pkg(1)

```
NAME
     pkg - image packaging retrieval client

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

     /usr/bin/pkg install [-nvq] [--accept] [--licenses] [--no-index]
         [--no-refresh] [--deny-new-be | --require-new-be] [--be-name name]
         pkg_fmri_pattern ...
     /usr/bin/pkg uninstall [-nrvq] [--no-index]
         [--deny-new-be | --require-new-be] [--be-name name]
         pkg_fmri_pattern ...

     /usr/bin/pkg update [-fnvq] [--accept] [--be-name name]
         [--deny-new-be | --require-new-be] [--licenses] [--no-index]
         [--no-refresh]

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

     /usr/bin/pkg contents [-Hmr] [-a attribute=pattern ...]
         [-o attribute ...] [-s sort_key] [-t action_type ...]
         [pkg_fmri_pattern ...]
     /usr/bin/pkg info [-lr] [--license] [pkg_fmri_pattern ...]
     /usr/bin/pkg list [-Hafnsuv] [--no-refresh] [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 image-create [-FPUfz] [--force]
         [--full|--partial|--user] [--zone] [-k ssl_key] [-c ssl_cert]
         [--no-refresh] [--variant <variant_spec>=<instance> ...]
         [-g uri|--origin=uri ...] [-m uri|--mirror=uri ...]
         [--facet <facet_spec>=[True|False] ...]
         (-p|--publisher) [<name>=]<repo_uri> dir

     /usr/bin/pkg variant [-H] [<variant_spec>]
     /usr/bin/pkg change-variant [-nvq] [--accept]
         [--deny-new-be | --require-new-be] [--be-name name]
         [--licenses] <variant_spec>=<instance> ...

     /usr/bin/pkg facet [-H] [<facet_spec>]
     /usr/bin/pkg change-facet [-nvq] [--accept] [--be-name name]
         [--deny-new-be | --require-new-be]
         [--licenses] <facet_spec>=[True|False|None] ...

     /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 property [-H] [propname ...]

     /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 ...]
         [-p repo_uri] [--enable] [--disable] [--no-refresh]
         [--reset-uuid] [--non-sticky] [--sticky]
         [--search-after=publisher] [--search-before=publisher]
         [--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 publisher [-HPn] [publisher ...]

    /usr/bin/pkg history [-Hl] [-n number]
    /usr/bin/pkg purge-history

    /usr/bin/pkg rebuild-index

    /usr/bin/pkg version
    /usr/bin/pkg help
```

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 may make their packages available at one or more repositories. pkg, then, retrieves packages from a publisher's repository and installs them into an image.

A publisher is a forward domain name that can be used to identify a person, group of persons, or an organization as the source of one or more packages. The name of a publisher does not have to be contained within the URIs that identify the locations of publisher repositories. For example, the name of a publisher might be "example.com", but its repositories may be hosted at "example.org" or "example.net".

A repository is a location where clients can publish and retrieve package content (files contained within the package such as programs, documents, etc.) and metadata (information about the package such as its name, description, etc.). As an example, a publisher named "example.org" may have their repository located at the URI "http://example.org/repository".

pkg can also uninstall packages, refresh publisher metadata (such as catalogs), 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, which contain only relocatable packages. (See NOTES on user images.)

pkg(1) attempts to determine, based on its working directory, in what image it has been invoked. If no image metadata can be found in the parent directories, the invocation will fail.

OPTIONS

The following options are supported:

-R dir
: Operate on the image rooted at dir, rather than the one discovered automatically.

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

SUBCOMMANDS

The following subcommands are supported:

image-create [-FPUfz] [--force] [--full|--partial|--user] [--zone]
  [-k ssl_key] [-c ssl_cert] [--no-refresh]
  [--variant <variant_spec>=<instance> ...]
  [-g uri|--origin=uri ...] [-m uri|--mirror=uri ...]
  [--facet <facet_spec>=[True|False] ...]
  (-p|--publisher) [<name>=]<repo_uri> dir

Create, at location given by dir, an image suitable for package operations. The default image type is user, as given by the -U (--user) option. The image type may 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, while 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 will be added when the image is created. If a publisher name is not provided, then all publishers known by the specified repository will be added to the image. An attempt to retrieve the catalog associated with this publisher will be made following the initial creation operations.

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

If the image is to be run within nonglobal zone context, then the -z (--zone) option can be used to set an appropriate filter.

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 publisher metadata (e.g. catalogs).

With --variant, set the specified variant to the indicated value.

With --facet, set the specified facet to the indicated value.

refresh [--full] [publisher ...]

Retrieve updates to the metadata (e.g. catalogs) for each publisher specified. When given no arguments, retrieves updates for each publisher registered within the image.

With --full, retrieve all publisher metadata instead of attempting an incremental update.

install [-nvq] [--accept] [--licenses] [--no-index] [--no-refresh]
  [--deny-new-be | --require-new-be] [--be-name] pkg_fmri_pattern ...

Installs and updates packages to the newest version that match pkg_fmri_pattern allowed by the packages installed in the image.

Some configuration files may be renamed or replaced during the install 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 pkg(5).

With the -n option, execute the requested operation but make no persistent changes to the image.

With the -v option, issue verbose progress messages during the requested operation. With the -q option, issue no progress messages during the requested operation.

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 will fail.

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

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

With --no-refresh, do not attempt to contact the repositories for the image's publishers to retrieve publisher metadata (e.g. catalogs).

With --be-name, rename the newly created boot environment to be the argument given. This option is only valid if a new boot environment is created during the operation. See also beadm(1m).

With --require-new-be, always create a new boot environment. Without this option, a boot environment is created automatically if needed.

With --deny-new-be, disallow creation of a new boot environment; the operation will not be performed if a new boot environment is required.

uninstall [-nrvq] [--no-index] [--deny-new-be | --require-new-be] [--be-name name] pkg_fmri_pattern ...

Removes installed packages that match pkg_fmri_pattern.

With -r, recursively uninstall any packages that contain 'require' dependencies on the initial package. (Packages containing 'optional' or 'incorporate' dependencies are not included in the removal.)

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

update [-fnvq] [--accept] [--be-name name] [--licenses] [--no-index] [--no-refresh] [--deny-new-be | --require-new-be] [pkg_fmri_pattern ...]

With no arguments, or if '*' 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.

If pkg_fmri_pattern is provided, update will replace packages that are installed, and that match pkg_fmri_pattern, with the newest version allowed by the pattern(s) and the constraints imposed on the system by installed packages and publisher configuration. Versions older or newer than what is already installed may be specified to perform in place downgrades or upgrades of specific packages. Please note that 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 will be renamed using the extension '.update'. For more information on how the package

system determines which files to preserve, and how these files are preserved during package upgrades, see "File Actions" in pkg(5).

With the -f option, don't 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.

info [-lr] [--license] [pkg_fmri_pattern ...]
Display information about packages in a human-readable form. Multiple FMRI patterns may be specified; with no patterns, display information on all installed packages in the image.

With -l, use the data available from locally installed packages. This is the default.

With -r, retrieve the data from the repositories of the image's configured publishers. Note that you must specify one or more package patterns in this case.

With --license, print out the license text(s) for the package. This may be combined with -l or -r.

contents [-Hmr] [-a attribute=pattern ...] [-o attribute ...] [-s sort_key] [-t action_type ...] [pkg_fmri_pattern ...]
Display the contents (action attributes) of packages in the current image. By default, only the path attribute is displayed, but the attribute set may be determined with the -o option. The -o option may be specified multiple times, or multiple attributes may be specified as the argument to one -o option by separating the attribute names with commas. Only actions which have the requested attributes will be displayed. The -m option may also be used, as a shorthand for '-Ho action.raw'.

The -a option allows you to limit the output to those actions which have an attribute named in the option argument the value of which 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 will be displayed.

The -s option specifies the attribute by which the listing should be sorted.

The -t option limits the action types which will be displayed.

The -H option causes the headers to be omitted.

The -r option retrieves the requested data from the repositories of the image's configured publishers. This option is intended to be used when the named packages are not already installed.

With no arguments, the output includes all installed packages. Alternatively, multiple FMRI patterns may be specified, which restricts the display to the contents of the matching packages. When using -r, one or more pkg_fmri_patterns must be specified.

