode=0555 owner=root \
    group=bin path=usr/bin/sparc/foo pkg.merge.blend=arch
file d285ada5f3cae14ea00e97a8d99bd3e357caadc0 mode=0555 owner=root \
    group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

Sample package from source 2 (i386):

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
file a285ada5f3cae14ea00e97a8d99bd3e357cb0dca mode=0555 owner=root \
    group=bin path=usr/bin/i386/foo pkg.merge.blend=arch
file d285ada5f3cae14ea00e97a8d99bd3e357caadc0 mode=0555 owner=root \
    group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

Merged package:

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
set name=variant.arch value=sparc value=i386
file d285ada5f3cae14ea00e97a8d99bd3e357caadc0 mode=0555 owner=root \
    group=bin path=usr/bin/foo
file a285ada5f3cae14ea00e97a8d99bd3e357cb0dca mode=0555 owner=root \
    group=bin path=usr/bin/i386/foo
file 1d5eac1aab628317f9c088d21e4afda9c754bb76 mode=0555 owner=root \
    group=bin path=usr/bin/sparc/foo
dir group=sys mode=0755 owner=root path=usr
```

Exit Status

The following exit values are returned:

`0` Command succeeded.

`1` An error occurred.

`2` Invalid command line options were specified.

`99` An unanticipated exception occurred.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also pkgrepo(1), pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

Name pkgmogrify – Image Packaging System manifest transmogrifier

Synopsis

```
/usr/bin/pkgmogrify [-vi] [-I includedir ...]
    [-D macro=value ...] [-O outputfile]
    [-P printfile] [inputfile ...]
```

Description `pkgmogrify` provides for programmatic editing of package manifests to simplify the typical transformations needed when automating software builds and package publication.

`pkgmogrify` provides the following:

- Macro replacement to facilitate sharing of a single manifest across various architectures and platforms.
- Inclusion of other manifests or manifest fragments such as standard components and transforms.
- Transformation of package actions, including the modification, deletion, or addition of action attributes.

Options The following options are supported:

`-D` *name=value*
Defines *name* as a macro, with the value *value*. Macros appear in the input file as `$(macro)`. Substitution is repeated until no more translations are found. Common idioms include:

- Elimination of lines in a manifest on other architectures by using an architecture-specific tag at the beginning of the line:

  ```
  $(sparc_ONLY)file ...
  ```

  When processing the SPARC architecture, this macro would be set to the empty string. When processing other architectures, this macro would be set to # on the command line, thus eliminating this action from the manifest on the current architecture.

- Specifying platform-specific portions of path names, such as the name of the 64-bit architecture directory for executables and libraries:

  ```
  file NOHASH path=usr/bin/$(ARCH64)/cputrack ...
  ```

  These macros should be set to the desired value on the command line. There are no predefined macro values.

`-I` *include_directory*
Adds the specified directory to the search path for both files specified on the command line and embedded include directives.

`-O` *outputfile*
Write manifest output to the specified file. The file is not written if an error occurs or if a transform directive causes an abort operation. By default, manifest output is written to `stdout`.

`-P` *printfile*
Write output resulting from transform directive print operations to the specified file. The file is not written if an error occurs or if a transform directive causes an abort operation. By default, print output is written to `stdout`.

`-i`
Ignore include directives in files. Only files specified on the command line (or `stdin`) are processed.

`-v`
Write comments into the output manifest showing the effect of transforms. This information can aid in debugging.

`--help` or `-?`
Displays a usage message.

Embedded Directives

Two types of directives are supported in manifest files: include directives and transform directives.

Include directives are of the form:

`<include` *file*`>`

This directive causes `pkgmogrify` to search for a file named `file` first in the current directory and then in the directories specified with the `-I` option. If found, the contents of the file are inserted into the manifest at the point at which the directive is encountered. If not found, `pkgmogrify` exits with an error.

Transform directives are of the form:

`<transform` *matching-criteria* `->` *operation*`>`

These directives are accumulated until all inputs have been read into memory, and then applied to the actions in the order in which they were encountered.

Matching criteria are of the form:

`[`*action-type* `... ] [`*attribute*`=<`*value-regexp*`> ...]`

One of the *action-types* specified must match. All of the *attributes* specified must match. The regular expression syntax used is that of Python. For more information about Python regular expression syntax, use the command `pydoc re` or see more complete documentation at `http://docs.python.org/dev/howto/regex.html`. The regular expression is anchored at the beginning but not at the end. Therefore, a regular expression matching files by their extensions must include a leading `.*` and should include a trailing `$`.

Multiple criteria can be specified, separated by spaces.

The following operations are available:

`add` Add a value to an attribute. This operation takes two arguments. The first argument is the name of the attribute, and the second is the value.

`default` Set the value of an attribute if it doesn't already exist. This operation takes the same two arguments as the `add` operation.

`delete` Remove attribute values. This operation takes two arguments. The first argument is the name of the attribute. The second argument is a regular expression to match the attribute values deleted. Unlike the regular expression used to match an action, this expression is unanchored.

`drop` Discards this action.

`edit` Modifies an attribute of the action. This operation takes three arguments. The first argument is the name of the attribute, and the second is a regular expression matching the attribute value. The third argument is the replacement string substituted for the portion of the value matched by the regular expression. Unlike the regular expression used to match an action, this expression is unanchored. Normal regular expression backreferences, of the form `\1`, `\2`, and so on, are available in the replacement string if groups are defined in the regular expression.

`emit` Emit a line to the manifest output stream. This must be a valid action string, empty (resulting in a blank line), or a comment (a # followed by arbitrary text).

`exit` Terminate manifest processing. No manifest is output and no `print` operations are applied. If one argument is given, it must be an integer, and it is used as the exit code. The default is 0. If two arguments are given, the first is the exit code, and the second is a message to be printed to `stderr`.

`print` Print a message to the output file specified with `-P`.

`set` Set the value of an attribute. This operation takes the same two arguments as the `add` operation.

All operations except for `delete` and `drop` take (possibly optional) arguments whose contents go to the output stream. These strings can contain three different kinds of special tokens which allow the output to contain information that is not based on a fixed transformation of each action.

The first kind of substitution allows the operation to refer to the values of attributes of the current action by putting the name of the attribute inside parentheses following a percent sign. For example, `%(alias)` refers to the value of the action's `alias` attribute.

Several synthetic attributes exist. Two are unique to `pkgmogrify`:

- `pkg.manifest.filename` refers to the name of the file in which the action was found.
- `pkg.manifest.lineno` refers to the line on which the action was found.

Three synthetic attributes are similar to ones used in pkg(1):

- action.hash refers to the hash value of the action if the action carries a payload. For actions with payloads, the set operation can change the hash of the action by operating on the action.hash attribute.
- action.key refers to the value of the key attribute.
- action.name refers to the name of the action type.

If the attribute whose value is requested does not exist, pkgmogrify exits with an error. To prevent exiting with an error, follow the attribute name with ;notfound= and a value to substitute in place of the attribute value. For example, %(alias;notfound='no alias') prints the value of the attribute alias if it exists, and prints no alias otherwise.

If the attribute whose value is requested is multi-valued, each value is printed, separated by spaces. Similarly to the notfound token, the tokens prefix, suffix, and sep can be used to change this behavior. The string denoted by prefix is prepended to each value, the string denoted by suffix is appended to each value, and sep is placed in between the suffix of one value and the prefix of the next.

Similarly to action attributes, pkgmogrify directives can reference package attributes using braces instead of parentheses: %{pkg.fmri}. At the point at which the transform directive is applied, the attribute must have been defined in a set action, or it is treated as notfound, described above. When the processing reaches the end of the manifest file describing the package, the attributes are cleared for the next package.

It is useful not only to reference package attributes as if they were action attributes, but also to match on them, and even temporarily modify them. Therefore a synthetic action name, pkg, is available (only in the context of pkgmogrify) for use in these situations.

When pkgmogrify finishes reading a manifest specified on the command line and that manifest defined a pkg.fmri package attribute, pkgmogrify creates this synthetic pkg action, whose attributes are the package's attributes. A <transform> directive can then match on this action, just like any other action type.

Operations on a pkg action are special in that they take place only in memory and do not directly affect the emitted manifest. For instance, trying to set an attribute on a pkg action via the add, default, or set operations does not result in a set action being added to the manifest, though it will be available for other <transform> directives to match on. Attempting to emit a pkg action causes an error. To add a package attribute, emit a set action instead.

The third kind of substitution is a backreference. This substitution is not like the ones usable in the edit operation, but is a reference to groups listed in the transformation match on the left-hand side of the ->. These are indicated by %<1>, %<2>, and so on, in the order seen in the matching criteria.

The order of processing is as follows:

1. Lines are read from input files.
2. Macros are applied.
3. `<include ...>` and `<transform>` directives are processed, causing additional files to be found and read.
4. Once all input has been accumulated, each line in the input is converted into actions and all transforms applied.
5. Once processing is complete and successful, the output is written.

Examples

EXAMPLE 1 Add Tags To SMF Manifests

Add tags to Service Management Facility (SMF) manifests so they get imported when the package is installed on a live system.

```
<transform file path=(var|lib)/svc/manifest/.*\.xml -> \
    add restart_fmri svc:/system/manifest-import:default>
```

EXAMPLE 2 Move Files

Move files from `usr/sfw/bin` to `usr/bin`.

```
<transform file -> edit path usr/sfw/bin usr/bin>
```

EXAMPLE 3 Specify Reboot Needed

Add `reboot-needed` tags to files under `/kernel` that are not `.conf` files. Note that this example leverages how transforms are applied to each action in the order seen in the input files.

```
<transform file path=kernel/.* -> set reboot-needed true>
<transform file path=kernel/.*\.conf -> delete reboot-needed .*>
```

This can also be done in a single transform rule with regular expressions.

EXAMPLE 4 Convert FMRI Attribute To Depend Action

Convert the package attribute `pkg.fmri` into a `depend` action to become part of an incorporation.

```
<transform set name=pkg.fmri -> \
    emit depend type=incorporate fmri=%(value)>
<transform set name=pkg.fmri -> drop>
```

EXAMPLE 5 Print a List of Bug Numbers

Print a comma-separated list of quoted and prefixed bug numbers.

```
set name=bugs value=12345 value=54321 value=13579 value=97531
<transform set name=bugs -> \
    print %(value;sep=",";prefix="bug='";suffix="'")>
```

EXAMPLE 6 Allow For Missing Attributes

Safely print a message even when an attribute is missing.

```
<transform driver -> print Found aliases: %(alias;notfound=<none>)>
```

EXAMPLE 7 Set Default Values

Set default owner, group, and permission values.

