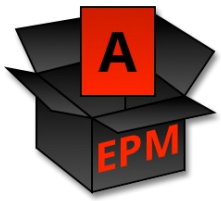


Software Distribution with ESP Package Manager

MICHAEL R. SWEET

JIM JAGIELSKI

Software Distribution with ESP Package Manager

Table of Contents

Preface

This book provides a tutorial and reference for the ESP Package Manager ("EPM") software, version 5.0, and is organized into the following chapters and appendices:

- 1 - Introduction to EPM
- 2 - Building EPM
- 3 - Packaging Your Software with EPM
- 4 - Advanced Packaging with EPM
- 5 - EPM Packaging Examples
- A - Software License Agreement
- B - Command Reference
- C - List File Reference

Notation Conventions

The names of commands; the first mention of a command or function in a chapter is followed by a manual page section number:

```
epm
epm(1)
```

File and directory names:

/var
/usr/bin/epm

Screen output:

```
Request ID is Printer-123
```

Literal user input; special keys like **ENTER** are in ALL CAPS:

lp -d printer filename ENTER

Long commands are broken up on multiple lines using the backslash (\) character; enter the commands without the backslash:

```
foo start of long command \
    end of long command ENTER
```

Numbers in the text are written using the period (.) to indicate the decimal point:

12.3

Abbreviations

The following abbreviations are used throughout this book:

kb
: Kilobytes, or 1024 bytes

Mb
: Megabytes, or 1048576 bytes

Gb
: Gigabytes, or 1073741824 bytes

Other References

https://jimjag.github.io/epm/
: The official home page of the ESP Package Manager software.

http://www.debian.org/devel/
: Debian Developers' Corner

http://techpubs.sgi.com/
: IRIX Documentation On-Line

http://www.rpm.org/
: The Red Hat Package Manager home page.

http://docs.sun.com/
: Solaris Documentation On-Line

Help Me Improve This Book!

We've done my best to ensure that this book is both accurate and clear. If you find errors or have a suggestion for improving the book, please file a bug at:

```
https://github.com/jimjag/epm/issues
```

Acknowledgments

We'd like to thank the following people for their contributions to EPM:

- Gareth Armstrong: HP-UX and %release enhancements
- Nicolas Bazin: Openserver and Unixware support
- Richard Begg: HP-UX fixes
- Dirk Datzert: Bug fixes
- Alan Eldridge: Makefile and RPM fixes
- Vicentini Emanuele: IRIX enhancements
- Jeff Harrell: IRIX enhancements
- Lars Kellogg-Stedman: Debian fixes
- Jochen Kmietsch: mkepmlist fixes

- Aneesh Kumar K.V.: Tru64 setld package support
- David Lee: Build system improvements
- Scott Leerssen: mkepmlist fixes, BSD package support
- Jeff Licquia: Debian support/enhancements
- David Maltz: AIX fixes
- Joel Nordell: SCO fixes
- Rok Papez: Bug fixes and absolute output directory support
- Holger Paschke: Documentation fixes
- Phil Reynolds: OpenBSD fixes
- Ganesan Rajagopal: Solaris fixes
- Uwe Räsche: AIX support
- Ralf Rohm: Solaris fixes
- Jochen Schaeuble: epminstall fixes
- Jason Shiffer: HP-UX fixes
- Andrea Suatoni: IRIX fixes
- Andy Walter: QNX support
- Geoffrey Wossum: --output-directory option
- Jean Yves: BSD package and mkepmlist fixes

1 - Introduction to EPM

This chapter provides an introduction to the ESP Package Manager ("EPM").

What is EPM?

Software distribution under UNIX/Linux can be a challenge, especially if you ship software for more than one operating system. Every operating system provides its own software packaging tools and each has unique requirements or implications for the software development environment.

The ESP Package Manager ("EPM") is one solution to this problem. Besides its own "portable" distribution format, EPM also supports the generation of several vendor-specific formats. This allows you to build software distribution files for almost any operating system from the same sources.

History and Evolution

When Easy Software Products was founded by me in 1993, the company originally shipped software only for the SGI IRIX operating system. Support was added for Solaris in 1997, followed quickly by HP-UX in 1998.

Each new operating system and supported processor required a new set of packaging files. While this worked, it also meant keeping all of the packaging files synchronized manually. Needless to say, this process was far from perfect and there was more than one distribution that was not identical on all operating systems.