Several special "pseudo" attribute names are available for convenience:

action.hash — Corresponds to the value of the action's hash, if the action carries a payload.

action.key — Corresponds to the value of the action's key attribute. For example, for a file

action, this is the path to the file.

action.name — Corresponds to the name of the action. For example, for a file action, this is "file"

action.raw — Corresponds to the complete contents of the action as represented in the package manifest. This corresponds to the lines of output of 'pkg contents -m'

pkg.fmri — Corresponds to the full form FMRI of the package containing the action, such as pkg://extra/virtualbox@3.0.0,5.11-0.101:20090702T175410Z

pkg.name — Corresponds to the name of the package containing the action, such as "SUNWcs"

pkg.publisher — Corresponds to the publisher of the the package containing the action, such as "opensolaris.org"

pkg.shortfmri — Corresponds to the short form FMRI of the package containing the action, such as pkg://opensolaris.org/SUNWzone@0.5.11-0.79

The contents and search subcommands are related: both are used to query the system for the contents of packages. The contents subcommand displays actions in one or more packages, filtering the output based on the options chosen by the user. The search subcommand approaches the query from the other direction, looking for packages which 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 may 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 may include content hashes and pathnames. 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 may be controlled. The -o option may be specified multiple times, or multiple attributes may be specified as the argument to one -o option by separating the attribute names with commas. In addition to the "pseudo" attributes outlined above, more are defined for search results:

search.match — Corresponds to the string which matched the

search query.

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

With -p, display packages which have some actions that match each query term. Using this option is equivalent to putting '<>' around each term in the query. (For a description of the '<>' operator, please see below.)

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 may be specified multiple times.

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

In addition to simple token matching and wildcard search, a more complicated query language is supported. Phrases may be searched for by using ' or ". Note: Please make 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 would match all actions types in all packages with a key of basename and which matched the token 'pkg'. Explicit wildcards are supported in the pkg_name and token fields, action_type and key must match exactly.

To convert actions to the packages which 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 which matched token.

list [-Hafnsuv] [--no-refresh] [pkg_fmri_pattern ...]
Display a list of packages in the current image, including state and other information. By default, package variants for a different architecture or zone type are excluded. The usual output is in four columns:

```
NAME (PUBLISHER)                  VERSION          STATE      UFOXI
SUNWcs                            0.5.11-0.126     installed  -----
web/firefox/plugin/flash (extra)  10.0.32.18-0.111 installed  -----
```

The first column contains the name of the package. If the publisher from which it is installed (or available, if not installed) is not the preferred publisher, 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 pkg(5)). The third column contains the state of the package as it exists on the system. Possible values are "installed" and "known". The last column contains a set of flags that show how the package relates to other packages:

- a "u" in the "U" column shows that a newer version is available, although it may not be possible to install this newer version due to package dependencies or constraints;

- an "f" in the "F" column shows that this version has been frozen (not implemented);

- an "o" in the "O" column shows that it is obsolete,

while an "r" shows that it has been renamed (a form of obsoletion);

- an "x" in the "X" column shows that it is prevented from being installed because some other package has excluded it (not implemented); and

- an "i" in the "I" column shows that it has been constrained by an incorporation (not implemented).

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 is also allowed by any installed incorporations and the image's variants will be 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.

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 description. This option may be used with -a, -n, -u or -v.

With -u, list only packages with newer versions available.

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

With --no-refresh, do not attempt to contact the repositories for the image's publishers to retrieve publisher metadata (e.g. catalogs).

verify [-Hqv] [pkg_fmri_pattern ...]
Validate the installation of packages in the current image. Please note that verification of installed package content is based on a custom content analysis that may return different results than those of other programs.

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.

variant [-H] [<variant_spec> ...]
Display the current values of all variants, or with arguments, only the variants specified

With -H, omit the headers from the listing.

change-variant [-nvq] [--accept] [--be-name name] [--licenses] <variant_spec>=<instance> ...
Change the specified variants in the current image.

With the -n option, plan the requested operation but make

no actual changes.

With the -v option, issue verbose progress messages during the requested operation. With the -q option, be completely silent.

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 will fail.

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

With --be-name, rename the newly created boot environment to be the argument given. This option is only valid if a new boot environment is created during image update. See also beadm(1m).

With --require-new-be, always create a new boot environment. Without this option, a new boot environment is only created if needed.

With --deny-new-be, disallow creation of a new boot environment; the operation will not be performed if a new boot environment is required.

facet [-H] [<facet_spec> ...]

Without arguments, displays the current values of all facets. With argument(s), evaluate if each facet would be true or false and print the result.

With -H, omit the headers from the listing.

change-facet [-nvq] [--accept] [--be-name name] [--licenses] <facet_spec>=[True|False|None] ...

Change the specified facets in the current image.

With the -n option, plan the requested operation but make no actual changes.

With the -v option, issue verbose progress messages during the requested operation. With the -q option, be completely silent.

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 will fail.

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

With --be-name, rename the newly created boot environment to be the argument given. This option is only valid if a new boot environment is created during the operation. See also beadm(1m).

With --require-new-be, always create a new boot environment. Without this option, a new boot environment is only created if needed.

With --deny-new-be, disallow creation of a new boot environment; the operation will not be performed if a new boot environment is required.

Facets may be set to True or False. Setting one to None removes that facet specification from the current image.

fix [--accept] [--licenses] [pkg_fmri_pattern ...]

Fix any errors reported by pkg verify. Please note that verification of installed package content is based on a

custom content analysis that may 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 will fail.

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

set-property propname propvalue
Update an existing image property or add a new image property;
except for preferred-publisher, which can only be changed using
set-publisher.

add-property-value propname propvalue
Add a value to an existing image property or add a new image property;
except for preferred-publisher, which can only be changed using
set-publisher.

remove-property-value propname propvalue
Remove a value from an existing image property; except for
preferred-publisher, which can only be changed using set-publisher.

unset-property propname ...
Remove an existing image property or properties; except for
preferred-publisher, which can only be changed using
set-publisher.

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.

With -H, omit the headers from the listing.

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]
[-p repo_uri] [--enable] [--disable] [--no-refresh]
[--reset-uuid] [--non-sticky] [--sticky]
[--search-after=publisher] [--search-before=publisher]
[--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 an additional 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, set the specified publisher as the preferred
publisher, i.e. first in the search order. When installing
new packages, this publisher will be searched first.
Updates to already installed packages will come from the
same publisher that originally provided the package so long
as that publisher remains sticky.

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

With --sticky, return to the default behavior of always sourcing updates from the same publisher that provided the package originally.

With --search-before, alter the publisher search order so that the publisher being modified is now searched before the specified publisher.

With --search-after, alter the publisher search order so that the publisher being modified is now searched after the specified publisher.

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

With --revoked-ca-cert, treat the certificate with the given hash as revoked. The hashes of the user revoked CA certificates are listed in the output of the detailed pkg publisher view for a publisher.

With --unset-ca-cert, remove the certificate with the given hash from the list of approved 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 URI as an origin for the given publisher. This should be the location of a package repository.

With -G (--remove-origin), remove the URI from the list of origins for the given publisher.

With --no-refresh, do not attempt to contact the publisher specified to retrieve its metadata (e.g. catalog).

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.

With -p, retrieve publisher configuration information from the specified repository URI. If a publisher is specified, then only the matching one will be added or updated. If no publisher is specified, all will be added or updated as appropriate. This option may not 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.

unset-publisher publisher ...
Remove the configuration associated with the given publisher or publishers.

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 the configuration values, including mirrors, associated with those publishers.

With -H, omit the headers from the listing.

With -P, display only the preferred publisher.

With -n, display only enabled publishers.

history [-Hl] [-n number]
Display the command history of the applicable image.

With -H, omit the headers from the listing.

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.

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.

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 PROPERTIES
The following properties are part of the image and may be set using the set-property subcommand. The values of these properties are viewable with the property subcommand.

ca-path
(string) A pathname that points to a directory where CA certs are kept for SSL operations. The format of this directory is specific to the underlying SSL implementation. If the administrator would like to use an alternate location for trusted CA certificates, this value should be changed to point to a different directory. Please see the 'CApath' portions of SSL_CTX_load_verify_locations(3openssl) for requirements about the CA directory.

Default value: /etc/openssl/certs

flush-content-cache-on-success

(boolean) If this is set to True, the package client will remove 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, it will flush its content cache, provided this option has not been changed.

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

Default value: False

mirror-discovery
(boolean) Mirror-discovery tells the client to discover link-local content mirrors using mDNS and DNS-SD. If this is set to True, the client will attempt 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 may disable this option, some network repositories may 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 a package into this 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. The following are the valid values for this property.

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.
require-signatures
Require that all newly installed packages have at least one valid signature. 'pkg fix' and 'pkg verify' will 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 verifiy the chains of trust of the signatures.

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

trust-anchor-directory
(string) The pathname of the directory that contains the trust anchors for the image. This path is relative to the image.

PUBLISHER PROPERTIES
The following properties are part of the image and may be set using

the set-property option of the set-publisher subcommand.

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

EXAMPLES

Example 1: 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 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: Install the latest version of the widget package in the current image.