```
<transform file dir -> default owner root>
<transform file dir -> default group bin>
<transform file -> default mode 0444>
<transform dir -> default mode 0755>
```

EXAMPLE 8 Add Dependencies To Packages That Are Not Marked Obsolete

For any package that is not marked obsolete, add a dependency on the incorporation for the consolidation that delivers the package. This set of transforms must occur after the manifest has been read in, or the dependency will always be emitted. Because modifying a `pkg` action has no permanent effect, there is no need to clean up attributes matching `pkg.obsolete=false`.

```
<transform pkg -> default pkg.obsolete false>
<transform pkg pkg.obsolete=false -> emit depend \
    fmri=consolidation/$(CONS)/$(CONS)-incorporation type=require>
```

EXAMPLE 9 Exit and Print a Message When an Error Is Found

Error out with a message when an obsolete attribute is found in a manifest.

```
<transform file dir link hardlink opensolaris.zone=.* -> \
    exit 1 The opensolaris.zone attribute is obsolete.>
```

EXAMPLE 10 Set the Appropriate Locale Facet

Set the locale facet appropriate for the path name under consideration.

```
<transform dir file link hardlink path=.*/locale/([^/]+).* -> \
    default facet.locale.%<1> true>
```

Exit Status

The following exit values are returned:

`0` Everything worked.

`1` Something bad but anticipated happened.

`2` Invalid command line options were specified.

`99` Unexpected processing error.

Files `/usr/share/pkg/transforms` This directory contains files with useful transforms to set facets, actuators, and other attributes.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also `pkg`(1), `pkg`(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Name pkgrecv – Image Packaging System content retrieval utility

Synopsis

```
/usr/bin/pkgrecv [-s src_uri] [-a] [-d (path|dest_uri)]
    [-c cache_dir] [-kr] [-m match] [-n] [--raw]
    [--key keyfile --cert certfile] (fmri|pattern) ...

/usr/bin/pkgrecv [-s src_uri] --newest
```

Description `pkgrecv` allows the user to retrieve packages from a `pkg`(5) repository or package archive. `pkgrecv` can also optionally republish the retrieved packages to a different package repository or archive them. By default, packages are retrieved in package repository format suitable for use with `pkg`(1), `pkg.depotd`(1M), and package publication tools.

After a `pkgrecv` operation, run `pkgrepo refresh` or `pkgrepo rebuild` on the repository to build search indexes.

Options The following options are supported:

`-a` Store the retrieved package data in a `pkg`(5) archive at the location specified by `-d`. The file cannot already exist. This option can be used only with file system based destinations. Although not required, using a file extension of `.p5p` (for example, `archive.p5p`) is strongly suggested. This option cannot be combined with `--raw`.

`-c` *cache_dir* The path to a directory that will be used to cache downloaded content. If this directory is not supplied, the client automatically selects a cache directory. In the case where a download is interrupted, and a cache directory was automatically chosen, use this option to resume the download. See the "Environment Variables" section below for details about how to set the location used for temporary data storage.

`-d` *path_or_uri* The file system path or URI of the target to republish packages to. If `-a` is specified, the target is a new package archive that cannot already exist. Otherwise, the target must be a package repository that already exists. New repositories can be created using `pkgrepo`(1).

`-h` Display a usage message.

`-k` Keep the retrieved package content compressed. This option is ignored when republishing. Compressed package content should not be used with `pkgsend`(1).

`-m` *match* Controls matching behavior using the following values:

- `all-timestamps` Includes all matching timestamps, not just the latest (implies `all-versions`).
- `all-versions` Includes all matching versions, not just the latest.

`-n` Perform a trial run with no changes made.

`-r`	Recursively retrieves all dependencies for the provided list of packages.
`-s` *src_repo_uri*	A URI representing the location of a `pkg`(5) repository or package archive from which to receive package data.
`--cert` *file*	Specify a client SSL certificate file to use for package retrieval from an HTTPS repository.
`--key` *file*	Specify a client SSL key file to use for package retrieval from an HTTPS repository.
`--newest`	List the most recent versions of the packages available from the specified repository and exit. (All other options except `-s` are ignored.)
`--raw`	Retrieve and store the raw package data in a set of directory structures by stem and version at the location specified by `-d`. This option can be used only with file system based destinations. This package data can be used to conveniently modify and republish packages, perhaps by correcting file contents or providing additional package metadata. This option cannot be combined with `-a`.

Examples

EXAMPLE 1 List Newest Packages

List the newest packages available from the repository on the system named `test`.

```
$ pkgrecv -s http://test --newest
pkg://solaris/system/library/c++-runtime@0.5.11,5.11-0.174.0.0.0.0.0:20110921T190358Z
pkg://solaris/system/library/freetype-2@2.4.8,5.11-0.175.1.0.0.7.1234:20120109T215840Z
pkg://solaris/system/library/math@0.5.11,5.11-0.174.0.0.0.0.0:20110921T190432Z
```

EXAMPLE 2 Retrieve Raw Package Data

Receive the `c++-runtime` package from Example 1 in a format suitable for use with `pkgsend publish`.

```
$ pkgrecv -s http://test \
-d /local/repo --raw \
c++-runtime@0.5.11,5.11-0.174.0.0.0.0.0:20110921T190358Z
Processing packages for publisher solaris ...
Retrieving and evaluating 1 package(s)...
PROCESS                                       ITEMS     GET (MB)    SEND (MB)
Completed                                       1/1      3.5/3.5      0.0/0.0
$ ls /local/repo
pkg5.repository  publisher  system%2Flibrary%2Fc%2B%2B-runtime
```

EXAMPLE 3 Retrieve Dependencies From a System

Receive the package `editor/vim` and all of its dependencies from the system named `test`.

```
$ pkgrecv -s http://test -d /local/repo -r editor/vim
```

EXAMPLE 4 Retrieve All Versions

Receive all versions of the package `editor/vim` from the system named `test`.

```
$ pkgrecv -s http://test -d /local/repo -m all-versions editor/vim
Processing packages for publisher solaris ...
Retrieving and evaluating 2 package(s)...
PROCESS                                       ITEMS     GET (MB)    SEND(MB)
Completed                                       2/2    16.7/16.7    44.9/44.9
```

EXAMPLE 5 Retrieve All Versions and Republish Remotely

Receive all versions of the package `library/zlib` from the system named `test` and republish it to a remote repository on the system named `remote`.

```
$ pkgrecv -s http://test -d http://remote:10000 -m all-versions library/zlib
```

EXAMPLE 6 Retrieve Dependencies From a Repository

Receive the package `editor/gnu-emacs` and all of its dependencies from the repository located at `/export/repo`.

```
$ pkgrecv -s /export/repo -d /local/repo -r editor/gnu-emacs
```

EXAMPLE 7 Retrieve Additional Packages

Receive all packages that do not already exist from the repository located at `http://example.com:10000`.

```
$ pkgrecv -s http://example.com:10000 -d /my/pkg/repo '*'
```

EXAMPLE 8 Create a Package Archive

Create a package archive containing the package `editor/gnu-emacs` and all of its dependencies from the repository located at `http://example.com:10000`.

```
$ pkgrecv -s http://example.com:10000 -d /my/emacs.p5p -a -r editor/gnu-emacs
```

EXAMPLE 9 Copy Packages From an Archive to a Repository

Copy all of the packages in a package archive to an existing repository located at `/export/repo`.

```
$ pkgrecv -s /my/archive.p5p -d /export/repo '*'
```

Environment Variables

The following environment variables are supported:

`PKG_DEST` The path of a directory to save the retrieved package to, or the file system path or URI of a repository or package archive where the packages will be copied.

`PKG_SRC` A URI or file system path representing the location of a `pkg`(5) repository or package archive from which to retrieve packages.

`TMPDIR` The absolute path of the directory where temporary data should be stored during program execution. If not set, the default is to store temporary data in `/var/tmp`.

Exit Status The following exit values are returned:

`0` Command succeeded.

`1` An error occurred.

`2` Invalid command line options were specified.

`3` Multiple operations were requested, but only some of them succeeded.

`99` An unanticipated exception occurred.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also `pkgrepo`(1), `pkgsend`(1), `pkg`(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Name pkgrepo – Image Packaging System repository management utility

Synopsis

```
/usr/bin/pkgrepo create [--version ver] uri_or_path

/usr/bin/pkgrepo add-publisher -s repo_uri_or_path publisher ...

/usr/bin/pkgrepo get [-F format] [-p publisher ...]
    -s repo_uri_or_path [section/property ...]

/usr/bin/pkgrepo info [-F format] [-H] [-p publisher ...]
    -s repo_uri_or_path

/usr/bin/pkgrepo list [-F format] [-H] [-p publisher ...]
    -s repo_uri_or_path [pkg_fmri_pattern ...]

/usr/bin/pkgrepo rebuild [-p publisher ...]
    -s repo_uri_or_path [--no-catalog] [--no-index]

/usr/bin/pkgrepo refresh [-p publisher ...]
    -s repo_uri_or_path [--no-catalog] [--no-index]

/usr/bin/pkgrepo remove [-n] [-p publisher ...]
    -s repo_uri_or_path pkg_fmri_pattern ...

/usr/bin/pkgrepo set [-p publisher] -s repo_uri_or_path
    section/property=[value] ... or
    section/property=([value]) ...

/usr/bin/pkgrepo help

/usr/bin/pkgrepo version
```

Description `pkgrepo` provides the ability to create and manage `pkg`(5) package repositories. Package repositories are a predefined set of directories and files that permit the storage and retrieval of package data by `pkg`(1) and publication clients such as `pkgsend`(1) or `pkgrecv`(1). In addition, when network-based access to a package repository is needed, `pkg.depotd`(1m) can provide clients access to the repository to store and/or retrieve package data.

Options The following options are supported:

`--help` or `-?` Displays a usage message.

Sub-commands The following subcommands are supported:

`create [--version` *ver*`]` *uri_or_path*

Creates a `pkg`(5) repository at the specified location.

This subcommand can be used only with file system based repositories.

With `--version`, create a repository in a format compatible with the specified version. By default, version 4 repositories are created. Supported versions are:

3 Supports storage of packages for a single publisher, catalog version 1, and search version 1.

4 Supports storage of packages for multiple publishers, catalog version 1, and search version 1.

`add-publisher -s` *repo_uri_or_path publisher* `...`
Adds the specified publishers to the repository. The new publishers have no packages or content.

This subcommand can be used only with version 4 file system based repositories.

`get [-F` *format*`] [-p` *publisher* `...] -s` *repo_uri_or_path* `[`*section/property* `...]`
Displays the property information for the repository or its publishers.

By default, each property and its value are printed on separate lines. Empty ASCII string values are represented by a pair of double quotation marks (""). The following Bourne shell metacharacters, and newline, space, and tab, in ASCII string values must be escaped by backslash characters (\):