As I began developing CUPS (https://www.cups.org/) in 1997, the initial goal was to add support for two additional operating systems: Linux and Compaq Tru64 UNIX. If I was to avoid the mistakes of the past, I clearly had to change how software distributions were produced.

The first version of EPM was released in 1999 and supported so-called "portable" software distributions that were not tied to any particular operating system or packaging software. Due to popular demand, we added support for vendor-specific packaging formats in the second major release of EPM, allowing the generation of portable or "native" distributions from one program and one set of software distribution files.

Existing Software Packaging Systems

As I looked for a solution to our problem, we naturally investigated the existing open-source packaging systems. Under Linux, I looked at the Red Hat Package Manager ("RPM") and Debian packaging software ("dpkg" and "dselect"). For the commercial UNIX's I looked at the vendor-supplied packaging systems. Table 1.1 shows the results of my investigation.

Table 1.1: Software Packaging Formats

Format	Operating Systems[1]	Binaries	Cross-Platform	Patches	Up-grades	Con-flicts	Re-quires	Re-places	Config Files	Map Files	Un-install
installp	AIX	Yes	No	No	No	Yes	Yes	No	No	No	Yes
pkg_add	FreeBSD	Yes	Yes[2]	No	No	No	No	No	No	No	Yes
pkg_add	NetBSD OpenBSD	Yes	Yes[2]	No	No	Yes	Yes	No	No	No	Yes
dpkg	Ubuntu Linux Debian GNU/Linux	Yes	Yes[2]	No	Yes	Yes	Yes	Yes	Yes	No	Yes
depot	HP-UX	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
inst	IRIX	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Install.app	macOS	Yes	No	No	Yes	No	No	No	No	No	No
pkgadd	Solaris	Yes	No	Yes	No	Yes	Yes	No	Yes	Yes	Yes
rpm	CentOS Red Hat SuSE	Yes	Yes[2]	No	Yes	Yes	Yes	No	Yes	No	Yes
setld	Tru64 UNIX	Yes	No	No	No	Yes	Yes	No	No	No	Yes
slackware	Slackware Linux	Yes	No	No	No	Yes	Yes	No	No	No	Yes

1. Standard packaging system for named operating systems.
2. These packaging systems are cross-platform but require the package management utilities to be installed on the platform before installing the package.

As you can see, none of the formats supported every feature we were looking for. One common fault of all these formats is that they do not support a common software specification file format. That is, making a Debian software distribution requires significantly different support files than required for a Solaris pkg distribution. This makes it extremely difficult to manage distributions for multiple operating systems.

All of the package formats support binary distributions. The RPM and Debian formats also support source distributions that specifically allow for recompilation and installation. Only the commercial UNIX formats support patch distributions - you have to completely upgrade a software package with RPM and Debian. All but the Solaris pkg format allow you to upgrade a package without removing the old version first.

When building the software packages, RPM and Debian force you to create the actual directories, copy the files to those directories, and set the ownerships and permissions. You essentially are creating a directory for your software that can be archived in the corresponding package format. To ensure that all file permissions and ownerships are correct, you must build the distribution as the root user or use the `fakeroot` software, introducing potential security risks and violating many corporate security policies. It can also make building distributions difficult when dynamic data such as changing data files or databases is involved.

The commercial UNIX formats use software list files that map source files to the correct directories and permissions. This allows for easier delivery of dynamic data, configuration management of what each distribution actually contains, and eliminates security issues with special permissions and building distributions as the root user. Using the proprietary format also has the added benefit of allowing for software patches and using the familiar software installation tools for that operating system. The primary disadvantage is that the same distributions and packaging software cannot be used on other operating systems.

Design Goals of EPM

EPM was designed from the beginning to build binary software distributions using a common software specification format. The same distribution files work for all operating systems and all distribution formats. Supporting source code distributions was not a goal since most RPM and Debian source distributions are little more than wrapping around a compressed tar file containing the source files and a configure script.

Over the years, additional features have made their way into EPM to support more advanced software packages. Whenever possible, EPM emulates a feature if the vendor package format does not support it natively.

Resources

The EPM project page provides access to the current software, documentation, and issue tracker for EPM:

```
https://jimjag.github.io/epm/
```

The EPM source code can be downloaded in compressed tar files or via the popular Subversion software. Please see the EPM project page for complete instructions.

2 - Building EPM

This chapter shows how to configure, build, and install the ESP Package Manager.

