```
packagemaker(1)           BSD General Commands Manual          packagemaker(1)
```

NAME

packagemaker, PackageMaker -- Installation-package creation tool

SYNOPSIS

packagemaker --root *root-path* [options]

packagemaker --doc *pmdoc-path* [options]

packagemaker --watch [options]

packagemaker -build --f *root-path* [old-pkg-options]

packagemaker -build -mi | **-mc** | **-ms** *packages-path* [old-mpkg-options]

packagemaker --sign *flat-pkg-or-distribution-path* **--certificate** *certificate-name* [--out *destination-path*]

packagemaker -help

DESCRIPTION

packagemaker allows you to package files in a way that makes it easy for end users to install them on their computers.

The **packagemaker** tool provides five pieces of functionality:

1. Build a PackageMaker document (.pmdoc) using the --doc flag

2. Build a package or flat package from a root using the --root flag

3. Build a snapshot package based on filesystem changes using the --watch flag

4. Sign a flat package, flat metapackage or distribution file using the --sign flag

5. A backwards-compatibility mode for building packages and metapackages using the old -build flag

The sections below describe each of these types of functionality.

BUILDING USING --root OR --doc

If a PackageMaker document (.pmdoc) file is provided via the --doc flag, **packagemaker** will build whatever is specified in that document. If the document specifies a package, you can also use any of the options described below to override certain settings in the document.

Alternatively, you can provide a directory to **packagemaker** using the --root flag and **packagemaker** will construct a package from that root. In this case, an identifier must be provided with the --id flag or specified in the info file provided by the --info flag. You may use the other options described below to customize the package.

Options:

--root, **-r** *root-path*
A path to a directory to package. Either this or the --doc flag must be specified.

--doc, **-d** *pmdoc-path*
A path to a .pmdoc file built using the PackageMaker GUI. Either this or the --root flag must be specified.

--id, **-i** *package-identifier*
A package identifier, which should be unique for this package. For example, com.apple.packagemaker. If --root is specified, either this or the --info flag must be specified.

--info, **-f** *info-path*
A path to an Info.plist file for a bundle package or a Package-Info file for a flat package. If --root is specified, either this or the --id flag must be specified. If specified along with --doc, the provided info file will be merged with the file generated by **packagemaker**

[--out, -o *destination-path*]
If specified, the build result will be output to destination-path. If not specified, the build result will be output into the present working directory with a name derived from the input file.

[--version, -n *version*]
The version number that will be given to your package. If not specified, will default to "1". Will override the version specified in a pmdoc for a package. However, if the pmdoc specifies a metapackage or distribution, this flag will have no effect.

[--title, -t *title*]
The title that will be given to your package. If not specified, the title will be derived from the root path (for 10.3 or 10.4 target) or from the package file name (for 10.5 target). If specified, a distribution will be created. Will override the title specified in a pmdoc.

[--resources, -e *resources-path*]
A path to a directory of resources to be copied into the package. The directory should be structured as you want it to be in the package. For example, localized resources should be in the appropriate lproj directory and all resources should have the appropriate names. See the Installer documentation for more information. If specified along with --doc, the resources will be merged with any specified in the document.

[--scripts, -s *scripts-path*]
A path to a directory of scripts to be copied into the package. The directory should be structured as you want it to be in the package. For example, scripts should have appropriate names. See the Installer documentation for more information. If specified along with --doc, the scripts will be merged with any specified in the document.

[--certificate, -c *certificate-name*]
The name of a certificate with which to sign the flat package or flat metapackage. The name should match that of a certificate in your keychain that is valid for signing. Please note that if **packagemaker** requires permission to use the certificate, using this option will cause the standard GUI permission dialog to appear.

[--filter, -x *regular-expression*]
Adds a file filter. Any files in the root matching the provided regular expression will be not be included in the package. This flag can be specified multiple times. If specified along with --doc, the filters will be appended to any specified in the document.

[--target, -g *10.5* | *10.4* | *10.3*]
Specifies the minimum target operating system version. Defaults to 10.3. For 10.5, flat packages and metapackages will be built; for 10.4, bundle packages and distributions will be built; and for 10.3, bundle packages and metapackages will be built.

[--domain, -h *system* | *home* | *anywhere*]
Adds an install domain. This flag can be specified multiple times. Prior to Mac OS X v10.5, the Installer will default to the 'anywhere' domain. You can use the --root-volume-only flag to achieve the same effect as the system domain. If specified

along with --doc, will override to domain settings of the document.

[--no-recommend, -m]
If specified, **packagemaker** will not apply recommended permissions to package contents.

[--discard-forks, -k]
If specified, **packagemaker** will not preserve resource forks when building packages.

[--root-volume-only, -b]
If specified, install will only be allowed on the root volume.

[--verbose, -v]
Provide detailed status information during construction.

SIGNING PACKAGES USING --sign

Existing flat packages, flat metapackages or distributions (.dist) can be signed using the --sign flag. Please note that if **packagemaker** requires permission to use the certificate, the standard system permission GUI dialog will appear.

Options:

--sign *flat-pkg-or-distribution-path*
The path to a flat package, flat metapackage or distribution file (.dist).

--certificate, -c *certificate-name*
The name of a certificate with which to sign the flat package or flat metapackage. The name should match that of a certificate in your keychain that is valid for signing.

[--out, -o destination-path]
If specified, the signed version of the input will be output to destination-path. If not specified, the signed version will replace the input.

SNAPSHOT PACKAGES USING THE --watch FLAG

If the --watch flag is specified, **packagemaker** will monitor filesystem changes until it receives the SIGUSR1 signal. It will then construct a package of all files that were created/modified while it was watching. All of the flags described above for --root and --doc are applied to the package, with the exception of --root and --doc themselves.