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

Example 4: List the contents of the SUNWzfs 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, will display 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 SUNWzfs
NAME  MODE   SIZE PATH                                TARGET
dir   0755        etc
dir   0755        etc/fs
dir   0755        etc/fs/zfs
link              etc/fs/zfs/mount                   ../../../sbin/zfs
link              etc/fs/zfs/umount                  ../../../sbin/zfs
dir   0755        etc/zfs
dir   0755        lib
dir   0755        lib/amd64
link              lib/amd64/libzfs.so                libzfs.so.1
file  0755 469616 lib/amd64/libzfs.so.1
file  0644  62057 lib/amd64/llib-lzfs.ln
link              lib/libzfs.so                      libzfs.so.1
....
```

Example 5: List the contents of SUNWfirefox and SUNWthunderbird, limiting the display to just the package name and path attributes of actions whose "path" attribute ends in ".desktop" or ".png".

```
$ pkg contents contents -o pkg.name,path -a path=\*.desktop \
    -a path=\*.png SUNWfirefox SUNWthunderbird
PKG.NAME        PATH
SUNWfirefox     usr/lib/firefox/chrome/icons/default/default16.png
SUNWfirefox     usr/lib/firefox/chrome/icons/default/default32.png
```

```
SUNWfirefox     usr/lib/firefox/chrome/icons/default/default48.png
SUNWfirefox     usr/lib/firefox/icons/document.png
SUNWfirefox     usr/lib/firefox/icons/mozicon128.png
SUNWfirefox     usr/lib/firefox/res/html/folder.png
SUNWfirefox     usr/share/applications/firefox.desktop
SUNWthunderbird usr/share/applications/thunderbird.desktop
SUNWfirefox     usr/share/pixmaps/firefox-icon.png
SUNWthunderbird usr/share/pixmaps/thunderbird-icon.png
```

Example 6: Search the package database for the token "bge".

```
$ pkg search bge
INDEX       ACTION VALUE                    PACKAGE
basename    file   kernel/drv/bge           pkg:/SUNWbge@0.5.11-0.79
driver_name driver bge                      pkg:/SUNWbge@0.5.11-0.79
```

The token shows up in the package SUNWbge both as the basename for the file action representing /kernel/drv/bge and as a driver name.

Example 7: Search for installed packages which depend on SUNWipkg.

```
$ pkg search -l 'depend::SUNWipkg'
INDEX       ACTION VALUE                    PACKAGE
incorporate depend SUNWipkg@0.5.11-0.111    pkg:/entire@0.5.11-0.111
require     depend SUNWipkg@0.5.11-0.111    pkg:/slim_install@0.1-0.111
require     depend SUNWipkg@0.5.11-0.111    pkg:/SUNWipkg-brand@0.5.11-0.111
```

Example 8: Search for all incorporate dependencies in installed packages.

```
$ pkg search -l 'depend:incorporate:'
INDEX       ACTION VALUE                    PACKAGE
incorporate depend BRCMbnx@0.5.11-0.111     pkg:/entire@0.5.11-0.111
incorporate depend BRCMbnx@0.5.11-0.111     pkg:/entire@0.5.11-0.111
....
```

Example 9: Add new publisher example.org, with a repository located at http://www.example.org/repo:

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

Example 10: Add new publisher example.com, with a secure repository located at https://secure.example.com/repo, and a key and cert 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 11: Add new publisher with a repository located at /export/repo using automatic configuration:

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

Example 12: Add new publisher example.org with a repository located at /export/repo/example.com using manual configuration:

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

Example 13: Configure an image to verify all signed packages.

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

Example 14: Configure an image to require all packages to be signed and the string "opensolaris.org" has to be seen as a common name for one of the certificates in the chain of trust.

```
$ pkg set-property signature-policy require-names opensolaris.org
```

Example 15: Configure an image so that all packages installed from publisher foo must be signed.

```
$ pkg set-publisher --set-property signature-policy=require-signatures
```

Example 16: 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 17: Remove the string "foo" from publisher test's list of common names that must be seen to validate a signature.

```
$ pkg set-publisher --remove-property-value signature-require-names=foo \
    test
```

Example 18: Add the certificate stored in /tmp/example_file.pem as a trusted CA certificate for the publisher test.

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

Example 19: Revoke the certificate with the hash a12345 for publisher test, preventing it from validating any signatures for packages from test.

```
$ pkg set-publisher --revoke-ca-cert a12345
```

Example 20: Make pkg forget that the certificate a12345 was ever added or revoked by the user.

```
$ pkg set-publisher --unset-ca-cert a12345
```

Example 21: Downgrade the installed package foo@1.1 to an older version:

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

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 as 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.

FILES

A pkg(5) image can be located arbitrarily within a larger file system. In the following, 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. We use the token $IMAGE_META to refer 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.

$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	SUNWipkg pkg:/package/pkg
Interface Stability	None / Under Development

SEE ALSO

pkgsend(1), pkg.depotd(1M), glob(3C), attributes(5), pkg(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject to change. Development is hosted in the OpenSolaris community at:

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

At present, user images are not restricted to relocatable packages--but they will be.

The pkg(1) command recognizes use of the http_proxy and https_proxy environment variables to select a suitable HTTP or HTTPS proxy server. At present, particular care is needed when using local repository URIs--such as http://localhost:10000/--with the http_proxy environment variable; this behavior may change in a future version of image packaging.

At present, pkg(1), on directory removal, will move unpackaged contents of that directory to $IMAGE_META/lost+found.

pkgdepend(1)

NAME

pkgdepend - image packaging system dependency analyzer

SYNOPSIS

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

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

/usr/bin/pkgdepend resolve [-dmosSv] manifest_path ...
```

DESCRIPTION

pkgdepend(1) is used to generate and resolve dependencies for packages. A package may depend upon files from other packages. pkgdepend(1) is typically used in two passes: file dependency generation and file-to-package resolution.

The generate subcommand examines a package's content and discovers what external files the package needs.

The resolve subcommand takes the list of files from the generate step, and figures out the name of the packages, limited to some reference set of packages, that contain the dependent files. Currently, the reference set of packages is defined as the packages currently installed on the publisher's system.

Several aspects 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, please see ldd(1) and the Solaris Linker and Libraries Guide.
Scripts	Shell scripts that contain '#!' lines referencing an interpreter will 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 may also serve as sources of dependency information.
Hard links	Hard links in manifests will result in a dependency on the package that delivers the link target.
SMF	SMF service manifests delivered that include "require_all" dependencies will 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, rather than the one discovered automatically.

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

SUBCOMMANDS

The following subcommands are supported:

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

With -I, the dependencies which are satisfied within the manifest given will be shown.

With -M, a list of those file types which could not be analyzed will be displayed.

With -m, the original manifest will be repeated with any discovered dependencies added after.

With -d, add the directory provided 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 given to the list of paths in which to look for kernel modules. Using the -k argument removes the default paths which are /kernel and /usr/kernel.

resolve [-mov] [-d output_dir] [-s suffix] manifest_path ...
Transform dependencies on files into dependencies on the packages which 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 whose name is determined by appending ".res" to the manifest's path.

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.

With -d, write the resolved dependencies for each manifest provided in a separate file in output_dir. By default, each file will have the same basename as the manifest that was the source of the dependencies written to that file.

With -s, for each output file, append the given suffix (a "." will be added to the argument if not provided) to the basename of the file that was the source of the resolved dependencies.

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

With -v, include additional package dependency debugging metadata.

EXAMPLES

Example 1: 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 foo ./bar/baz > foo.fdeps
```

Example 2: Resolve the file dependencies in foo.fdeps, and bar.fdeps against each other, and the packages currently installed on the system.

```
$ pkgdepend resolve foo.fdeps bar.fdeps
```

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

Example 3: 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, which could be used with pkgsend publish, in ./res

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

Example 4: Replace all $PLATFORM tokens in the run paths in elf files with sun4v and sun4u while generating the dependences for the manifest written in foo whose content directory is in /.

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

Example 5: Use /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 -k /kmod foo /
```

EXIT STATUS

The following exit values are returned:

0 Everything worked.

1 Something bad happened.

2 Invalid command line options were specified.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWipkg pkg:/package/pkg
Interface Stability	None / Under Development

SEE ALSO

attributes(5), pkg(5)

pkgdiff(1)

NAME

pkgdiff - compare pkg manifests

SYNOPSIS

/usr/bin/pkgdiff [-i attribute] [-o attribute] ... file1 file2

DESCRIPTION

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

Output is of the form:

+ <complete action> if this action is in file2 but not in file1
- <complete action> if this action is in file1 but not in file2

actionname keyvalue <variant values, if any>
- attribute1=value1 if this attribute,value is in file1 but not in file2
+ attribute2=value2 if this attribute,value is in file2 but not in file1

Note that actions with different variants but the same type and key attribute value are treated as separate actions for purposes of comparison, so actions changing attributes will appear in their complete form rather than as attribute changes.

OPTIONS

-i attribute Ignore this attribute if present during comparisons File hash values may be ignored with -i hash. May not be used with -o option.