Requirements

EPM requires very little pre-installed software to work. Most items will likely be provided as part of your OS. Your development system will need a C compiler, the `make(1)` program (GNU, BSD, and most vendor `make` programs should work), a POSIX shell (Bourne, Korn, Bash, etc.), and `gzip(1)`.

The optional graphical setup program requires a C++ compiler, the FLTK library, version 1.1.x or 1.3.x, and (for UNIX/Linux) the X11 libraries. FLTK is available at the following URL:

```
http://www.fltk.org/
```

Your end-user systems will require a POSIX shell, the `df(1)` program, the `tar(1)` program, and the `gzip(1)` program to install portable distributions. All but the last are standard items, and most vendors include `gzip` as well.

EPM can also generate vendor-specific distributions. These require the particular vendor tool, such as `rpm(8)` and `dpkg(8)`, to generate the software distribution on the development system and load the software distribution on the end-user system.

Configuring the Software

EPM uses GNU `autoconf(1)` to configure itself for your system. The `configure` script is used to configure the EPM software, as follows:

```
./configure ENTER
```

Choosing Compilers

If the `configure` script is unable to determine the name of your C or C++ compiler, set the `CC` and `CXX` environment variables to point to the C and C++ compiler programs, respectively. You can set these variables using the following commands in the Bourne, Korn, or Bash shells:

```
export CC=/foo/bar/gcc ENTER
export CXX=/foo/bar/gcc ENTER
```

If you are using C shell or tcsh, use the following commands instead:

```
setenv CC /foo/bar/gcc ENTER
setenv CXX /foo/bar/gcc ENTER
```

Run the `configure` script again to use the new commands.

Choosing Installation Directories

The default installation prefix is */usr/local*, which will place the EPM programs in */usr/local/bin*, the setup GUI in */usr/local/lib/epm*, and the man pages in */usr/local/share/man*.

Use the `--prefix` option to relocate these files to another directory:

```
./configure --prefix=/example/path ENTER
```

The `configure` script also accepts the `--bindir`, `--libdir`, and `--mandir` options to relocate each directory separately, as follows:

```
./configure --bindir=/example/path/bin --libdir=/example/path/lib \
    --mandir=/example/path/share/man ENTER
```

Options for the Setup GUI

The setup GUI requires the FLTK library. The configure script will look for the `fltk-config` utility that comes with FLTK. Set the FLTKCONFIG environment variable to the full path of this utility if it cannot be found in the current path:

```
setenv FLTKCONFIG /foo/bar/bin/fltk-config ENTER
```

or:

```
FLTKCONFIG=/foo/bar/bin/fltk-config ENTER
export FLTKCONFIG
```

Building the Software

Once you have configured the software, type the following command to compile it:

```
make ENTER
```

Compilation should take a few minutes at most. Then type the following command to determine if the software compiled successfully:

```
make test ENTER
Portable distribution build test PASSED.
Native distribution build test PASSED.
```

The `test` target builds a portable and native distribution of EPM and reports if the two distributions were generated successfully.

Installing the Software

Now that you have compiled and tested the software, you can install it using the `make` command or one of the distributions that was created. You should be logged in as the super-user unless you specified installation directories for which you have write permission. The `su(8)` command is usually sufficient to install software:

```
su ENTER
```

Operating systems such as macOS do not enable the root account by default. The `sudo(8)` command is used instead:

```
sudo installation command ENTER
```

Installing Using the `make` Command

Type the following command to install the EPM software using the `make` command:

```
make install ENTER
Installing EPM setup in /usr/local/lib/epm
Installing EPM programs in /usr/local/bin
Installing EPM manpages in /usr/local/share/man/man1
Installing EPM documentation in /usr/local/share/doc/epm
```

Use the `sudo` command to install on macOS:

```
sudo make install ENTER
Installing EPM setup in /usr/local/lib/epm
Installing EPM programs in /usr/local/bin
Installing EPM manpages in /usr/local/share/man/man1
Installing EPM documentation in /usr/local/share/doc/epm
```

Installing Using the Portable Distribution

The portable distribution can be found in a subdirectory named using the operating system, version, and architecture. For example, the subdirectory for a Linux 2.4.x system on an Intel-based system would be *linux-2.4-intel*. The subdirectory name is built from the following template:

```
os-major.minor-architecture
```

The `os` name is the common name for the operating system. Table 2.1 lists the abbreviations for most operating systems.