```
; & ( ) | ^ < > \ " ’ ‘
```

See the “Examples” section.

For a list of possible properties and the purpose and value of each property, see the `set` subcommand below.

With `-F`, specify an alternative output format. The value of *format* can be `tsv` (Tab Separated Values), `json` (JavaScript Object Notation as a single line), or `json-formatted` (JavaScript Object Notation, formatted for readability).

With `-H`, omit the headers from the listing.

With `-p`, display the property information for the given publisher. The special value `all` displays the properties for all publishers. This option can be specified multiple times.

With `-s`, operate on the repository located at the given URI or file system path.

`info [-F` *format*`] [-H] [-p` *publisher* `...] -s` *repo_uri_or_path*
Displays a listing of the package publishers known by the repository. The listing includes the number of packages for each publisher, when the publisher's package data was last updated, and the status of the publisher's package data (such as whether it is currently being processed).

With `-F`, specify an alternative output format. The value of *format* can be `tsv` (Tab Separated Values), `json` (JavaScript Object Notation as a single line), or `json-formatted` (JavaScript Object Notation, formatted for readability).

With `-H`, omit the headers from the listing.

With `-p`, only display the data for the given publisher. If not provided, the data for all publishers is displayed. This option can be specified multiple times.

With `-s`, operate on the repository located at the given URI or file system path.

`list [-F` *format*`] [-H] [-p` *publisher* `...] -s` *repo_uri_or_path* `[`*pkg_fmri_pattern* `...]`
List the packages in the *repo_uri_or_path* repository that match the specified *pkg_fmri_pattern* patterns. If no patterns are specified, all packages in the repository are listed.

In the default output, the first column contains the name of the publisher of the package. The second column contains the name of the package. The third column is a flag that shows the status of the package. A value of `o` in the status column indicates the package is obsolete. A value of `r` in the status column indicates the package has been renamed, which is a form of obsoletion. The fourth column contains the release and branch versions of the package. See `pkg`(5) for information about release and branch versions.

With `-F`, specify an alternative output format. The value of *format* can be `tsv` (Tab Separated Values), `json` (JavaScript Object Notation as a single line), or `json-formatted` (JavaScript Object Notation, formatted for readability).

With `-H`, omit the headers from the listing.

With `-p`, only display the packages for the given publisher. If not provided, the packages for all publishers are displayed. This option can be specified multiple times.

With `-s`, operate on the repository located at the given URI or file system path.

`rebuild [-p` *publisher* `...] -s` *repo_uri_or_path* `[--no-catalog] [--no-index]`
Discards all catalog, search, and other cached information found in the repository, and then recreates it based on the current contents of the repository.

With `-p`, perform the operation only for the given publisher. If not provided, or if the special value `all` is specified, the operation is performed for all publishers. This option can be specified multiple times.

With `-s`, operate on the repository located at the given URI or file system path.

With `--no-catalog`, do not rebuild package data.

With `--no-index`, do not rebuild search indexes.

`refresh [-p` *publisher* `...] -s` *repo_uri_or_path* `[--no-catalog] [--no-index]`
Catalogs any new packages found in the repository and updates all search indexes. This is intended for use with deferred publication (`--no-catalog` or `--no-index` options of `pkgsend`).

With `-p`, perform the operation only for the given publisher. If not provided, or if the special value `all` is specified, the operation is performed for all publishers. This option can be specified multiple times.

With `-s`, operate on the repository located at the given URI or file system path.

With `--no-catalog`, do not add any new packages.

With `--no-index`, do not update search indexes.

`remove [-n] [-p` *publisher* `...] -s` *repo_uri_or_path pkg_fmri_pattern* `...`
Removes the packages matching the specified patterns from the repository, including any files they reference that are not in use by any other package.

Note – All search index data for related publishers is removed.

This subcommand can be used only with file system based repositories.

Caution – This operation is not reversible and should not be used while other clients are accessing the repository since it might cause them to fail during retrieval operations.

With `-n`, perform a trial run of the operation with no package changes made. A list of the packages to be removed is displayed before exiting.

With `-p`, only remove matching packages for the given publisher. If not provided, any matching packages are removed for all publishers. This option can be specified multiple times.

With `-s`, operate on the repository located at the given URI or file system path.

`set [-p` *publisher*`] -s` *repo_uri_or_path section/property*`=[`*value*`] ... or` *section/property*`=([`*value*`]) ...`
Sets the value of the specified properties for the repository or publisher.

This subcommand can be used only with file system based repositories.

With `-p`, only set property data for the given publisher. If the publisher does not already exist, it is added. The special value `all` can be used to set the property for all publishers.

With `-s`, operate on the repository located at the given URI or file system path.

Properties and values can be specified using one of the following forms:

section/*property*=	Clear the property value.
section/*property*=*value*	Replace the property value with the given value.
section/*property*=(*value1 value2 valueN*)	Replace the property value with the list of values.

For repository versions 3 and 4, the following properties can be set for the repository:

`publisher/prefix`	A string that represents the name of the default publisher. The first character must be a-z, A-Z, or 0-9. The remainder of the string can only contain the characters 0-9, -, ., a-z, and A-Z. This value indicates the publisher that should be used when more than one publisher's packages are present, or when packages are published

to the repository and a publisher is not specified.

For repository versions 3 and 4, the following properties can be set for individual publishers in the repository:

`publisher/alias`	A string that represents the default alias that clients should use when adding a publisher using the repository's configuration data. The first character must be a-z, A-Z, or 0-9. The remainder of the string can only contain the characters 0-9, -, ., a-z, and A-Z.
`repository/collection_type`	Can have the value `core` or `supplemental`, indicating the type of packages offered in this repository. The `core` type indicates that the repository contains all of the dependencies declared by packages in the repository. The `core` type is primarily used for operating system repositories. The `supplemental` type indicates that the repository contains packages that rely on or are intended to be used with packages located in another repository.
`repository/description`	A paragraph of plain text that describes the purpose and contents of the repository.
`repository/detailed_url`	A URI that represents the location of a document (such as a web page) that provides additional information about the repository.
`repository/legal_uris`	A list of locations (URIs) for documents that provide additional legal information about the repository.
`repository/mirrors`	A list of locations (URIs) of repositories that contain a copy of the repository's package content but not the package metadata.
`repository/name`	A plain text string that contains the name of the repository.
`repository/origins`	A list of locations (URIs) of repositories that contain a complete copy of the repository's package metadata and content.
`repository/refresh_seconds`	An integer value that represents the number of seconds clients should wait before checking the repository for updated package data after each update check.

`repository/registration_uri`	A URI that represents the location of a resource that must be used to obtain credentials for access to the repository. A registration web page is one example.
repository/related_uris	A list of locations (URIs) of repositories that contain packages that users might be interested in.

Properties not documented here, but listed in the output of the `get` subcommand, are reserved for internal use and should not be set.

`version`
Displays a unique string that identifies the version of the `pkg`(5) system. The values produced by the `version` operation are not sortable and are not safe for comparison beyond equality.

Examples

EXAMPLE 1 Create a Package Repository

```
$ pkgrepo create /my/repository
```

EXAMPLE 2 Display Information

Display a summary of publishers and the number of packages in a repository.

```
$ pkgrepo info -s /my/repository
PUBLISHER   PACKAGES STATUS UPDATED
example.com 5        online 2011-07-22T18:09:09.769106Z
$ pkgrepo info -s http://pkg.oracle.com/solaris/release/
PUBLISHER PACKAGES STATUS UPDATED
solaris   3941     online 2010-11-12T19:24:25.967246Z
```

EXAMPLE 3 Rebuild Catalogs and Search Data

Rebuild the repository's catalogs and search data.

```
$ pkgrepo rebuild -s /my/repository
```

EXAMPLE 4 Refresh Catalogs and Search Data

Refresh the repository's catalogs and search data.

```
$ pkgrepo refresh -s /my/repository
$ pkgrepo refresh -s http://example.com/repository
```

EXAMPLE 5 Display All Repository Properties

```
$ pkgrepo get -s /my/repository
SECTION    PROPERTY VALUE
publisher  prefix   ""
repository version  4
$ pkgrepo get -s http://pkg.oracle.com/solaris/release/
SECTION    PROPERTY VALUE
publisher  prefix   solaris
```

EXAMPLE 5 Display All Repository Properties *(Continued)*

```
repository version  4
```

EXAMPLE 6 Display All Publisher Properties

```
$ pkgrepo get -s http://pkg.oracle.com/solaris/release/ -p all
PUBLISHER SECTION    PROPERTY         VALUE
solaris   publisher  alias
solaris   publisher  prefix           solaris
solaris   repository collection-type  core
solaris   repository description      This\ repository\ serves\ the\ Oracle\
Solaris\ 11\ Package\ repository.
solaris   repository legal-uris       ()
solaris   repository mirrors          (http://pkg-cdn1.oracle.com/solaris.release/)
solaris   repository name             Oracle\ Solaris\ 11\ Package\ Repository
solaris   repository origins          ()
solaris   repository refresh-seconds
solaris   repository registration-uri ""
solaris   repository related-uris     ()
```

EXAMPLE 7 Set the Default Publisher

```
$ pkgrepo set -s /my/repository publisher/prefix=example.com
```

EXAMPLE 8 Set a Publisher Property

```
$ pkgrepo set -s /my/repository -p example.com \
repository/origins=http://example.com/repository
```

EXAMPLE 9 Add a New Publisher To the Repository

```
$ pkgrepo add-publisher -s /my/repository example.com
```

Exit Status

The following exit values are returned:

`0` Command succeeded.

`1` An error occurred.

`2` Invalid command line options were specified.

`3` Multiple operations were requested, but only some of them succeeded.

`4` No changes were made, nothing to do.

`99` An unanticipated exception occurred.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	package/pkg
Interface Stability	Uncommitted

See Also pkg(1), pkgrecv(1), pkgsend(1), pkg.depotd(1m), pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

Name pkgsend – Image Packaging System publication client

Synopsis

```
/usr/bin/pkgsend [options] command [cmd_options] [operands]

/usr/bin/pkgsend generate [-T pattern] [--target file]
    source ...