-o attribute Only report differences in this attribute. May not be used with -i option. Will elide any action changes that don't affect this attribute on an action.

EXIT STATUS

The following exit values are returned:

0 No differences were found

1 Differences were found

>1 An error occurred

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWipkg pkg:/package/pkg
Interface Stability	None / Under Development

SEE ALSO

attributes(5), pkg(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject

to change. Development is hosted in the OpenSolaris community at

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

pkgfmt(1)

NAME

pkgfmt - format a package manifest

SYNOPSIS

/usr/bin/pkgfmt [package-manifest-file]

DESCRIPTION

Pkgfmt without the -c option will format a package manifest in a consistent manner, wrapping lines at 80 characters, sorting actions by type, placing key attributes first, etc. Pkgfmt tries to be robust in the case of lines that don't parse into actions; such lines will not appear in sorted order.

If no arguments are given, pkgfmt will read the stdin until EOF, and then write the formatted manifest on the stdout. Any manifests specified on the command line are formatted in place.

Pkgfmt with the -c option will check to make sure the manifest(s) are formatted in pkgfmt style.

OPTIONS

-c Check if the manifest is formatted in the pkgfmt style.

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

EXIT STATUS

The following exit values are returned:

0 Normal execution

1 The -c option was specified, and one or more manifests are not in pkgfmt-normal form.

99 An unanticipated exception occurred

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	pkg:/package/pkg
Interface Stability	None / Under Development

SEE ALSO

attributes(5), pkg(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject to change. Development is hosted in the OpenSolaris community at

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

pkglint(1)

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(1) 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 invoking pkgsend(1) on a set of manifests to perform exhaustive testing on the packages that may be too expensive to perform during normal operation of pkgsend(1) or pkg.depotd(1M). The 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 can be retrieved from a repository.

 If using repositories, on first-run, pkglint will create and populate pkg(5) user-images in the supplied cache directory, one for the reference repository, if the -r option is supplied, and one for the lint repository, if the -l option is supplied. No content is installed in these images, they are only used by pkglint(1) to retrieve manifests from the repositories. The reference and lint images are named <cache_dir>/ref_image and <cache_dir>/lint_image, respectively.

 Subsequent invocations of pkglint(1) can re-use the cache directory and can omit any -r or -l arguments.

 Limited support is provided in pkglint(1) for configuring publishers in the cache directory. It is assumed that pkg(1) will be used on these images to perform more complex publisher configuration.

 pkglint(1) allows package authors to bypass checks for a given manifest or action. Manifests or actions containing an attribute "pkg.linted" set to "True" will not produce any lint output for that manifest or action.

 The behavior of pkglint can be changed using a supplied pkglintrc file. By default, the tool searches in /usr/share/lib/pkg/pkglintrc and $HOME/.pkglintrc for configuration options. The command line option -f can be used 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 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 ver-

sions of packages are used. See also the "version.pattern" configuration property.

-c cache_dir
: A local directory used for caching package metadata from the lint and reference repositories.

-l lint_uri
: A URI representing the location of the lint repository. Both HTTP and filesystem-based publication are supported.

-L
: List the known and excluded lint checks, showing the short name of each check, and its description then exit. When combined with the -v flag, the method that implements the check is printed instead of the description.

-f config_file
: Configure the pkglint session using the config file provided.

-p regexp
: A regular expression used to narrow the list of packages from the lint repository that are to be checked. Note that all manifests from the reference repository are loaded (assuming they match the value for -b, if supplied) ignoring this pattern.

-r repo_uri
: A URI representing the location of the reference repository.

-v
: Run the tool in a verbose mode, overriding any log_level settings in the config file.

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

CONFIGURATION FILE

The pkglintrc config file takes a number of key/value arguments, which are listed below:

log_level
: The minimum level at which to emit lint messages. Lint messages lower than this level are discarded.

  By default, this is set to INFO. Log levels in order of least to most severe are:

  DEBUG, INFO, WARNING, ERROR, CRITICAL

do_pub_checks
: Whether to perform checks which may only make sense for published packages. Set to True by default.

pkglint.ext.* The plugin mechanism of pkglint allows for additional lint modules to be added at runtime. Any key starting with "pkglintext" takes a value that must be a fully-specified Python module. (See section "Developers", below)

pkglint.exclude A space-separated list of fully specified Python modules, classes or function names which should be omitted from the set of checks performed.

use_progress_tracker Whether to use a progress tracker when iterating over manifests during lint runs, set to True by default.

version.pattern A version pattern, used when specifying a build number to lint against (-b). If not specified in the rcfile, this pattern is "*,5.11-0.", matching all components of the '5.11' build, with a branch prefix of '0.'

DEVELOPERS

Developers wishing to extend the set of checks performed should subclass pkg.lint.base.Checker and its subclasses, ManifestChecker, ActionChecker and ContentChecker. The Python module name containing those classes should be added to a new "pkglint.ext." key in the configuration file.

Instances of those new subclasses are created by pkglint(1) on startup, and 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.

EXAMPLES

Example 1: Running a pkglint session for the first time.

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

Example 2: A subsequent run against the same repository used in Example 1.

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

Example 3: Running a pkglint session with a lint repository, specifying a subset of packages to run checks on.

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

Example 4: Running a pkglint session against a given build in verbose mode.

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

Example 5: 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.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	pkg:/package/pkg
Interface Stability	None / Under Development

SEE ALSO

pkg(1), pkg.depotd(1M), pkgsend(1), pkg(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject to change. Development is hosted in the OpenSolaris community at:

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

pkgmogrify(1)

NAME
pkgmogrify - image packaging system manifest transmogrifier

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

DESCRIPTION
pkgmogrify(1) provides for programmatic editing of package manifests to simplify the typical transformations needed when automating software builds and package publication.

pkgmogrify(1) 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 pathnames, such as the name of the 64-bit architecture directory for executables and libraries:

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

Again, the expectation is that these macros would 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 out comments into output manifest showing effect of transforms to aid debugging.

--help or -?
Displays a usage message.

EMBEDDED DIRECTIVES

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

Include directives are of the form:

```
<include file>
```

This causes pkgmogrify(1) 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(1) exits with an error.

Transform directives are of the form:

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

These 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; more information is available via the command "pydoc re" or (more completely) at http://docs.python.org/dev/howto/regex.html. The regular expression is anchored at the beginning, but not the end, so the regular expression matching files by their extensions would have to include a leading ".*" and, for safety, a trailing "$".

Multiple criteria may be specified, separated by spaces.

The following operations are available:

add Add a value to an attribute. Two arguments are taken. The first 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. The same two arguments are taken as for "add".

delete Remove attribute values. Two arguments are taken. The first is the name of the attribute, and the second 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. Three arguments are

taken. The first is the name of the attribute, the second is a regular expression matching the attribute value, and the third 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', etc., are available in the replacement string, if there are groups defined in the regular expression.

emit Emit a line to the manifest output stream. This must be a valid action string, empty (which will result 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. The same two arguments are taken as for "add".

All operations aside from "delete" and "drop" take (possibly optional) arguments whose contents will end up in the output stream. These strings can contain three different kinds of special tokens which allow the output to contain information that isn't 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(1): pkg.manifest.filename and pkg.manifest.lineno. These refer to the name of the file in which the action was found and the line on which it was found, respectively. Three others are similar to ones used in pkg(1): action.hash, action.key, and action.name, which refer to the hash value of the action if the action carries a payload, the value of the key attribute, and the name of the action type, respectively. For actions with payloads, the "set" operation can change the hash of the action by operating on the action.hash attribute.

If the attribute whose value is requested doesn't exist, pkgmogrify(1) will exit with an error. To prevent this, the attribute name may be followed by ";notfound=" and a value to substitute in its place. For example "%(alias;notfound='no alias')" will print the value of the attribute "alias" if it exists, and "no alias" otherwise.

If the attribute whose value is requested is multi-valued, each value will be printed, separated by spaces. Similarly to the "notfound" token, the tokens "prefix", "suffix, and "sep" may 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(1) directives may 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(1)) for use in these situations.

When pkgmogrify(1) finishes reading a manifest specified on the commandline and that manifest defined a "pkg.fmri" package attribute, it will create this synthetic "pkg" action, whose attributes are the package's attributes. A <transform> directive may then match on this action, just like any other action type.

Operations on a "pkg" action are special, though, 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 will 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 will cause an error. If you wish to add a package attribute, emit a "set" action instead.

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

The order of processing is as follows:

1) Lines are read from input file(s).
2) Macros are applied.
3) <include ...> and <transform> directives are processed causing additional files to be found and read in.
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 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 from usr/sfw/bin to usr/bin.