The `major.minor` string is the operating system version number. Any patch revision information is stripped from the version number, as are leading characters before the major version number. For example, HP-UX version B.11.11 will result in a version number string of `11.11`.

Table 2.1: Operating System Name Abbreviations

Operating System	Name
AIX	`aix`
Compaq Tru64 UNIX Digital UNIX OSF/1	`tru64`
FreeBSD	`freebsd`
HP-UX	`hpux`
IRIX	`irix`
Linux	`linux`
macOS	`osx`
NetBSD	`netbsd`
OpenBSD	`openbsd`
Solaris	`solaris`

Table 2.2: Processor Architecture Abbreviations

Processor(s)	Abbreviation
Compaq Alpha	`alpha`
HP Precision Architecture	`hppa`
INTEL 80x86	`intel`
INTEL 80x86 w/64bit Extensions	`x86_64`
MIPS RISC	`mips`
IBM Power PC	`powerpc`
SPARC MicroSPARC UltraSPARC	`sparc`

The `architecture` string identifies the target processor. Table 2.2 lists the supported processors.

Once you have determined the subdirectory containing the distribution, type the following commands to install EPM from the portable distribution:

```
cd os-major.minor-architecture ENTER
./epm.install ENTER
```

The software will be installed after answering a few yes/no questions.

Installing Using the Native Distribution

The `test` target also builds a distribution in the native operating system format, if supported. Table 2.3 lists the native formats for each supported operating system and the command to run to install the software.

Table 2.3: Native Operating System Formats

Operating System	**Format**	**Command**
AIX	`aix`	`installp -d`*`directory`* `epm`
Compaq Tru64 UNIX Digital UNIX OSF/1	`setld`	`setld -a` *`directory`*
FreeBSD NetBSD OpenBSD	`bsd`	`cd` *`directory`* `pkg_add epm`
HP-UX	`depot`	`swinstall -f` *`directory`*
IRIX	`inst`	`swmgr -f` *`directory`*
Linux	`rpm`	`rpm -i` *`directory`*`/epm-4.1.rpm`
macOS	`osx`	`open` *`directory`*`/epm-4.1.pkg`
Solaris	`pkg`	`pkgadd -d` *`directory`* `epm`

3 - Packaging Your Software with EPM

This chapter describes how to use EPM to package your own software packages.

The Basics

EPM reads one or more software "list" files that describe a single software package. Each list file contains one or more lines of ASCII text containing product or file information.

Comments lines start with the # character, directive lines start with the % character, variables lines start with the $ character, and file, directory, init script, and symlink lines start with a letter.

Product Information

Every list file needs to define the product name, copyright, description, license, README file, vendor, and version:

```
%product Kung Foo Firewall
%copyright 1999-2005 by Foo Industries, All Rights Reserved.
%vendor Foo Industries
%license COPYING
%readme README
%description Kung Foo firewall software for your firewall.
%version 1.2.3p4 1020304
```

The `%license` and `%readme` directives specify files for the license agreement and README files for the package, respectively.

The `%product`, `%copyright`, `%vendor`, and `%description` directives take text directly from the line.

The `%version` directive specifies the version numbers of the package. The first number is the human-readable version number, while the second number is the integer version number. If you omit the integer version number, EPM will calculate one for you.

Files, Directories, and Symlinks

Each file in the distribution is listed on a line starting with a letter. The format of all lines is:

```
type mode owner group destination source options
```

Regular files use the letter `f` for the type field:

```
f 755 root sys /usr/bin/foo foo
```

Configuration files use the letter `c` for the type field:

```
c 644 root sys /etc/foo.conf foo.conf
```

Directories use the letter `d` for the type field and use a source path of "-":

```
d 755 root sys /var/spool/foo -
```

Finally, symbolic links use the letter `l` (lowercase L) for the type field:

```
l 000 root sys /usr/bin/foobar foo
```

The source field specifies the file to link to and can be a relative path. Just as with the `ln` command, source paths are relative to the destination directory. For example, the symbolic link */usr/bin/foobar* above points to the file */usr/bin/foo*.

Wildcards

Wildcard patterns can be used in the source field to include multiple files on a single line. The destination field contains the destination directory for the matched files:

```
f 0444 root sys /usr/share/doc/foo *.html
```

For example, if the source directory contains three HTML files, *bar.html*, *baz.html*, and *foo.html*, the wildcard line above would expand to:

```
f 0444 root sys /usr/share/doc/foo/bar.html bar.html
f 0444 root sys /usr/share/doc/foo/baz.html baz.html
f 0444 root sys /usr/share/doc/foo/foo.html foo.html
```

Subpackages

Subpackages are optional parts of your software package. For example, if your package includes developer files, you might provide them as a subpackage so that users that will not be developing add-ons to your software can omit them from the installation.