/usr/bin/pkgsend publish [-b bundle ...] [-d source ...]
    [-s repo_uri_or_path] [-T pattern] [--no-catalog]
    [manifest ...]
```

Description `pkgsend` enables the publication of new packages and new package versions to an image packaging repository using package manifests. To create or manage repositories, see `pkgrepo`(1). To create package archives from packages in an existing repository, see `pkgrecv`(1). For more information about package manifests, see `pkg`(5).

After a `pkgsend` operation, run `pkgrepo refresh` or `pkgrepo rebuild` on the repository to build search indexes.

Options The following options are supported:

`--help` or `-?` Displays a usage message.

Sub-commands The following subcommands are supported:

`generate [-T` *pattern*`] [--target` *file*`]` *source* `...`
Read each *source* (such as an SVR4 package, a directory, or a `tar` file) and emit the manifest that describes the *source* to `stdout`. In the output manifest, `file` and `dir` actions have owner set to `root` and group set to `bin`.

The output manifest can then be annotated, have dependencies added or analyzed using `pkgdepend`(1), and have its correctness verified using `pkglint`(1) before being passed to the `publish` subcommand.

The following are supported sources:

- Filesystem format SVR4 packages
- Datastream format SVR4 packages
- `tar` files
- Directories

If the base name of files in the source match the patterns specified with `-T`, the timestamp of the file is added to the action for that file. The *pattern* uses shell matching rules:

*	Matches everything.
?	Matches any single character.
[*seq*]	Matches any character in *seq*.
![*seq*]	Matches any character not in *seq*.

When the specified source is a directory, there is no clear way to distinguish a `file` action from a `hardlink` action when there are multiple path names for a single inode. Normally, the first one found in the file system walk is treated as a file and the rest as hardlinks. This can be arbitrary, depending on the implementation of the file system. To specify which path names should be treated as files, pass each path name as an argument to the `--target` option. This option has no effect on other types of sources because they are capable of expressing which path names are files and which are hardlinks.

When SVR4 packages are provided as a source, `pkgsend` checks that no files with class action scripts are present and no preinstall, postinstall, preremove, or postremove scripts are present. An exception is made for any SMF manifests installed with the `manifest` class. `BASEDIR` is removed from all relocatable paths.

The SVR4 `DESC` parameter is converted to a `pkg.description` value. The SVR4 `NAME` parameter is converted to a `pkg.summary` value.

`publish [-b` *bundle* `...] [-d` *source* `...] [-s` *repo_uri_or_path*`] [-T` *pattern*`] [--no-catalog] [`*manifest* `...]`

Publishes a package using the specified package manifests to the target package repository, retrieving files for the package from the provided sources. If multiple manifests are specified, they are joined in the order provided. If a manifest is not specified, the manifest is read from `stdin`.

With `-b`, add the specified bundle to the list of sources to search when looking for files in the manifest. Bundles are sources such as tar files and SVR4 packages. If this option is specified multiple times, sources are searched in the order they appear on the command line. If both `-b` and `-d` are specified, `-d` sources are searched first. For a description of supported bundles and how they are used, refer to the `generate` subcommand above.

With `-d`, add the specified directory to the list of sources to search when looking for files in the manifest. If this option is specified multiple times, sources are searched in the order they appear on the command line. For a description of supported sources and how they are used, refer to the `generate` subcommand above.

With `-s`, publish the package to the repository located at the given URI or file system path. See the “Notes” section below for more information about restrictions and suggestions for publication. See also the “Environment Variables” section.

With `--no-catalog`, do not add the package to the publisher's catalog. This option is recommended whenever multiple packages are being published at one time as updates to publisher catalogs must be performed serially. Once publication is complete, the `refresh` subcommand of `pkgrepo`(1) can be used to add the new packages to the respective publisher catalogs.

For all other options, refer to the `generate` subcommand above for usage and their effects.

Environment Variables

`PKG_REPO` The path or URI of the destination repository.

Examples

EXAMPLE 1 Generate and Publish a Package

Create a package using `pkgsend generate` and publish it.

```
$ pkgsend generate /path/to/proto > /path/to/manifests/foo.p5m
```

Add the package FMRI for the `example.com` publisher to the beginning of `foo.p5m`.

```
set name=pkg.fmri value=pkg://example.com/foo@1.0
```

The resulting manifest should look like this:

```
set name=pkg.fmri value=pkg://example.com/foo@1.0
dir group=sys mode=0755 owner=root path=usr
dir group=bin mode=0755 owner=root path=usr/bin
file usr/bin/foo group=bin mode=0555 owner=root path=usr/bin/foo
```

```
$ pkgsend publish -s http://example.com:10000 -d /path/to/proto \
/path/to/manifests/foo.p5m
```

EXAMPLE 2 Create and Publish a Trivial Package

Create a manifest for publisher `example.com` containing the following lines:

```
set name=pkg.fmri value=pkg://example.com/foo@1.0-1
file /exdir/foo mode=0555 owner=root group=bin path=/usr/bin/foo
```

Publish the package:

```
$ pkgsend publish -s http://example.com:10000 -d /exdir
```

EXAMPLE 3 Use a Preexisting Manifest

Publish a package using file system based publication and a preexisting manifest.

```
$ pkgsend publish -s /tmp/example_repo -d /tmp/pkg_files \
/tmp/pkg_manifest
```

Exit Status

The following exit values are returned:

`0` Command succeeded.

`1` An error occurred.

`2` Invalid command line options were specified.

`99` An unanticipated exception occurred.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	package/pkg
Interface Stability	Uncommitted

See Also pkgdepend(1), pkgrepo(1), pkg.depotd(1m), pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

Notes Because of publication protocol limitations, file system based publication must be used when publishing individual package files that are greater than 128 MB in size. File system based publication is also recommended when access control for a repository is needed.

When using file system based publication, any pkg.depotd processes that are serving the target repository must be restarted after publication is completed for the changes to be reflected in its web interface or search responses. See pkg.depotd(1M) for more information.

Name pkgsign – Image Packaging System signing utility

Synopsis

```
/usr/bin/pkgsign [-a hash_algorithm]
    [-c path_to_signing_certificate]
    [-i path_to_intermediate_cert] ...
    [-k path_to_private_key] [-n] -s path_or_uri
    [--help] [--no-index] [--no-catalog]
    (fmri|pattern) ...
```

Description `pkgsign` updates the manifest for the given FMRIs in place in the repository by adding a signature action using the provided key and certificates. The modified package retains the original timestamp.

Options The following options are supported:

With `-a`, use the signature algorithm *hash_algorithm* instead of the default. The default signature algorithm is `rsa-sha256`. Supported signature algorithms are `rsa-sha256`, `rsa-sha384`, `rsa-sha512`, `sha256`, `sha384`, and `sha512`. A signature algorithm that only specifies a hash algorithm causes the signature value to be the hash of the manifest of the package. A signature algorithm that specifies `rsa` and a hash algorithm causes the signature value to be the hash of the manifest signed with the private key provided (see the `-c` and `-k` options).

With `-c`, add the certificate *path_to_signing_certificate* as the certificate to use when verifying the value of the signature in the action. The `-c` option can only be used with the `-k` option.

With `-i`, add the certificate *path_to_intermediate_cert* as a certificate to use when validating the certificate *path_to_signing_certificate* given as an argument to `-c`. Multiple certificates can be provided by specifying `-i` multiple times.

With `-k`, use the private key stored in *path_to_private_key* to sign the manifest. The `-k` option can only be used with the `-c` option. If `-k` is not set, then the signature value is the hash of the manifest.

With `-n`, perform a trial run that does not change the repository in any way.

With `-s`, sign packages in the repository at *path_or_uri*.

With `--help`, display a usage message.

With `--no-index`, do not update the repository search indexes after the signed manifest has been republished.

With `--no-catalog`, do not update the repository catalog after the signed manifest has been republished.

Examples

EXAMPLE 1 Sign Using the Hash Value of the Manifest

Sign a package published to `http://localhost:10000` using the hash value of the manifest. This is often useful for testing.

```
$ pkgsign -s http://localhost:10000 -a sha256 \
example_pkg@1.0,5.11-0:20100626T030108Z
```

EXAMPLE 2 Sign Using a Key and Certificate

Sign a package published into the file repository in `/foo/bar` using `rsa-sha384` to hash and sign the manifest. The signature key is in `/key/usr2.key`, its associated certificate is in `/key/usr2.cert`, and a certificate needed to validate the certificate is in `/icerts/usr1.cert`.

```
$ pkgsign -s file:///foo/bar/ -a rsa-sha384 \
-k /key/usr2.key -c /key/usr2.cert -i /icerts/usr1.cert \
example_pkg@1.0,5.11-0:20100626T031341Z
```

Exit Status

The following exit values are returned:

`0` Command succeeded.

`1` An error occurred.

`2` Invalid command line options were specified.

`3` Multiple operations were requested, but only some of them succeeded.

`99` An unanticipated exception occurred.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also

`pkg`(1), `pkgrecv`(1), `pkgsend`(1), `pkgrepo`(1), `pkg`(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Name pm-updatemanager – application to update packages

Synopsis

```
/usr/bin/pm-updatemanager [options]

/usr/bin/pm-updatemanager [-hdR] [--help] [--debug]
    [--image-dir dir]
```

Description `pm-updatemanager` checks for and installs available updates for the packages installed on the system.

Note – If the `package/pkg`, `package/pkg/package-manager`, or `package/pkg/update-manager` packages need to be updated, `pm-updatemanager` first updates these packages and then restarts to carry out any remaining updates.

Options The following options are supported:

`-h` or `--help`	Displays a usage message.
`-d` or `--debug`	Runs `pm-updatemanager` in debug mode.
`-R` or `--image-dir` *dir*	Operate on the image rooted at *dir*, rather than on the image discovered automatically.

Examples

EXAMPLE 1 Update the Current Image

Invoke `pm-updatemanager` on the current image. This checks for and installs all available updates for packages installed in the current image.

```
$ /usr/lib/pm-launch pm-updatemanager
```

This is the same command that the desktop menu option System>Administration>Update Manager invokes.

EXAMPLE 2 Update a Specified Image

Invoke `pm-updatemanager` on the image stored at `/aux0/example_root`.