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

Example 3: Add "reboot-needed" tags to files under /kernel which aren't .conf files. This can also be done in a single transform rule with regular expression magic, but this formulation will make more sense to most people. Note that this 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 .*>
```

Example 4: 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 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: Safely print a message even when an attribute is missing.

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

Example 7: Set some default permissions.

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

Example 8: For any package which isn't marked obsolete, add a dependency on the incorporation for the consolidation that delivers it. 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: 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 locale facet appropriate for the pathname 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

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWipkg pkg:/package/pkg
Interface Stability	None / Under Development

SEE ALSO

pkg(1), attributes(5), pkg(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject

to change. Development is hosted in the OpenSolaris community at

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

pkgrecv(1)

NAME
pkgrecv - image packaging system content retrieval utility

SYNOPSIS
/usr/bin/pkgrecv [-s src_uri] [-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 the contents of a package from a pkg(5) repository. With the default options, the contents are retrieved in a repository format suitable for use with pkg.depotd(1M).

OPTIONS
The following options are supported:

-c cache_dir The path to a directory that will be used to cache downloaded content. If one is not supplied, the client will automatically pick 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.

-d path_or_uri The filesystem path or URI of the target repository to republish packages to. If not provided, the default value is the current working directory. The target must already exist. New repositories can be created using pkgrepo(1).

-h Display usage message.

-k Keep the retrieved package content compressed, ignored when republishing. Should not be used with pkgsend(1).

-m match Controls matching behavior using the following values:
all-timestamps
includes all matching timestamps, not just latest (implies all-versions)
all-versions
includes all matching versions, not just 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 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 will be 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. May only be used with filesystem-based destinations. This can be used with pkgsend(1) include to conveniently modify and republish packages, perhaps by correcting file contents or providing additional package metadata.

EXAMPLES

Example 1: List newest packages available from the repository on the system named 'test'.

```
$ pkgrecv -s http://test --newest
pkg:/SUNWlibC@0.5.11,5.11-0.79:20080221T125720Z
pkg:/SUNWfreetype2@0.5.11,5.11-0.79:20080221T123955Z
pkg:/SUNWlibm@0.5.11,5.11-0.79:20080221T125728Z
pkg:/SUNWliboil@0.5.11,5.11-0.79:20080221T125730Z
```

Example 2: Receive the SUNWlibC, SUNWfreetype, and SUNWlibm packages from example 1 in a format suitable for use with pkgsend(1) include.

```
$ pkgrecv -s http://test SUNWlibC@0.5.11,5.11-0.79:20080221T125720Z \
    --raw
SUNWfreetype2@0.5.11,5.11-0.79:20080221T123955Z
SUNWlibm@0.5.11,5.11-0.79:20080221T125728Z
$ ls -d SUNW*
SUNWfreetype2  SUNWlibC       SUNWlibm
```

Example 3: Receive the package "SUNWvim" and all of its dependencies from the system named 'test'.

```
$ pkgrecv -s http://test -r SUNWvim
```

Example 4: Receive all versions of the package "SUNWvim" from the system named 'test'.

```
$ pkgrecv -s http://test -m all-versions SUNWvim
```

Example 5: Receive all versions of the package "SUNWvim" from the system named 'test' and republish it to a local repository.

```
$ pkgrecv -s http://test -d /local/repo SUNWvim
```

Example 6: Receive all versions of the package "SUNWzlib" 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 SUNWzlib
```

Example 7: Receive the package "SUNWemacs" and all of its dependencies from the repository located at '/var/pkg/repo'.

```
$ pkgrecv -s /var/pkg/repo -r SUNWemacs
```

Example 8: 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 '*'
```

ENVIRONMENT VARIABLES

The following environment variables are supported:

PKG_DEST — The path of a directory to save the retrieved package to, or the URI of a repository to republish it to. If not provided, the default value is the current working directory.

PKG_SRC — A URI representing the location of a pkg(5)

repository to retrieve package data from.

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.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	

SEE ALSO

pkgrepo(1), pkgsend(1), attributes(5), pkg(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject to change. Development is hosted in the OpenSolaris community at

http://opensolaris.org/os/project/pkg/

pkgrepo(1)

NAME
     pkgrepo - image packaging system repository management utility

SYNOPSIS
     pkgrepo create [--version] uri_or_path

     pkgrepo add-signing-ca-cert [-p publisher ...]
         [-s repo_uri_or_path] path ...

     pkgrepo add-signing-intermediate-cert [-p publisher ...]
         [-s repo_uri_or_path] path ...

     pkgrepo get [-p publisher ...] [-s repo_uri_or_path]
         [section/property ...]

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

     pkgrepo rebuild [-s repo_uri_or_path] [--no-catalog]
         [--no-index]

     pkgrepo refresh [-s repo_uri_or_path] [--no-catalog]
         [--no-index]

     pkgrepo remove-signing-ca-cert [-p publisher ...]
         [-s repo_uri_or_path] hash ...

     pkgrepo remove-signing-intermediate-cert [-p publisher ...]
         [-s repo_uri_or_path] hash ...

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

     pkgrepo help
     pkgrepo version

DESCRIPTION
     pkgrepo provides the ability to create and manage pkg(5) package
     repositories. Package repositories are a pre-defined 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.

SUBCOMMANDS
     The following subcommands are supported:

     create [--version] <uri_or_path>
          May only be used with filesystem-based repositories.

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

          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-signing-ca-certs [-p publisher ...]
[-s repo_uri_or_path] path ...
May only be used with filesystem-based repositories.

Add the certificates provided as approved CA certificates.

With -p, only add certificate data for the given publisher. The special value 'all' may be used to add the certificate data for all publishers. If not specified, the default publisher will be used.

With -s, operate on the repository located at the given URI or filesystem path.

add-signing-intermediate-certs [-p publisher ...]
[-s repo_uri_or_path] path ...
May only be used with filesystem-based repositories.

Add the certificates provided as intermediate certificates.

With -p, only add certificate data for the given publisher. The special value 'all' may be used to add the certificate data for all publishers. If not specified, the default publisher will be used.

With -s, operate on the repository located at the given URI or filesystem path.

get [-F format] [-H] [-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 quotes (""). Bourne shell meta-characters (';', '&', '(', ')', '|', '^', '<', '>', newline, space, tab, backslash, '"', single-quote, '`') in ASCII string values are quoted by backslashes (\).

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

With -H, omit the headers from the listing.

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

With -s, operate on the repository located at the given URI or filesystem 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. Currently, only 'tsv' (Tab Separated Values) is valid.

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 will be displayed. This option may be specified multiple times.

With -s, operate on the repository located at the given URI or filesystem path.

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

With -s, operate on the repository located at the given URI or filesystem path.

With --no-catalog, don't rebuild package data.

With --no-index, don't rebuild search indexes.

refresh [-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 -s, operate on the repository located at the given URI or filesystem path.

With --no-catalog, don't add any new packages.

With --no-index, don't update search indexes.

remove-signing-ca-certs [-p publisher ...]
[-s repo_uri_or_path] hash ...
May only be used with filesystem-based repositories.

Remove the certificates provided from the list of CA certificates.

With -p, only remove certificate data for the given publisher. The special value 'all' may be used to remove the certificate data for all publishers. If not specified, the default publisher will be used.

With -s, operate on the repository located at the given URI or filesystem path.

remove-signing-intermediate-certs [-p publisher ...]
[-s repo_uri_or_path] hash ...
May only be used with filesystem-based repositories.

Remove the certificates provided from the list of intermediate certificates.

With -p, only remove certificate data for the given publisher. The special value 'all' may be used to remove the certificate data for all publishers. If not specified, the default publisher will be used.

With -s, operate on the repository located at the given URI or filesystem path.

```
     set [-p publisher] [-s repo_uri_or_path]
       [section/property=value ...]
          May only be used with filesystem-based repositories.

          Sets the value of the specified properties for the
          repository or publishers.

          With -p, only set property data for the given publisher.
          The special value 'all' may be used to set the property
          for all publishers.

          With -s, operate on the repository located at the given URI
          or filesystem path.

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

               section/property=
                    The property's value will be cleared.

               section/property=value
                    The property's value will replaced with the given
                    value.

               section/property=(value1 value2 valueN)
                    The property's value will be replaced with the
                    list of values.

     version
          Display a unique string identifying the version of the
          pkg(5) system.  The values produced by the version
          operation are not sortable, or safe for comparison
          beyond equality.

EXAMPLES
     Example 1:  Create a package repository.

     $ pkgrepo create /my/repository

     Example 2:  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 May 22, 2010 12:06:03 PM

     $ pkgrepo info -s http://example.com/repository
     PUBLISHER   PACKAGES STATUS UPDATED
     example.com 5        online May 22, 2010 12:06:03 PM

     Example 3:  Rebuild the repository's catalogs and search data.

     $ pkgrepo rebuild -s /my/repository

     Example 4:  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
     PUBLISHER SECTION    PROPERTY VALUE
               publisher  prefix   example.com
```

Example 6: Display all publisher properties.