BUILDING IN BACKWARDS-COMPATIBILITY MODE

In addition to the interface described above, **packagemaker** supports the old CLI for backwards compatibility. It can be used to build packages or metapackages using the following options:

-build Create an installation package or metapackage. Must be specified to trigger backwards-compatibility mode.

-proj A path to a pmproj document. **packagemaker** will import and build the document. This will cause the -f, -i, -r, and -d flags to be ignored.

-p The path, including the package name and extension (.pkg or .mpkg) where the package is created.

-f Directory containing the contents of the package. Not applicable when building a metapackage.

-b Directory used to temporarily copy and modify the root if splitting resource forks (suggestion: /tmp). Not applicable when building a metapackage.

-s Split files with resource forks (Installer will reassemble them). Overrides default behavior that discards resource forks. Not applicable when building a metapackage.

-ds Filter .DS_Store files out of the creation process. Not applicable when building a metapackage.

-v Verbose output during archiving.

-u Create uncompressed archive. Not applicable when building a metapackage.

-r Directory containing installation resources, such as scripts and Read Me files.

-i Path to property list file (.plist) that is copied to the package's Contents directory as Info.plist. It will be modified to contain the package's installed size (IFPkgFlagInstalledSize) and other package flags as necessary. This option must be specified and the Info.plist must contain a CFBundleIdentifier key. This CFBundleIdentifier should uniquely identify your package.

-d Path to property list file (.plist) that is copied to the package's Resources directory as Description.plist. If this option is unspecified, a skeletal Description.plist is generated for the package. You should add the title and description entries to the Description.plist file after creating the package.

-mi Path to directory of packages/metapackages to be included in the metapackage. The packages will be stored within the created metapackage.

-ms Path to directory of packages/metapackages to be included in the metapackage. The packages will not be stored within the created metapackage, rather, they should be on the same level as the .mpkg file.

-mc Path to directory of packages/metapackages to be included in the metapackage. The packages will not be stored within the created metapackage, rather, they should remain at the location specified.

THE PACKAGE FORMAT

An installation package contains everything the Installer application needs to install a group of files (the package's payload), which can include application bundles, documentation files, scripts, and so on. In general, a package contains the following:

A bill of materials file:
A binary file that describes the contents of the package.

An information property list:
An XML file that contains the information about the package, such as default location and version.

An archive file:
The set of files to be installed, also known as the payload. With **packagemaker**, this archive is always compressed.

Resources directory:
This directory contains files Installer uses during an installation but doesn't install on the target computer. They include Read Me files, license-agreement files, and scripts.

A metapackage is a file that includes a list of packages (and possibly other metapackages) and any additional information needed to install them. The actual packages can be stored in the metapackage, on the same level as the metapackage, or at a custom location. In general, a metapackage contains the following:

An information property list:
An XML file that contains the information about the metapackage, such as version and package list.

Packages directory:
Contains any packages stored within the metapackage.

Resources directory:
This directory contains files Installer uses during an installation but doesn't install on the target computer. They include Read Me files, license-agreement files, and scripts.

A distribution is similar to a metapackage, except that it contains a distribution.dist file, which contains XML and JavaScript which specify the UI for the Install. In general, a distribution contains the following:

An distribution script:
An XML file specifying the UI for the install. May also contain JavaScript.

Packages directory:
Contains any packages stored within the distribution.

Resources directory:

This directory contains files Installer uses during an installation but doesn't install on the target computer. They include Read Me files, license-agreement files, and scripts.

See <**http://developer.apple.com/documentation/DeveloperTools/Conceptual/SoftwareDistribution/**> as well as the help integrated into PackageMaker for information on the keys of the property-list files as well as a detailed explanation of package creation, format, and use.

Scripts can be included in your packages/metapackages to test certain conditions before installation or when you need to perform special tasks as the installation takes place.

Scripts can be run before and after the package's payload is installed. There are two types of scripts: environment-test scripts and installation scripts.

These are the environment-test scripts you can define for an installation:

InstallationCheck
Installer runs this script to determine whether the installation should proceed.

VolumeCheck
Installer runs this script to determine whether a particular volume can receive the package's payload.

If the environment-test scripts allow the installation to proceed, Installer performs the installation scripts and the installs the payload in the following order:

preflight
preinstall   or *preupgrade*
Payload installation
postinstall   or *postupgrade*
postflight

EXAMPLES

Building a root with an identifier:
```
      packagemaker --root /tmp/MyGreatApp.dst --id com.example.MyGreatApp
      --out /tmp/MyGreatApp.pkg
```
Building a flat package with an existing PackageInfo file:
```
      packagemaker --root /tmp/MyGreatApp.dst --info /tmp/MyPackageInfo
      --target 10.5 --out /tmp/MyGreatApp.pkg
```
Building a package from a pmdoc, overriding the version and title:
```
      packagemaker --doc /tmp/MyGreatDoc.pmdoc --version 2.0 --title 'My
      Great App v2'
```
Filtering files named "foo" from your package:
```
      packagemaker --root /tmp/MyGreatApp.dst --id com.example.MyGreatApp
      --filter '/foo$'
```
Creating Cool_App.pkg with existing Info.plist and Description.plist files:
```
      packagemaker -build -p /Volumes/Packages/Cool_App/Cool_App.pkg -f
      /Volumes/Packages/Cool_App/Package_contents -i
      /Volumes/Packages/Cool_App/Info.plist -d
      /Volumes/Packages/Cool_App/Description.plist
```

SEE ALSO

installer(8),
http://developer.apple.com/documentation/DeveloperTools/Conceptual/SoftwareDistribution/