> **Note:**
> Subpackages are implemented as native subsets of the main package for the AIX, HPUX, IRIX, Solaris, and Tru64 formats and as separate packages that depend on the main (parent) package for all other formats.

To define a subpackage, use the `%subpackage` directive followed by a `%description` directive:

```
%subpackage foo
%description One-Line Description of Foo
```

Files, scripts, and dependencies that follow the `%subpackage` directive are treated as part of that subpackage. Specifying the `%subpackage` directive with no name returns processing to the main (parent) package.

You can alternate between subpackages as many times as you like:

```
%description Main package description
f 0755 /usr/bin/bar bar

%subpackage foo
%description Foo programs
f 0755 /usr/bin/foo foo
%requires bla

%subpackage
f 0644 /usr/share/man/man1/bar.1

%subpackage foo
f 0644 /usr/share/man/man1/foo.1
```

The above example creates a package containing the "bar" program and man page with a subpackage containing the "foo" program and man page. The "foo" subpackage depends both on the main package (implicit `%requires`) and another package called "bla".

Building a Software Package

The `epm(1)` program is used to build software package from list files. To build a portable software package for an application called "foo", type the following command:

```
epm foo ENTER
```

If your application uses a different base name than the list file, you can specify the list filename on the command-line as well:

```
epm foo bar.list ENTER
```

Installing the Software Package

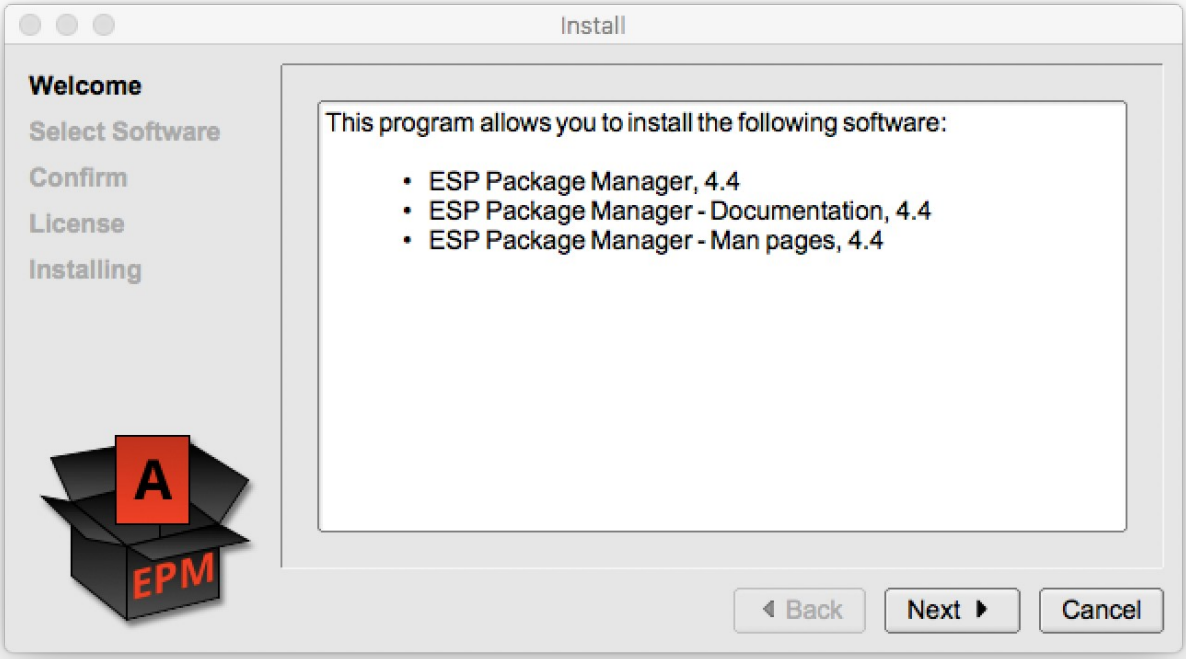
Once you have created the software package, you can install it. Portable packages include an installation script called *product.install*, where "product" is the name of the package:

```
cd os-release-arch ENTER
./product.install ENTER
```

After answering a few yes/no questions, the software will be installed. To bypass the questions, run the script with the `now` argument:

```
cd os-release-arch ENTER
./product.install now ENTER
```

Figure 3.1: The EPM Setup GUI



Including the Setup GUI

EPM also provides an optional graphical setup program (Figure 3.1). To include the setup program in your distributions, create a product logo image in GIF or XPM format and use the `--setup-image` option when creating your distribution:

```
epm --setup-image foo.xpm foo ENTER
```

This option is only supported when creating for portable and macOS software packages.