```
$ pkgrepo get -s /my/repository -p all
PUBLISHER   SECTION    PROPERTY VALUE
example.com repository origins  http://example.com/repository
example.net repository origins  http://example.net/repository
```

Example 7: Set a repository property.

```
$ 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 the certificate stored in /tmp/example_file.pem as a signing CA certificate for the repository located at /my/repository for the default publisher.

```
$ pkgrepo add-signing-ca-cert -s /my/repository \
    /tmp/example_file.pem
```

Example 10: Add the certificate stored in /tmp/example_file.pem as a signing CA certificate for the repository located at /my/repository for a specific publisher.

```
$ pkgrepo add-signing-ca-cert -s /my/repository \
    -p example.com /tmp/example_file.pem
```

Example 11: Remove the certificate with hash a12345 from the list of intermediate certificates for the repository located at /my/repository for the default publisher.

```
$ pkgrepo remove-signing-intermediate-ca-cert -s /my/repository \
    a12345
```

Example 12: Remove the certificate with hash a12345 from the list of intermediate certificates for the repository located at /my/repository for a specific publisher.

```
$ pkgrepo remove-signing-intermediate-ca-cert -s /my/repository \
    -p example.com a12345
```

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.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	pkg:/package/pkg
Interface Stability	None / Under Development

SEE ALSO

pkg(1), pkgrecv(1), pkgsend(1), pkg.depotd(1M), pkg(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject to change. Development is hosted in the OpenSolaris community at:

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

pkgsend(1)

NAME
pkgsend - image packaging system publication client

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

/usr/bin/pkgsend refresh-index

/usr/bin/pkgsend open [-en] pkg_fmri
/usr/bin/pkgsend add action arguments
/usr/bin/pkgsend include [-d basedir] ... [-T pattern] [manifest] ...
/usr/bin/pkgsend publish [-d basedir] ... [-T pattern] [--no-index]
[--fmri-in-manifest | pkg_fmri] [--no-catalog] [manifest] ...
/usr/bin/pkgsend close [-A | --no-index] [--no-catalog]

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

DESCRIPTION
pkgsend allows the publication of new packages and new package versions to an image packaging repository. Each publication is structured as a transaction; multiple invocations of pkgsend through a transaction allow the addition of resources and the transaction submission. To create or manage repositories, see pkgrepo(1).

OPTIONS
The following options are supported:

-s repo_uri A URI representing the location of the target repository. Both HTTP and filesystem-based publication are supported. The default value is http://localhost:10000/.

For filesystem-based publication, the repo_uri should be specified as follows:

file://<absolute_path_to_repository>

See the NOTES section below for more information about restrictions or suggestions for publication.

--help or -?
Displays a usage message.

SUBCOMMANDS
The following subcommands are supported:

open [-en] pkg_fmri
Begin a transaction on the package and version specified by pkg_fmri.

By default, or if the -e option is given, a successfully opened transaction's ID will be published in a form suitable for use with eval(1), to set the PKG_TRANS_ID environment variable. This form can be convenient for delivering a package transaction from a shell script, for example. If the -n option is given, the transaction ID is displayed as a string; this may be more useful in situations where shell scripting is not an available option.

add action arguments
Add a resource associated with an action to the current

transaction. Requires transaction context. See ACTIONS below.

import [-T pattern] [--target file] bundlefile ...
Add each given bundlefile into the current transaction. Supported bundle types are:

- filesystem format SVr4 packages
- datastream format SVr4 packages
- tar files
- directories

If the basename of files in the bundle match the pattern(s) 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 bundle is a directory, there is no clear way to distinguish a file action from a hardlink action when there are multiple pathnames for a single inode. Normally, the first one found in the filesystem walk is treated as a file and the rest as hardlinks. This may be arbitrary, depending on the implementation of the filesystem. To specify which pathnames should be treated as files, pass each pathname as an argument to the --target option. This option has no effect on other bundle types, as they are capable of expressing which pathnames are files and which are hardlinks.

include [-d basedir] ... [-T pattern] [manifest] ...
Add resources associated with the multiple actions present in each manifest file to the current transaction. If no files are specified, the standard input is read. Each line in the file should be the string representation of an action. In particular, the "add" token as described above should not be present, nor should there be open and close tokens representing transaction boundaries. For those actions with datastreams, the path to the file containing the data should be the second word on the line; for file actions if this is set to "NOHASH" the value of the "path" attribute is used instead.

If the user specifies the -d option, basedir is prepended to the search path when locating files in the manifest. Multiple -d options have their directories searched in the order they appear on the command line.

The -T option works as in the import command.

close [-A] [--no-index]
Close current transaction. With -A, abandon the current transaction. With --no-index, do not update the repository's search indices. With --no-catalog, do not add to catalog which for now is only effective with file:// scheme repositories. This improves performance when publishing many packages at once; the repository will need to be started with a --add-content flag (see pkg.depotd(1M) man page) to catalog and index the new content.

publish [-d basedir] ... [-T pattern] [--no-index]
[--fmri-in-manifest | fmri] [--no-catalog] [manifest] ...

Combines open, include and close in a single operation. -d and --no-index options behave as described for include and close subcommands. With --fmri-in-manifest, use the fmri

specified in the package manifest w/ a set action name of "pkg.fmri". Otherwise the fmri must be specified on the command line. With --no-catalog (which has effect only w/ file:// scheme repositories), do not update the catalog with the new package. This improves performance when publishing many packages at once; the repository will need to be started with a --add-content flag (see pkg.depotd(1M) man page) to catalog and index the new content.

The -T option works as in the import command.

generate [-T pattern] [--target file] bundlefile ...
Read each given bundlefile (such as an SVR4 package, directory, tarfile, etc) and emit the manifest describing the bundlefile to the stdout. The -T and --target options work as in the import command. The repo argument or PKG_TRANS_ID is not used with this subcommand. Once obtained, the manifest can be annotated, have dependencies analyzed, added, etc before being passed to the publish or import subcommands.

refresh-index
Update the repository's search indices. Should be used after closing one or more transactions using --no-index.

ENVIRONMENT VARIABLES

The following environment variables are supported:

PKG_TRANS_ID	Identifier to use for this transaction. If undefined (and no alternative means of specifying a transaction is given), subcommands requiring transaction context will fail.

ACTIONS

Each resource within a package must be associated with an action. See pkg(5) for a complete list of actions.

If an action has an associated payload, the path to the payload must be the first argument after the action name. (At present, the file and license actions have payloads.) All other attributes are specified as a list of name-value pairs, and may be given in any order. An action must always have a key attribute; however, some actions may require additional attributes to work correctly. Arbitrary attributes, beyond those defined for a given action, may be included in the action. Such attributes are expected to follow defined conventions such that they carry appropriate meaning or avoid collision with attributes from other action providers; specific attributes that cause additional operations, known as "actuators", are documented in pkg(5).

depend type=<depend_type> fmri=<pkg_fmri> [property_list]
Make this package version dependent on the give package FMRI. Valid depend_types are require and optional.

dir mode=<mode> owner=<userid> group=<groupid> path=<path> \
[property_list]
Deliver a directory with the listed attributes into the transaction.

driver name=<name> perms=<perms> class=<class> alias=<alias> \
[property_list]
Deliver driver configuration with the listed attributes into the transaction. Multiple alias entries may be given to record multiple device aliases.

file src_path mode=<mode> owner=<userid> group=<groupid> \

```
   path=<path> [ property_list ]
     Deliver a file with the listed attributes into the
     transaction.

group groupname=<groupname> gid=<gid> [ property_list ]
     Deliver a group definition as defined in group(4).  If the
     group ID, or gid, is omitted, the first free group under 100
     will be assigned at installation time.  See pkg(5) for
     additional details, and relation to the user action below.

hardlink path=<path> target=<target> [ property_list ]

  Deliver a link with given path, pointing to the given
     target.

legacy category=<category> desc=<description> hotline=<hotline> \
  name=<name> pkg=<legacy_pkg> vendor=<vendor> version=<version> \
  [ property_list ]
     Deliver sufficient metadata, as given by the various fields,
     to represent a System V package with name given by legacy_pkg.

license src_path license=<license_name> \
  [ property_list ]
     Deliver, into the image packaging metadata, the license file
     at src_path, labeled by the given license value.

link path=<path> target=<target> [ property_list ]
     Deliver a symbolic link with given path, pointing to the given
     target.

set name=<name> value=<value> [ property_list ]
     Deliver a "package property" defined by the given name and
     value.

user username=<username> password=<password> uid=<uid> \
  group=<group> gcos-field=<gcos> home-dir=<homedir> \
  login-shell=<login-shell> group-list=<group> \
  [ group-list=<group> ... ] ftpuser={true|false} \
  lastchng=<lastchange> min=<mindays> max=<maxdays> \
  warn=<warndays> inactive=<inactivedays> expire=<expiredate> \
  flag=<flag> [ property_list ]
     Deliver a user account, as defined in passwd(4), shadow(4),
     group(4), and ftpusers(4).  If the user ID, or uid, is
     omitted, then the first free user ID under 100 will be
     assigned at installation time.  The primary group, specified
     by group, must exist prior to the action's execution; this
     condition can be ensured by having a group action deliver the
     group definition in the same package as the user action, or by
     having a depend action that expresses a requirement on a
     package possessing a group action.  The group-list property
     delivers one group to the group(4) entry for the specified
     user.  Multiple groups are delivered by multiple assignments
     to group-list.  See pkg(5) for additional details.

EXAMPLES
     Example 1:  Create a trivial package.

     $ eval `pkgsend open example@1.0-1`
     $ pkgsend add file example mode=0555 owner=root group=bin \
         path=/usr/bin/example
     $ pkgsend close

     Example 2:  Publish a package using filesystem-based publication
         and a pre-existing manifest:

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

Example 3: Create a package that delivers two groups and a user.