```
$ /usr/lib/pm-launch pm-updatemanager -R /aux0/example_root
```

Exit Status The following exit values are returned:

`0` Everything worked.

`1` An error occurred.

`2` Invalid command line options were specified.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg/update-manager`

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Uncommitted

See Also packagemanager(1), pkg(1), pkg(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Notes When operating on an image you do not own, `pm-updatemanager` needs to be invoked with enough privilege. You will normally invoke `pm-updatemanager` using `/usr/lib/pm-launch` under these circumstances.

REFERENCE

System Administration Commands

Name pkg.depotd – Image Packaging System depot server

Synopsis

```
/usr/lib/pkg.depotd [-a address] [-d inst_root] [-p port]
    [-s threads] [-t socket_timeout] [--add-content]
    [--cfg] [--content-root] [--debug feature_list]
    [--disable-ops=op[/1][,...]]
    [--log-access] [--log-errors] [--mirror]
    [--proxy-base url] [--readonly] [--rebuild]
    [--ssl-cert-file] [--ssl-dialog] [--ssl-key-file]
    [--writable-root]
```

Description `pkg.depotd` is the depot server for the image packaging system. It provides network access to the data contained within a package repository. Clients that do not support direct access to a repository through the file system, or for which network access is the only available or preferred method of transport, typically use the package depot.

Clients such as `pkg`(1), the retrieval client, can retrieve a list of packages and package metadata from a repository directly or through the depot server. `pkgsend`(1), the publication client, can send new versions of packages to a repository directly or through the depot server. `pkgrepo`(1) can be used to create repositories for use with the depot server, or to manage them both directly and through the depot server.

`pkg.depotd` is typically run as a service on the system. Package and software developers might want to run private copies for testing.

The depot does not provide any access control methods of its own. By default, all of the clients that are able to connect are able to read all package data and publish new package versions. The exception is that when running under Service Management Facility (SMF), the default is to run in read-only mode. The "Notes" section below describes some best practices for maintaining a public depot server with evolving content.

Smf Properties The `pkg.depot` server is generally configured via the `smf`(5) properties associated with its service. The following properties are recognized:

`pkg/address` (`net_address`) The IP address on which to listen for connections. The default value is 0.0.0.0 (`INADDR_ANY`), which listens on all active interfaces. To listen on all active IPv6 interfaces, use `::`. Only the first value is used.

`pkg/content_root` (`astring`) The file system path at which the instance should find its static and other web content. The default value is `/usr/share/lib/pkg`.

`pkg/debug` (`astring`) A comma-separated list of debug features to enable. Possible values are:

`headers` Logs the headers of every request to the error log.

`pkg/disable_ops`	(`astring`) A comma-separated list of operations that should be disabled for the depot server. Operations are given as *operation*[/*version*] (`catalog` or `search_1`, for example).
`pkg/image_root`	(`astring`) The path to the image whose file information will be used as a cache for file data.
`pkg/inst_root`	(`astring`) The file system path at which the instance should find its repository data. Required unless `file_root` or `PKG_REPO` has been provided. The default value is `/var/pkgrepo`.
`pkg/log_access`	(`astring`) The destination for any access related information logged by the depot process. Possible values are: `stderr`, `stdout`, `none`, or an absolute path name. The default value is `stdout` if `stdout` is a `tty`. If `stdout` is not a `tty`, the default value is `none`.
`pkg/log_errors`	(`astring`) The destination for any errors or other information logged by the depot process. Possible values are: `stderr`, `stdout`, `none`, or an absolute path name. The default value is `stderr`.
`pkg/mirror`	(`boolean`) Sets whether package mirror mode is used. When true, publishing and metadata operations are disabled and only a limited browser user interface is provided. This property cannot be true when the `pkg/readonly` property is true. The default value is `false`.
`pkg/port`	(`count`) The port number on which the instance should listen for incoming package requests. If SSL certificate and key information has not been provided, the default value is 80; otherwise, the default value is 443.
`pkg/proxy_base`	(`uri`) This changes the base URL for the depot server and is most useful when running behind Apache or some other web server in a reverse proxy configuration.
`pkg/readonly`	(`boolean`) Sets whether modifying operations, such as those initiated by `pkgsend`(1), are disabled. Retrieval operations are still available. This property cannot be true when the `pkg/mirror` property is true. The default value is `true`.

`pkg/socket_timeout` (`count`) The maximum number of seconds the server should wait for a response from a client before closing a connection. The default value is 60.

`pkg/sort_file_max_size` (`count`) The maximum size of the indexer sort file. Used to limit the amount of RAM the depot uses for indexing, or increase it for speed.

`pkg/ssl_cert_file` (`astring`) The absolute path name to a PEM-encoded Certificate file. The default value is `none`. This property must be used with `ssl_key_file`. The depot only responds to SSL requests if both `ssl_cert_file` and `/ssl_key_file` are provided.

`pkg/ssl_dialog` (`astring`) Specifies what method should be used to obtain the passphrase used to decrypt the `ssl_key_file`. Possible values are:

`builtin`	Prompt for the passphrase. This is the default value.
`exec:`*/path/to/program*	Execute the specified external program to obtain the passphrase. The first argument to the program is `''`, and is reserved. The second argument to the program is the port number of the server. The passphrase is printed to `stdout`.
`smf:fmri`	Attempt to retrieve the value of the property `pkg_secure/ssl_key_passphrase` from the service instance related to the FMRI.

`pkg/ssl_key_file` (`astring`) The absolute path name to a PEM-encoded Private Key file. This property must be used with the property `ssl_cert_file`. The depot only responds to SSL requests if both `/ssl_key_file` and `ssl_cert_file` are provided.

`pkg/threads` (`count`) The number of threads started to serve requests. The default value is 60. Suitable only for

	small deployments. This value should be approximately 20 times the number of concurrent clients. The maximum value of `threads` is 5000.
`pkg/writable_root`	(`astring`) The file system path to a directory to which the program has write access. This is used with the `-readonly` option to enable the depot server to create files, such as search indexes, without needing write access to the package information.
`pkg_secure/ssl_key_passphrase`	(`astring`) The password to use to decrypt the `pkg/ssl_key_file`. This value is read-authorization protected using the attribute `solaris.smf.read.pkg-server`.

The presentation and behavior of the Browser User Interface (BUI) of the depot server is controlled using the following properties:

`pkg_bui/feed_description`	(`astring`) A descriptive paragraph for the RSS/Atom feed.
`pkg_bui/feed_icon`	(`astring`) The path name of a small image used to visually represent the RSS/Atom feed. The path name should be relative to the `content_root`. The default value is `web/_themes/pkg-block-icon.png`.
`pkg_bui/feed_logo`	(`astring`) The path name of a large image that will be used to visually brand or identify the RSS/Atom feed. This value should be relative to the `content_root`. The default value is `web/_themes/pkg-block-icon.png`.
`pkg_bui/feed_name`	(`astring`) A short, descriptive name for RSS/Atom feeds generated by the depot serving the repository. The default value is “package repository feed”.
`pkg_bui/feed_window`	(`count`) The number of hours before the feed for the repository was last generated, to include when generating the feed.

The package depot is also able to act as a mirror server for local client images from `pkg`(5). This enables clients that share a subnet on a LAN to mirror their file caches. Clients can download files from one another, thereby reducing load on the package depot server. This functionality is available as an alternate depot service configured by `smf`(5). It uses mDNS and `dns-sd` for service discovery.

The mDNS mirror is generally configured via the `smf`(5) properties associated with its service. The following properties are recognized:

`pkg/image_root` (`astring`) The path to the image whose file information will be used as a cache for file data. The default value is `/`.

`pkg/port` (`count`) The port number on which the instance should listen for incoming package requests. The default value is 80.

Options `pkg.depotd` can read its base configuration information from a file or from the property data of an existing `smf`(5) service instance.

`--cfg` *source* The path name of the file to use when reading and writing configuration data, or a string of the form `smf:`*fmri* where *fmri* is the service fault management resource identifier (FMRI) of the instance to read configuration data from. See "Depot Configuration" below for details on the format of the file specified.

If no preexisting configuration source is available, or to override values read from a configuration file provided using `--cfg`, the following options can be used to alter the default behavior of the depot server:

`-a address`	See `pkg/address` above.
`--content-root` *root_dir*	See `pkg/content_root` above.
`-d` *inst_root*	See `pkg/inst_root` above.
`--debug` *features*	See `pkg/debug` above.
`--disable-ops` *op_list*	See `pkg/disable_ops` above.
`--image-root` *path*	See `pkg/image_root` above.
`--log-access` *dest*	See `pkg/log_access` above.
`--log-errors` *dest*	See `pkg/log_errors` above.
`--mirror`	See `pkg/mirror` above.
`-p` *port*	See `pkg/port` above.
`--proxy-base` *url*	See `pkg/proxy_base` above. This option is ignored if an empty value is provided.
`--readonly`	See `pkg/readonly` above.
`-s` *threads*	See `pkg/threads` above.
`-s-ort-file-max-size` *bytes*	See `pkg/sort_file_max_size` above.
`--ssl-cert-file` *source*	See `pkg/ssl_cert_file` above.
`--ssl-dialog` *type*	See `pkg/ssl_dialog` above.
`--ssl-key-file` *source*	See `pkg/ssl_key_file` above.

`-t` *socket_timeout* — See `pkg/socket_timeout` above.

`--writable-root` *path* — See `pkg/writable_root` above.

Additional administrative and management functionality for package repositories is provided by `pkgrepo`(1).

Depot Configuration

When a configuration file is provided (instead of an `smf`(5) FMRI) by using the `--cfg` option, the depot server reads and writes all configuration data in a simple text format. The configuration data is described in "SMF Properties" above. The configuration data consists of sections, lead by a `[`*section*`]` header, and followed by `name` = *value* entries. Continuations are in the style of RFC 822. Values can be split over multiple lines by beginning continuation lines with whitespace.

Any required values not provided in the configuration file must be provided using the option listed in "Options" above. A sample configuration file might look like this:

```
[pkg]
port = 80
inst_root = /export/repo

[pub_example_com]
feed_description = example.com's software
  update log
```

Examples

EXAMPLE 1 Enabling the Depot Server

```
# svcadm enable application/pkg/server
```

EXAMPLE 2 Changing the Listening Port of the Server.

```
# svccfg -s application/pkg/server setprop pkg/port = 10000
# svcadm refresh application/pkg/server
# svcadm restart application/pkg/server
```

EXAMPLE 3 Enabling the Mirror

```
# svcadm enable application/pkg/dynamic-mirror
```

Environment Variables

`PKG_REPO` — Specifies the directory that contains the repository to serve. This value is ignored if `-d` is specified.

`PKG_DEPOT_CONTENT` — Specifies the directory that contains static content served by the depot. The files listed below under "Files" should be present in this directory, although their content can differ from the supplied default content.

Exit Status

The following exit values are returned:

`0` Successful operation.

1 An error occurred.

2 Invalid command line options were specified.

99 An unanticipated exception occurred.

Files

`/usr/share/lib/pkg` Default presentation content location. Modify `pkg/content_root` to select an alternate location.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also

`dns-sd`(1M), `mdnsd`(1M), `pkg`(1), `pkgrepo`(1), `pkgsend`(1), `syslogd`(1M), `smf`(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

Notes

The `pkd.depotd` service is managed by SMF under the service identifier `svc:/application/pkg/server`.

The mDNS mirror service is managed by SMF under the service identifier `svc:/application/pkg/dynamic-mirror`.

To control read access to the depot, you can use an HTTP reverse proxy in combination with authentication methods such as client based SSL certificate access, which `pkg`(1) natively supports.

Changes to configuration, or changes to package data using file system based operations, require a restart of the depot server process so that the changes can be reflected in operations and output. Use one of the following methods to restart the depot server process:

- Use `svcadm`(1M) to restart the `application/pkg/server` instance.
- Send a `SIGUSR1` signal to the depot server process using `kill`(1). This executes a "graceful restart" that leaves the process intact but reloads all configuration, package, and search data:

  `# kill -USR1` *pid*

Name pkg.sysrepo – Image Packaging System system repository configuration

Synopsis