```
$ eval `pkgsend open groups-and-user@1.0-1`
$ pkgsend add group groupname=maingrp
$ pkgsend add group groupname=auxgrp
$ pkgsend add user ftpuser=false gcos-field="Example User" \
    group=maingrp group-list=auxgrp password=NP username=exuser
$ pkgsend close
```

EXIT STATUS

The following exit values are returned:

0 Command succeeded.

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	

SEE ALSO

eval(1), pkgrepo(1), pkg.depotd(1M), ftpusers(4), group(4), passwd(4), shadow(4), attributes(5), pkg(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject to change. Development is hosted in the OpenSolaris community at:

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

Other package bundle formats can be created. Other forms of package publication, via the underlying Python API or via the web API, are also possible.

When publishing individual package files that are greater than 128MB in size, filesystem-based publication must be used due to publication protocol limitations. It is also the recommended method of publication when access control for a repository is needed.

When using filesystem-based publication, any pkg.depot(1M) processes 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.

pkgsign(1)

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]
    [-s src_uri] [--help] [--no-index] [--no-catalog] [--sign-all |
    fmri_with_timestamp ...]
```

DESCRIPTION

pkgsign updates the manifest for the given fmri(s) in place in the repository by adding a signature action using the provided key and certificates. The package modified will retain the original timestamp.

OPTIONS

The following options are supported:

With -a, use the signature algorithm provided instead of the default, which is rsa-sha256. The following are the currently supported signature algorithms: rsa-sha256, rsa-sha384, rsa-sha512, sha256, sha384, sha512. A signature algorithm which only specifies a hash algorithm will cause the signature value to be the hash of the manifest of the package. A signature algorithm which specifies rsa and a hash algorithm will cause 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 provided as the certificate to use when verifying the value of the signature in the action. It can only be used if -k is also used.

With -i, add the certificate provided as a certificate to use when validating the certificate given as an argument to -c. Multiple certificates may be provided by using -i multiple times.

With -k, use the private key stored in the given path to sign the manifest. It can only be used if -c is also used. If -k is not set, then the signature value will be the hash of the manifest.

With -s, sign packages in the repository at the given URI.

With --help, show the usage information for the command.

With --no-index, tell the repository not to update the search indices after the signed manifest has been republished.

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

With --sign-all, sign all the packages in the repository.

EXAMPLES

Example 1: 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 a package published into the file repository in /foo/bar using rsa-sha384 to hash and sign the manifest, the key in /key/usr2.key, its associated certificate in /key/usr2.cert, and a certificate needed to validate the certificate 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.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	pkg:/package/pkg
Interface Stability	None / Under Development

SEE ALSO

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

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject to change. Development is hosted in the OpenSolaris community at:

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

pkg.depotd(1M)

NAME

pkg.depotd - image packaging system depot server

SYNOPSIS

```
/usr/lib/pkg.depotd [-d inst_root] [-p port] [-s threads]
    [-t socket_timeout] [--add-content] [--cfg] [--content-root]
    [--debug] [--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 filesystem, or are in situations where networkaccess may be 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. See the man pages of each program for more information about supported operations.

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

The depot currently 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 (with the exception that when running under 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.

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

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>] (e.g. catalog, search_1).

pkg/file_root	(astring) The root of the file content for this repository. If undefined, this defaults to <inst_root>/file.
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/pkg/repo.
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 pathname. The default value is stdout if stdout is a tty; if it 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 pathname. 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 may not 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. The default value is 80 if ssl certificate and key information has not been provided; 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 webserver in a reverse proxy configuration.
pkg/readonly	(boolean) Sets whether modifying operations, such as those initiated by pkgsend(1M) are disabled. Retrieval operations are still available. This property may not 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 pathname to a PEM-encoded Certificate file. The default value is none. This property must be used with ssl_key_file. The depot will only respond to SSL requests if 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: exec:/path/to/program, builtin, or smf:fmri. builtin will prompt for the passphrase; exec will execute the specified external program to obtain the passphrase, which should be printed to stdout. The first argument to the program will be '', and is reserved. The second argument to the program will be the port number of the server. smf:fmri will attempt to retrieve the value of the property pkg_secure/ssl_key_passphrase from the service instance related to the fmri. The default value is builtin.

pkg/ssl_key_file (astring) The absolute pathname to a PEM-encoded Private Key file. This property must be used with the property ssl_cert_file. The depot will only respond to SSL requests if provided.

pkg/threads (count) The number of threads that will be started to serve requests. The default value is 10.

pkg/writable_root (astring) The file system path to a directory to which the program has write access. This is used with --readonly to allow the depot server to create files, such as search indices, 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 pathname of a small image that will be used to visually represent the RSS/Atom feed. The pathname should be relative to the content_root. The default value is "web/_themes/pkg-block-icon.png"

pkg_bui/feed_logo (astring) The pathname 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) A numeric value representing the number of hours, before the feed for the repository was last generated, to include when generating the feed.

The pkg depot is also able to act as a mirror server for local client images from pkg(5). This allows clients that share a subnet on a LAN to mirror their file caches. Clients may 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/file_root — (astring) The file system path at which the instance should find its file content. The default value is /var/pkg/download.

pkg/port — (count) The port number on which the instance should listen for incoming package requests. The default value is 80.

OPTIONS

pkg.depotd(1m) can read its base configuration information from a file or from the property data of an existing smf(5) service instance:

--cfg source — The pathname 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 pre-existing configuration source is available, or to override values read from a provided configuration file using '--cfg', the following options can be used to alter the default behavior of the depot server:

-d inst_root — See pkg/inst_root above.

-p port — See pkg/port above.

-s threads — See pkg/threads above.

-t socket_timeout — See pkg/socket_timeout above.

--content-root root_dir — See pkg/content_root above.

--disable-ops op_list — See pkg/disable_ops above.

--debug features — See pkg/debug above.

```
     --file-root path           See pkg/file_root above.

     --log-access dest          See pkg/log_access above.

     --log-errors dest          See pkg/log_errors above.

     --mirror                   See pkg/mirror above.

     --proxy-base url           See pkg/proxy_base above; ignored if
                                empty value is provided.

     --readonly                 See pkg/readonly above.

     --sort-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.

     --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)
     using the --cfg option, the depot server will read and write all
     configuration data (listed in DESRIPTION above) in a simple text
     format that consists of sections, lead by a "[section]" header,
     and followed by "name = value" entries, with continuations, etc.
     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 = /var/pkg/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

     Example 4:  Changing the listening port of the server.

     # svccfg -s application/pkg/server setprop pkg/port = 20000
```

```
     # svcadm refresh application/pkg/server
     # svcadm restart application/pkg/server

ENVIRONMENT VARIABLES
     PKG_REPO                  Specifies the directory containing the
                               repository to server.  This will be
                               ignored if -d was used.

     PKG_DEPOT_CONTENT         Specifies the directory containing
                               static content served by the depot.
                               The files listed below under FILES
                               should be present in this directory,
                               although their content may differ
                               from the supplied default content.

EXIT STATUS
     The following exit values are returned:

     0     Successful operation.

     1     Error encountered.

     2     Invalid command line options were specified.

FILES
     /var/pkg/repo             Default repository location; modify
                               pkg/inst_root to select an
                               alternate location.

     /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	SUNWipkg pkg:/package/pkg
Interface Stability	None / Under Development

```
SEE ALSO
     dns-sd(1), mdnsd(1), pkg(1), pkgrepo(1), pkgsend(1), syslogd(1M),
     attributes(5), smf(5)

NOTES
     The image packaging system is an under-development feature.
     Command names, invocation, formats, and operations are all
     subject to change.  Development is hosted in the OpenSolaris
     community at:

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

     The pkd.depotd service is managed by the service management
     facility, smf(5), under the service identifier:

          svc:/application/pkg/server

     The mDNS mirror service is managed by the service management
     facility, smf(5), under the service identifier:

          svc:/application/pkg/dynamic-mirror

     Because the depot server and mirror expect to be run by an smf(5)
```

restarter, they do not daemonize. Error messages are generally sent to standard output, or to the syslogd(1M) system.