```
pkg.sysrepo -p port [-c cache_dir] [-s cache_size]
    [-w http_proxy] [-W https_proxy]
```

Description `pkg.sysrepo` is used to generate the configuration files for the Image Packaging System (IPS) system repository. `pkg.sysrepo` is called by the `svc:/application/pkg/system-repository` Service Management Facility (SMF) service. Changes in configuration should be made to the properties in the SMF service.

The system repository is responsible for providing access to the package repositories configured in a reference image through a centralized proxy. Publisher configuration changes made to that reference image are seen immediately by any clients configured to use the system repository.

The system repository is primarily used in the global zone to allow non-global zones to access the repositories configured in the global zone. The SMF services `svc:/application/pkg/zones-proxyd` and `svc:/application/pkg/zones-proxy-client` are responsible for providing the transport between non-global zones and the global zone. This transport is only used by `pkg`(5).

Note that only http, https, and v4 file repositories are supported. p5p-based file repositories or older file repository formats are not supported. See `pkgrepo`(1) for more information about repository versions.

Options The following options are supported:

`-c` *cache_dir* — The absolute path to a directory that should be used by the system repository for caching responses from the publishers configured.

By default, a file-cache is used. However, the special value `memory` can be used to indicate the an in-memory cache should be used. The special value `None` can be used to indicate that the system repository should not perform any caching. This setting should be configured using the `config/cache_dir` SMF property.

`-p` *port* — The port that the system repository should use to listen for requests. This setting should be configured using the `config/port` SMF property.

`-s` *cache_size* — An integer value in megabytes that defines the maximum cache size of the system repository. This setting should be configured using the `config/cache_max` SMF property.

`-w` *http_proxy* — A string of the form *scheme*`://`*hostname*`[:`*port*`]` that defines a web proxy that the system repository can use to access http-based package repositories. This setting can be configured using the `config/http_proxy` SMF property.

`-W` *https_proxy* A string of the form *scheme*`://`*hostname*`[:`*port*`]` that defines a web proxy that the system repository can use to access https-based package repositories. This setting can be configured using the `config/https_proxy` SMF property.

Examples

EXAMPLE 1 Enabling the System Repository

```
$ svcadm enable svc:/application/pkg/system-repository
```

Exit Status

The following exit values are returned:

`0` Command succeeded.

`1` Command failed to write a valid configuration.

`2` Invalid command line options were specified.

`99` An unanticipated exception occurred.

Attributes

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also

`pkg`(1), `pkg.depotd`(1m), `pkg`(5)

`http://hub.opensolaris.org/bin/view/Project+pkg/`

REFERENCE

Standards, Environments, and Macros

Name pkg – Image Packaging System

Description The image packaging system, pkg(5), is a framework that provides for software lifecycle management (installation, upgrade, and removal). Image packaging manages software in units of packages, which are collections of actions, defined by a set of key/value pairs and possibly a data payload. In many cases, actions are files found in a file system, but they also represent other installable objects, such as drivers, services, and users.

Package Fmris and Versions Each package is represented by a fault management resource identifier (FMRI) with the scheme pkg:. The full FMRI for a package consists of the scheme, a publisher, the package name, and a version string in the following format:

```
pkg://solaris/system/library/c++-runtime@0.5.11,5.11-0.174.0.0.0.0.0:20110921T190358Z
```

solaris is the publisher. system/library/c++-runtime is the package name. Although the namespace is hierarchical and arbitrarily deep, there is no enforced containment; the name is essentially arbitrary. The publisher information is optional, but must be preceded by pkg:// if present. An FMRI that includes the publisher is often referred to as being "fully qualified." If publisher information is not present, then the package name should generally be preceded by pkg:/.

Packaging clients often allow the scheme of an FMRI to be omitted if it does not contain publisher information. For example, pkg:/system/library/c++-runtime can be written as system/library/c++-runtime. If the scheme is omitted, clients also allow omission of all but the last component of a package name for matching purposes. For example, system/library/c++-runtime could be written as library/c++-runtime or c++-runtime, which would then match packages named c++-runtime or package names ending in /c++-runtime.

A publisher name identifies a person, group of persons, or an organization as the source of one or more packages. To avoid publisher name collisions and help identify the publisher, a best practice is to use a domain name that represents the entity publishing the packages as a publisher name.

The version follows the package name, separated by an at sign (@). The version consists of four sequences of numbers, separated by punctuation. The elements in the first three sequences are separated by dots, and the sequences are arbitrarily long. Leading zeros in version components (for example, 01.1 or 1.01) are not permitted. Trailing zeros (for example, 1.10) are permitted.

The first part of the version is the component version. For components tightly bound to the operating system, this is usually the value of uname -r for that version of the operating system. For a component with its own development lifecycle, this sequence is a dotted release number, such as 2.4.10.

The second part of the version, which if present must follow a comma (,), is the build version. The build version specifies what version of the operating system the contents of the package were built on, providing a minimum bound on which operating system version the contents can be expected to run successfully.

The third part of the version, which if present must follow a hyphen (-), is the branch version. The branch version is a versioning component that provides vendor-specific information. The branch version can be incremented when the packaging metadata is changed, independently of the component version. The branch version might contain a build number or other information.

The fourth part of the version, which if present must follow a colon (:), is a timestamp. The timestamp represents when the package was published.

When performing comparisons between versions, no component of the full version is considered unless the components to its left are the same. Thus, "4.3-1" is greater than "4.2-7" because "4.3" is greater than "4.2", and "4.3-3" is greater than "4.3-1" because "3" is greater than "1".

Many parts of the system, when appropriate, abridge FMRIs when displaying them, and accept input in shorter forms to reduce the volume of information displayed or required. Typically, the scheme, publisher, build version, and timestamp can be elided. Sometimes all of the versioning information can be omitted.

Actions Actions represent the installable objects on a system. Actions are described in the manifest of a package. Every action consists primarily of its name and a key attribute. Together, these refer to a unique object as it follows a version history. Actions can have other attributes. Some attributes are interpreted directly by the packaging system. Other attributes might be useful only to the system administrator or the end-user.

Actions have a simple text representation:

action_name attribute1=value1 attribute2=value2 ...

Names of attributes cannot have whitespace, quotation marks, or equals signs (=) in them. All characters after the first equals sign belong to the value. Values can have all of those, though spaces must be enclosed in single or double quotation marks. Single quotation marks do not need to be escaped inside a string that is enclosed in double quotation marks, and double quotation marks do not need to be escaped inside a string that is enclosed in single quotation marks. A quotation mark can be prefixed with a backslash (\) character to avoid terminating the quoted string. A backslash can be escaped with a backslash.

Attributes can be named more than once with multiple values. These are treated as unordered lists.

Actions with many attributes can create long lines in a manifest file. Such lines can be wrapped by terminating each incomplete line with a backslash. Note that this continuation character must occur between attribute/value pairs. Neither attributes nor their values nor the combination can be split.

The attributes listed below are not an exhaustive set. In fact, the attributes that can be attached to an action are arbitrary, and the standard sets of attributes are easy to augment to incorporate future developments.

Certain action attributes cause additional operations to be executed outside of the packaging context. These attributes are documented in the "Actuators" section below.

File Actions

The `file` action represents an ordinary file. The `file` action references a payload, and has four standard attributes:

`path` The file system path where the file is installed. This is a `file` action's key attribute.

`mode` The access permissions (in numeric form) of the file. These are simple permissions only, not ACLs.

`owner` The name of the user that owns the file.

`group` The name of the group that owns the file.

The payload is a positional attribute in that it is not named. It is the first word after the action name. In a published manifest, it is the `SHA-1` hash of the file contents. If present in a manifest that has yet to be published, it represents the path where the payload can be found. See `pkgsend`(1). The hash attribute can be used instead of the positional attribute, should the value include an equals sign. Both can be used in the same action. However, the hashes must be identical.

Other attributes include:

`preserve` This specifies that the file's contents should not be overwritten on upgrade if the contents are determined to have changed since the file was installed or last upgraded. On initial installs, if an existing file is found, the file is salvaged (stored in `/var/pkg/lost+found`).

If the value of `preserve` is `renameold`, then the existing file is renamed with the extension `.old`, and the new file is put in its place.

If the value of `preserve` is `renamenew`, then the existing file is left alone, and the new file is installed with the extension `.new`.

If the value of `preserve` is `legacy`, then this file is not installed for initial package installs. On upgrades, any existing file is renamed with the extension `.legacy`, and then the new file is put in its place.

If the value of `preserve` is `true` (or a value not listed above, such as `strawberry`), then the existing file is left alone, and the new file is not installed.

`overlay` This specifies whether the action allows other packages to deliver a file at the same location or whether it delivers a file intended to overlay another. This functionality is intended for use with configuration files that do not participate in any self-assembly (for example, `/etc/motd`) and that can be safely overwritten.

If `overlay` is not specified, multiple packages cannot deliver files to the same location.

If the value of `overlay` is `allow`, one other package is allowed to deliver a file to the same location. This value has no effect unless the `preserve` attribute is also set.

If the value of `overlay` is `true`, the file delivered by the action overwrites any other action that has specified `allow`. Changes to the installed file are preserved based on the value of the `preserve` attribute of the overlaying file. On removal, the contents of the file are preserved if the action being overlaid is still installed, regardless of whether the `preserve` attribute was specified. Only one action can overlay another, and the `mode`, `owner`, and `group` attributes must match.

Files can also be "tasted," and depending on the flavor, can have additional attributes. For ELF files, the following attributes are recognized:

`elfarch` The architecture of the ELF file. This is the output of `uname -p` on the architecture for which the file is built.

`elfbits` This is `32` or `64`.

`elfhash` This is the hash of the "interesting" ELF sections in the file. These are the sections that are mapped into memory when the binary is loaded. These are the only sections necessary to consider when determining whether the executable behavior of two binaries will differ.

`original_name` This attribute is used to handle editable files moving from package to package or from place to place, or both. The form this takes is the name of the originating package, followed by a colon and the original path to the file. Any file being deleted is recorded either with its package and path, or with the value of the `original_name` attribute if specified. Any editable file being installed that has the `original_name` attribute set uses the file of that name if it is deleted as part of the same packaging operation.

`revert-tag` This attribute is used to tag editable files that should be reverted as a set. Multiple `revert-tag` values can be specified. The file reverts to its manifest-defined state when `pkg revert` is invoked with any of those tags

specified. See pkg(1).

Directory Actions

The dir action is like the file action in that it represents a file system object. The dir action represents a directory instead of an ordinary file. The dir action has the same four standard attributes as the file action, and path is the key attribute.

Directories are reference counted in IPS. When the last package that either explicitly or implicitly references a directory no longer does so, that directory is removed. If that directory contains unpackaged file system objects, those items are moved into $IMAGE_META/lost+found. See the "Files" section for more information about $IMAGE_META.

To move unpackaged contents into a new directory, the following attribute might be useful:

salvage-from
: This names a directory of salvaged items. A directory with such an attribute inherits on creation the salvaged directory contents if they exist.

Link Actions

The link action represents a symbolic link. The link action has the following standard attributes:

path
: The file system path where the symbolic link is installed. This is a link action's key attribute.

target
: The target of the symbolic link. The file system object to which the link resolves.

mediator
: Specifies the entry in the mediation namespace shared by all path names participating in a given mediation group (for example, python). Link mediation can be performed based on mediator-version and/or mediator-implementation. All mediated links for a given path name must specify the same mediator. However, not all mediator versions and implementations need to provide a link at a given path. If a mediation does not provide a link, then the link is removed when that mediation is selected. A mediator, in combination with a specific version and/or implementation represents a mediation that can be selected for use by the packaging system.

mediator-version
: Specifies the version (expressed as a dot-separated sequence of nonnegative integers) of the interface described by the mediator attribute. This attribute is required if mediator is specified and mediator-implementation is not. A local system administrator can set the version to use explicitly. The value specified should generally match the version of the package delivering the link (for example, runtime/python-26 should use mediator-version=2.6), although this is not required.

mediator-implementation
: Specifies the implementation of the mediator for use in addition to or instead of the mediator-version. Implementation strings are not considered to be ordered and a string is arbitrary selected by pkg(5) if not explicitly specified by a system administrator.

The value can be a string of arbitrary length composed of alphanumeric characters and spaces. If the implementation itself can be versioned or is versioned, then the version should be specified at the end of the string, after a @ (expressed as a dot-separated sequence of nonnegative integers). If multiple versions of an implementation exist, the default behavior is to select the implementation with the greatest version.

If only one instance of an implementation mediation link at a particular path is installed on a system, then that one is chosen automatically. If future links at the path are installed, the link is not switched unless a vendor, site, or local override applies, or if one of the links is version mediated.

`mediator-priority`
When resolving conflicts in mediated links, `pkg`(5) normally chooses the link with the greatest value of `mediator-version` or based on `mediator-implementation` if that is not possible. This attribute is used to specify an override for the normal conflict resolution process.

If this attribute is not specified, the default mediator selection logic is applied.

If the value is `vendor`, the link is preferred over those that do not have a `mediator-priority` specified.

If the value is `site`, the link is preferred over those that have a value of `vendor` or that do not have a `mediator-priority` specified.

A local system administrator can override the selection logic described above.

Hardlink Actions

The `hardlink` action represents a hard link. It has the same attributes as the `link` action, and `path` is also its key attribute.

Driver Actions

The `driver` action represents a device driver. The `driver` action does not reference a payload. The driver files themselves must be installed as `file` actions. The following attributes are recognized (see `add_drv`(1M) for more information):

`name`	The name of the driver. This is usually, but not always, the file name of the driver binary. This is the `driver` action's key attribute.
`alias`	This represents an alias for the driver. A given driver can have more than one `alias` attribute. No special quoting rules are necessary.
`class`	This represents a driver class. A given driver can have more than one `class` attribute.
`perms`	This represents the file system permissions for the driver's device nodes.
`clone_perms`	This represents the file system permissions for the clone driver's minor nodes for this driver.

`policy` This specifies additional security policy for the device. A given driver can have more than one `policy` attribute, but no minor device specification can be present in more than one attribute.

`privs` This specifies privileges used by the driver. A given driver can have more than one `privs` attribute.

`devlink` This specifies an entry in `/etc/devlink.tab`. The value is the exact line to go into the file, with tabs denoted by `\t`. See `devlinks`(1M) for more information. A given driver can have more than one `devlink` attribute.

Depend Actions

The `depend` action represents an inter-package dependency. A package can depend on another package because the first requires functionality in the second for the functionality in the first to work, or even to install. Dependencies can be optional. If a dependency is not met at the time of installation, the packaging system attempts to install or update the dependent package to a sufficiently new version, subject to other constraints.

The following attributes are recognized:

`fmri` The FMRI representing the dependent package. This is the `dependency` action's key attribute. The `fmri` value must not include the publisher. The package name is assumed to be complete. Dependencies of type `require-any` can have multiple `fmri` attributes. A version is optional on the `fmri` value, though for some types of dependencies, an `fmri` with no version has no meaning.

`type` The type of the dependency.

If the value is `require`, then the dependency is required and must have a version equal to or greater than the version specified in the `fmri` attribute. If the version is not specified, any version satisfies the dependency. A package cannot be installed if any of its required dependencies cannot be satisfied.

If the value is `optional`, then the dependency, if present, must be at the specified version level or greater.

If the value is `exclude`, then the containing package cannot be installed if the dependency is present at the specified version level or greater. If no version is specified, the dependent package cannot be installed concurrently with the package specifying the dependency.

If the value is `incorporate`, then the dependency is optional, but the version of the dependent package is constrained. See “Constraints and Freezing” below.

If the value is `require-any`, then any one of multiple dependent packages as specified by multiple `fmri` attributes can satisfy the dependency, following the same rules as the `require` dependency type.

If the value is `conditional`, the dependency is required only if the package defined by the `predicate` attribute is present on the system.

If the value is `origin`, the dependency must, if present, be at the specified value or better on the image to be modified prior to installation. If the value of the `root-image` attribute is `true`, the dependency must be present on the image rooted at / in order to install this package.

If the value is `group`, the dependency is required unless the package is on the image avoid list. Note that obsolete packages silently satisfy the group dependency. See the `avoid` subcommand in `pkg`(1).

If the value is `parent`, then the dependency is ignored if the image is not a child image. If the image is a child image then the dependency is required to be present in the parent image. The package version matching for a `parent` dependency is the same as that used for `incorporate` dependencies.

`predicate` The FMRI representing the predicate for `conditional` dependencies.

`root-image` Has an effect only for `origin` dependencies as mentioned above.

License Actions

The `license` action represents a license or other informational file associated with the package contents. A package can deliver licenses, disclaimers, or other guidance to the package installer through the use of the `license` action.

The payload of the `license` action is delivered into the image metadata directory related to the package, and should only contain human-readable text data. It should not contain HTML or any other form of markup. Through attributes, `license` actions can indicate to clients that the related payload must be displayed and/or require acceptance of it. The method of display and/or acceptance is at the discretion of clients.

The following attributes are recognized:

`license` This is a `license` action's key attribute. This attribute provides a meaningful description for the license to assist users in determining the contents without reading the license text itself. Some example values include:

- ABC Co. Copyright Notice
- ABC Co. Custom License
- Common Development and Distribution License 1.0 (CDDL)
- GNU General Public License 2.0 (GPL)
- GNU General Public License 2.0 (GPL) Only

- MIT License
- Mozilla Public License 1.1 (MPL)
- Simplified BSD License

The `license` value must be unique within a package. Including the version of the license in the description, as shown in several of the examples above, is recommended. If a package has code under multiple licenses, use multiple `license` actions. The length of the license attribute value should not be more than 64 characters.

`must-accept` When `true`, this license must be accepted by a user before the related package can be installed or updated. Omission of this attribute is equivalent to `false`. The method of acceptance (interactive or configuration-based, for example) is at the discretion of clients.

`must-display` When `true`, the action's payload must be displayed by clients during packaging operations. Omission of this value is equivalent to `false`. This attribute should not be used for copyright notices, only actual licenses or other material that must be displayed during operations. The method of display is at the discretion of clients.

Legacy Actions

The `legacy` action represents package data used by a legacy packaging system. The attributes associated with this action are added into the legacy system's databases so that the tools querying those databases can operate as if the legacy package were actually installed. In particular, this should be sufficient to convince the legacy system that the package named by the `pkg` attribute is installed on the system, so that the package can be used to satisfy dependencies.

The following attributes, named in accordance with the parameters on `pkginfo`(4), are recognized:

`category` The value for the `CATEGORY` parameter. The default value is `system`.

`desc` The value for the `DESC` parameter.

`hotline` The value for the `HOTLINE` parameter.

`name` The value for the `NAME` parameter. The default value is `none provided`.

`pkg` The abbreviation for the package being installed. The default value is the name from the FMRI of the package. This is a `legacy` action's key attribute.

`vendor` The value for the `VENDOR` parameter.

`version` The value for the VERSION parameter. The default value is the version from the FMRI of the package.

Set Actions The `set` action represents a package-level attribute, or metadata, such as the package description.

The following attributes are recognized:

`name` The name of the attribute.

`value` The value given to the attribute.

The `set` action can deliver any metadata the package author chooses. However, there are a number of well defined attribute names that have specific meaning to the packaging system.

`pkg.fmri`	See "Package FMRIs and Versions" in the "Description" section.
`info.classification`	One or more tokens that a `pkg`(5) client can use to classify the package. The value should have a scheme (such as "org.opensolaris.category.2008" or "org.acm.class.1998") and the actual classification, such as "Applications/Games", separated by a colon (:).
`pkg.description`	A detailed description of the contents and functionality of the package, typically a paragraph or so in length.
`pkg.obsolete`	When `true`, the package is marked obsolete. An obsolete package can have no actions other than more set actions, and must not be marked renamed.
`pkg.renamed`	When `true`, the package has been renamed. There must be one or more `depend` actions in the package as well that point to the package versions to which this package has been renamed. A package cannot be marked both renamed and obsolete, but otherwise can have any number of set actions.
`pkg.summary`	A short, one-line description of the package.

Group Actions The `group` action defines a UNIX group as defined in `group`(4). No support is present for group passwords. Groups defined with this action initially have no user list. Users can be added with the `user` action. The following attributes are recognized:

`groupname` The value for the name of the group.

`gid` The group's unique numerical id. The default value is the first free group under 100.

User Actions The `user` action defines a UNIX user as defined in `/etc/passwd`, `/etc/shadow`, `/etc/group`, and `/etc/ftpd/ftpusers` files. Users defined with this attribute have entries added to the appropriate files.

The following attributes are recognized:

`username`	The unique name of the user
`password`	The encrypted password of the user. Default value is `*LK*`. See `shadow`(4).
`uid`	The unique uid of the user. Default value is first free value under 100.
`group`	The name of the user's primary group. Must be found in `/etc/group`.
`gcos-field`	The value of the `gcos` field in `/etc/passwd`. Default value is `username`.
`home-dir`	The user's home directory. Default value is /.
`login-shell`	The user's default shell. Default value is empty.
`group-list`	Secondary groups to which the user belongs. See `group`(4).
`ftpuser`	Can be set to `true` or `false`. The default value of `true` indicates that the user is permitted to login via FTP. See `ftpusers`(4).
`lastchg`	The number of days between January 1, 1970, and the date that the password was last modified. Default value is empty. See `shadow`(4).
`min`	The minimum number of days required between password changes. This field must be set to 0 or above to enable password aging. Default value is empty. See `shadow`(4).
`max`	The maximum number of days the password is valid. Default value is empty. See `shadow`(4).
`warn`	The number of days before password expires that the user is warned. See `shadow`(4).
`inactive`	The number of days of inactivity allowed for that user. This is counted on a per-machine basis. The information about the last login is taken from the machine's `lastlog` file. See `shadow`(4).
`expire`	An absolute date expressed as the number of days since the UNIX Epoch (January 1, 1970). When this number is reached, the login can no longer be used. For example, an expire value of 13514 specifies a login expiration of January 1, 2007. See `shadow`(4).
`flag`	Set to empty. See `shadow`(4).

Actuators In certain contexts, additional operations can be appropriate to execute in preparation for or following the introduction of a particular action. These additional operations are generally needed only on a live system image, and are operating system specific. When multiple actions involved in a package installation or removal have identical actuators, then the operation corresponding to actuator presence is executed once for that installation or removal.

Incorrectly specified actuators can result in package installation failure, if the actuator cannot determine a means of making safe installation progress.

The following actuators are defined:

`reboot-needed`
: Can be set to `true` or `false`. If an action with this actuator set to `true` is installed or updated during a package installation, then the packaging transaction can be advertised as requiring a reboot. Certain client implementations might take additional steps, such as performing the entire package operation using a clone of the image, in the case that the image is the live system image.

`disable_fmri`, `refresh_fmri`, `restart_fmri`, `suspend_fmri`
: Each of these actuators takes the value of an FMRI of a service instance to operate on during the package installation or removal. `disable_fmri` causes the given FMRI to be disabled prior to action removal, per the `disable` subcommand to `svcadm`(1M). `refresh_fmri` and `restart_fmri` cause the given FMRI to be refreshed or restarted after action installation, update, or removal per the respective subcommands of `svcadm`(1M). Finally, `suspend_fmri` causes the given FMRI to be disabled temporarily prior to the action install phase, and then enabled after the completion of that phase.

  The value can contain a pattern that matches multiple service instances. However, it must do so explicitly with a glob as accepted by `svcs`(1), rather than doing so implicitly by not indicating any instances.

Constraints and Freezing

When a package is transitioned to a new version, or when it is added to or removed from the system, the version that is chosen, or whether removal is allowed, is determined by a variety of constraints put on the package. Those constraints can be defined by other packages in the form of dependencies, or by the administrator in the form of freezes.

The most common form of constraint is delivered by the `require` dependency, as described in "Depend Actions" above. Such a constraint prevents the package from being downgraded or removed.

Most parts of the operating system are encapsulated by packages called *incorporations*. These packages primarily deliver constraints represented by the `incorporate` dependency.

As described above, an incorporated package need not be present on the system, but if it is, then it specifies both an inclusive minimum version and an exclusive maximum version. For example, if the dependent FMRI has a version of 1.4.3, then no version less than 1.4.3 would satisfy the dependency, and neither would any version greater than or equal to 1.4.4. However, versions that merely extended the dotted sequence, such as 1.4.3.7, would be allowed.

Incorporations are used to force parts of the system to upgrade synchronously. For some components, such as the C library and the kernel, this is a basic requirement. For others, such as a simple userland component on which nothing else has a dependency, the synchronous upgrade is used merely to provide a known and tested set of package versions that can be referred to by a particular version of the incorporation.

Since an incorporation is simply a package, it can be removed, and all the constraints it delivers are therefore relaxed. However, many of the incorporations delivered by Oracle Solaris are required by the packages they incorporate because that relaxation would not be safe.

Attempting an upgrade of a package to a version that is not allowed by an installed incorporation will not attempt to find a newer version of the incorporation in order to satisfy the request, but will instead fail. If the constraint itself must be moved, and the incorporation specifying it cannot be removed, then the incorporation must be upgraded to a version that specifies a desired version of the constraint. Upgrading an incorporation causes all of the incorporated packages that would not satisfy the constraints delivered by the new version to be upgraded as well.

A system administrator can constrain a package by using the `pkg freeze` command. The named package is constrained to the version installed on the system if no version is provided. If a versioned package is provided, then this administrative constraint, or freeze, acts as if an incorporate dependency were installed where the `fmri` attribute had the value of the provided package version.

A freeze is never lifted automatically by the packaging system. To relax a constraint, use the `pkg unfreeze` command.

Publishers and Repositories

As detailed above, a publisher is simply a name that package clients use to identify the provider of packages. Publishers can distribute their packages using package repositories and/or package archives. There are two types of repositories currently supported by the package system: origin repositories and mirror repositories.

An *origin* is a package repository that contains all of the metadata (such as catalogs, manifests, and search indexes) and content (files) for one or more packages. If multiple origins are configured for a given publisher in an image, the package client API attempts to choose the best origin to retrieve package data from. This is the most common type of repository, and is implicitly created whenever `pkgsend` or `pkgrecv` is used on a package repository.

A *mirror* is a package repository that contains only package content (files). If one or more mirrors are configured for a given publisher in an image, the client API prefers the mirrors for package content retrieval and attempts to choose the best one to retrieve package content from. If the mirror is unreachable, does not have the required content, or is slower, the client API retrieves the content from any configured origin repositories. Mirrors are intended for content sharing among a trusted set of clients using the dynamic mirror functionality of `pkg.depotd`(1M). Mirrors are also intended to be used to authenticate access to package metadata, but distribute the package content without authentication. For example, a client might be configured with an `https` origin that requires an SSL key and certificate pair to access, and with an `http` mirror that provides the package content. In this way, only authorized clients can install or update the packages, while the overhead of authentication for package content retrieval is avoided. A mirror can be created by removing all subdirectories of a repository except those named `file` and their parents. An origin repository can be also be provisioned as a mirror by using the mirror mode of `pkg.depotd`(1M).

Facets and Variants Software can have components that are optional and components that are mutually exclusive. Examples of optional components include locales and documentation. Examples of mutually exclusive components include SPARC or x86 and debug or non-debug binaries.

In IPS, optional components are called *facets* and mutually exclusive components are called *variants*. Facets and variants are specified as tags on package actions. Each facet and variant tag has a name and a value. A single action can have multiple facet and variant tags. Examples of components with multiple facet and variant tags include an architecture-specific header file that is used by developers, or a component that is only for a SPARC global zone.

An example of a variant tag is `variant.arch=sparc`. An example of a facet tag is `facet.devel=true`. Facets and variants are often referred to without the leading `facet.` and `variant.`.

Facets and variants are special properties of the image and cannot be set on individual packages. To view the current values of the facets and variants set on the image, use the `pkg facet` and `pkg variant` commands as shown in the pkg(1) man page. To modify the values of the facets and variants set on the image, use the `pkg change-facet` and `pkg change-variant` commands.

Facets are boolean: They can be set only to `true` (enabled) or `false` (disabled). By default, all facets are considered to be set to `true` in the image. A facet tag on an action should only have the value `true`; other values have undefined behavior. A facet set on the image can be a full facet such as `doc.man` or a pattern such as `locale.*`. This is useful when you want to disable a portion of the facet namespace, and only enable individual facets within it. For example, you could disable all locales and then only enable one or two specific locales, as shown in the following example:

```
# pkg change-facet locale.*=false
[output about packages being updated]
# pkg change-facet locale.en_US=true
[output about packages being updated]
```

Most variants can have any number of values. For example, the `arch` variant can be set to `i386`, `sparc`, `ppc`, `arm`, or whatever architectures the distribution supports. (Only `i386` and `sparc` are used in Oracle Solaris.) The exception are the `debug` variants. The `debug` variants can only be set to `true` or `false`; other values have undefined behavior. If a file action has both non-debug and debug versions, both versions must have the applicable `debug` variant explicitly set, as shown in the following example:

```
file group=sys mode=0644 overlay=allow owner=root \
  path=etc/motd pkg.csize=115 pkg.size=103 preserve=true \
  variant.debug.osnet=true

file group=sys mode=0644 overlay=allow owner=root \
  path=etc/motd pkg.csize=68 pkg.size=48 preserve=true \
  variant.debug.osnet=false
```

The variant value must be set on the image in order for a package using the variant to be installed. The `arch` and `zone` variants are set by the program that creates the image and installs its initial contents. The `debug.*` variants are `false` in the image by default.

The facets and variants set on the image affect whether a particular action is installed.

- Actions with no facet or variant tags are always installed.
- Actions with facet tags are installed unless all of the facets or facet patterns matching the tags are set to `false` on the image. If any facet is set to `true` or is not explicitly set (`true` is the default), then the action is installed.
- Actions with variant tags are installed only if the values of all the variant tags are the same as the values set in the image.
- Actions with both facet and variant tags are installed if both the facets and the variants allow the action to be installed.

You can create your own facet and variant tags. The following tags are in common use in Oracle Solaris.

Variant Name	Possible Values
`variant.arch`	`sparc`, `i386`
`variant.opensolaris.zone`	`global`, `nonglobal`
`variant.debug.*`	`true`, `false`

The following list shows a small sample of the facet tags that are used in Oracle Solaris:

```
facet.devel             facet.doc
facet.doc.html          facet.doc.info
facet.doc.man           facet.doc.pdf
facet.locale.de         facet.locale.en_GB
facet.locale.en_US      facet.locale.fr
facet.locale.ja_JP      facet.locale.zh_CN
```

Image Policies Image policies are defined by image properties with boolean values. See "Image Properties" in the `pkg`(1) man page for descriptions of the `flush-content-cache-on-success` and `send-uuid` properties and information about how to view and modify their values.

Files Since `pkg`(5) images can be located arbitrarily within a larger file system, the token `$IMAGE_ROOT` is used to distinguish relative paths. For a typical system installation, `$IMAGE_ROOT` is equivalent to /.

`$IMAGE_ROOT/var/pkg`	Metadata directory for a full or partial image.
`$IMAGE_ROOT/.org.opensolaris,pkg`	Metadata directory for a user image.

Within the metadata of a particular image, certain files and directories can contain information useful during repair and recovery. The token `$IMAGE_META` is used to refer to the top-level directory that contains the metadata. `$IMAGE_META` is typically one of the two paths given above.

`$IMAGE_META/lost+found`	Location of conflicting directories and files moved during a package operation.
`$IMAGE_META/publisher`	Contains a directory for each publisher. Each directory stores publisher-specific metadata.

Other paths within the `$IMAGE_META` directory hierarchy are Private, and are subject to change.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`package/pkg`
Interface Stability	Uncommitted

See Also `pkg`(1), `pkgsend`(1), `pkg.depotd`(1m), `pkg.sysrepo`(1m), `svcs`(1), `svcadm`(1M)

`http://hub.opensolaris.org/bin/view/Project+pkg/`