When publishing individual package files that are greater than 128MB in size, filesystem-based publication must be used due to publication protocol limitations. See pkgsend(1) for more information.

For controlling read access to the depot, it is suggested that administrators use an HTTP reverse proxy in combination with authentication methods such as client-based SSL certificate access which pkg(1) natively supports.

If new package versions must be added to a public depot, it is suggested that filesystem-based publication be performed instead, and that the depot server process be restarted afterwards. For more information about filesystem-based publication, see pkgsend(1).

Changes to configuration, or changes to package data using file-system based operations require a restart of the depot server process so that they can be reflected in operations and output. This can be done by using one of the following methods:

* using svcadm(1M) to restart the application/pkg/server

  instance

* sending a SIGUSR1 signal to the depot server process using kill(1), which will execute a 'graceful restart' that leaves the process intact, but reloads all configuration, package, and search data:

  ```
  kill -USR1 <pid>
  ```

pkg(5)

NAME

pkg - image packaging system

DESCRIPTION

The image packaging system, pkg(5), is a framework which 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 filesystem, 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://opensolaris.org/library/libc@5.11,5.11-0.75:20071001T163427Z
```

'opensolaris.org' is the publisher. 'library/libc' is the package name. Although the namespace is hierarchical and arbitrarily deep, there is no enforced containment -- the name is essentially arbitrary.

A publisher's identifying name is a forward domain name that can be used to identify a person, group of persons, or an organization as the source of one or more packages. A publisher name should be constructed from an Internet domain name that is owned, managed, or represented by or on behalf of a publisher. Other constructions may lead to collisions and be unsafe for general use.

The version follows the package name, separated by an '@'. It 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.

The first part is the component version. For components tightly bound to OpenSolaris, this will usually be the value of 'uname -r' for that version of OpenSolaris. For a component with its own development lifecycle, this sequence will be the dotted release number, such as '2.4.10'.

The second part, following the comma, is the build version, specifying what version of OpenSolaris the contents of the package were built on, providing a minimum bound on which OpenSolaris version the contents can be expected to run successfully.

The third part, following the dash, is the branch version, a versioning component, providing vendor-specific information. This may be incremented when the packaging metadata is changed, independently of the component, may contain a build number, or some other information.

The fourth part, following the colon, is a timestamp. It represents when the package was published.

Many parts of the system, when appropriate, will contract FMRIs when displaying them to reduce the volume of information provided. Typically, the scheme, publisher, build version, and timestamp will be elided, and sometimes the versioning information altogether.

Actions

Actions represent the installable objects on a system. They are

described in a package's manifest. 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 may have other attributes. Some of these will be interpreted directly by the packaging system, others may be useful only to the system administrator or the end-user.

The attributes listed below are not an exhaustive set; indeed, the attributes which may be attached to an action are arbitrary, and indeed, the standard sets are easily augmented to incorporate future developments.

Certain action attributes cause additional operations to be executed outside of the packaging context. These are documented below, under the topic "Actuators".

File Actions

The 'file' action is perhaps the most common action, and represents an 'ordinary file'. It references a payload, and has four standard attributes:

path The filesystem 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 which owns the file.

group The name of the group which owns the file.

Other attributes include:

preserve This specifies that the file's contents should not be overwritten on upgrade if they are determined to have changed since it was installed or last upgraded.

If the value of this attribute is 'renameold', then the existing file will be renamed, and the new file will be put in its place.

If the value of this attribute is 'renamenew', then the existing file will be left alone, and the new file will be installed with a new name.

If the value of this attribute is 'strawberry', then the existing file will be left alone, and the new file will not be installed.

Files may also be 'tasted', and depending on the flavor, may have additional interesting attributes. For ELF files, the following attributes are recognized:

elfarch The architecture of the ELF file. This will be the output of 'uname -p' on the architecture for which the file is built.

elfbits This will be '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, and are the only ones necessary to consider when determining whether two binaries' executable behavior 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 will use the file of that name if it is deleted as part of the same packaging operation.

Directory Actions

The 'dir' action is like the file action in that it represents a filesystem object, but a directory instead of an ordinary file. It has the same four standard attributes as the file action, and 'path' is the key attribute.

If a package operation triggers a directory removal, the client can take two possible actions. It will either preserve the directory, or will move its contents into $IMAGE_META/lost+found. See FILES for more information about $IMAGE_META. Directory removal typically occurs during an uninstall operation, but it may also occur during an update from one version to another.

Link Actions

The 'link' action represents a symbolic link. It has two standard attributes:

path The filesystem path where the symlink is installed. This is a link action's key attribute.

target The target of the symlink; the filesystem object to which the link resolves.

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. It 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 filename of the driver binary. This is the driver action's key attribute.

alias This represents an alias for the driver. There may be more than one alias attribute for any given driver. There are no special quoting rules necessary.

class This represents a driver class. There may be more than one class attribute for any given driver.

perms This represents the filesystem permissions for the driver's device nodes.

clone_perms This represents the filesystem permissions for the "clone" driver's minor nodes for this driver.

policy This specifies additional security policy for the device. There may be more than one policy attribute for any given driver, but no minor device specification may be present in more than one attribute.

privs This specifies privileges used by the driver. There may be more than one privs attribute for any given driver.

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. There may be more than one devlink attribute for any given driver.

Depend actions

The 'depend' action represents an inter-package dependency. A package may 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 may be optional.

The following attributes are recognized:

type The type of the dependency. If the value is 'require', then the dependency is required. A package cannot be installed if any of its required dependencies cannot be satisfied.

If the value is 'optional', then the dependency is optional. Optional dependencies are followed or not depending on image policy (see 'Policy' below).

If the value is 'exclude', then the package is non-functional if the dependent package is present on the system.

If the value is 'incorporate', then the dependency is optional, but the version of the dependent package will become 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 will satisfy the dependency

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.

fmri The FMRI representing the depended-upon package. In the case of require-any dependencies, there may be multiple values. This is the dependency action's key attribute.

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 may deliver licenses, disclaimers, or other guidance to the package installer through the use of the license action.

The payload of the license action will be delivered into the image metadata directory related to the package, and should only contain human-readable textual data. It should not contain HTML or any other form of markup. License actions, through attributes, may indicate to clients that the related payload must be displayed and/or require "acceptance" of it. Please note that the exact method of display and/or acceptance is at the discretion of clients.

The following attributes are recognized:

license
: 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 might 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"

  Wherever possible, including the version of the license in the description is recommended as shown above. This value must be unique within a package.

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 will be considered equivalent to "false". The method of acceptance (interactive, configuration-based, etc.) 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 considered 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.

The 'license' attribute is the key attribute for the license action.

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 in order that the tools querying those databases might 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 it may 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 package's FMRI.

vendor
: The value for the VENDOR parameter.

version
: The value for the VERSION parameter. The default value is

the version from the package's FMRI.

The 'pkg' attribute is the key attribute for the legacy action.

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 which have specific meaning to the packaging system.

info.classification	One or more tokens which a pkg(5) client may 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 may 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 which point to the package versions to which this package has been renamed. A package may not be marked both renamed and obsolete, but otherwise may 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 may be added with the 'user' action. The following attributes are recognized:

groupname The value for the name of the group.

gid The groups 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 users primary group. Must be found in /etc/group

gcos-field The value of the gecos 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).

lastchng 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 may 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 may 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 may 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 take the value of an FMRI of a service instance to operate upon during the package installation or removal. disable_fmri causes the mentioned 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 or update, 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 may contain a pattern matching 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
TBD

Client-Server Operation
TBD

Publishers and Mirroring
TBD

Images and Substrates
TBD

Properties
Images can have one or more properties associated with them. These properties can be used to store information about the purpose, content and behavior of the image.

Image Policies
Policies are defined by image properties with boolean values. The supported policies include:

flush-content-cache-on-success
When true, the cache of downloaded files is erased after a successful install of a package. Default value: False.

send-uuid
When true, a unique identifier (UUID) that identifies the image to the publisher is sent on all requests. Default value: True.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWipkg pkg:/package/pkg

Interface Stability	None / Under Development

FILES

Since pkg(5) images can be located arbitrarily within a larger file system, we use the token $IMAGE_ROOT 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. We use the token $IMAGE_META to refer 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.

$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.

SEE ALSO

pkg(1), pkgsend(1), pkg.depotd(1M), svcadm(1M), pkginfo(4), attributes(5)

NOTES

The image packaging system is an under-development feature. Command names, invocation, formats, and operations are all subject to change. Development is hosted in the OpenSolaris community at

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