Creating Vendor Package Files

EPM can also produce vendor-specific packages using the `-f` option:

```
epm -f format foo bar.list ENTER
```

The *format* option can be one of the following keywords:

- `aix` - AIX software packages.
- `bsd` - FreeBSD, NetBSD, or OpenBSD software packages.
- `depot` or `swinstall` - HP-UX software packages.
- `dpkg` - Debian software packages.
- `inst` or `tardist` - IRIX software packages.
- `native` - "Native" software packages (RPM, INST, DEPOT, PKG, etc.) for the platform.
- `osx` - macOS software packages.
- `pkg` - Solaris software packages.
- `portable` - Portable software packages (default).
- `rpm` - Red Hat software packages.
- `setld` - Tru64 (setld) software packages.
- `slackware` - Slackware software packages.

Everything in the software list file stays the same - you just use the `-f` option to select the format. For example, to build an RPM distribution of EPM, type:

```
epm -f rpm epm
```

The result will be one or more RPM package files instead of the portable package files.

Package Files

EPM creates the package files in the output directory. As mentioned in Chapter 1, "Installing Using the Portable Distribution", the default output directory is based on the operating system name, version, and architecture. Each package format will leave different files in the output directory.

AIX Package Files

AIX packages are contained in a file called *name.bff*, where "name" is the product/package name you supplied on the command-line.

BSD Package Files

BSD packages are contained in a file called *name.tgz*, where "name" is the product/package name you supplied on the command-line.

HP-UX Package Files

HP-UX packages are contained in two files called *name.depot.gz* and *name.depot.tgz*, where "name" is the product/package name you supplied on the command-line. The *name.depot.gz* file can be supplied directly to the `swinstall(1m)` command, while the *name.depot.tgz* file contains a compressed `tar(1)` archive that can be used to install the software from CD-ROM or network filesystem.

Debian Package Files

Debian packages are contained in a file called *name.deb* or *name.deb.tgz* when there are subpackages, where "name" is the product/package name you supplied on the command-line. The *name.deb.tgz* file contains a compressed `tar` archive containing *name.deb* and *name-subpackage.deb* files that can be installed from CD-ROM, disk, or network filesystem.

IRIX Package Files

IRIX packages are contained in a file called *name.tardist*, where "name" is the product/package name you supplied on the command-line.

macOS Package Files

macOS packages are contained in a file called *name.dmg*, where "name" is the product/package name you supplied on the command-line.

RPM Package Files

RPM packages are contained in a file called *name.rpm* or *name.rpm.tgz* when there are subpackages, where "name" is the product/package name you supplied on the command-line. The *name.rpm.tgz* file contains a compressed `tar` archive containing *name.rpm* and *name-subpackage.rpm* files that can be installed from CD-ROM, disk, or network filesystem.

Slackware Package Files

Slackware packages are contained in a file called *name.tgz*, where "name" is the product/package name you supplied on the command-line.

Solaris Package Files

Solaris packages are contained in two files called *name.pkg.gz* and *name.pkg.tgz*, where "name" is the product/package name you supplied on the command-line. The *name.pkg.gz* file is a compressed package file that can be used directly with the `pkgadd(1m)` command, while the *name.pkg.tgz* file is a compressed `tar` archive that can be used to install the software from CD-ROM, disk, or network filesystem.

Tru64 Package Files

Tru64 packages are contained in a file called *name.tar.gz*, where "name" is the product/package name you supplied on the command-line.

4 - Advanced Packaging with EPM

This chapter describes the advanced packaging features of EPM.

Including Other List Files

The `%include` directive includes another list file:

```
%include filename
```

Includes can usually be nested up to 250 levels depending on the host operating system and libraries.

Dependencies

EPM supports four types of dependencies in list files: `%incompat`, `%provides`, `%replaces`, and `%requires`. Table 4.1 shows the level of support for each package format.

Table 4.1: Dependency Support

Format	%incompat	%provides	%replaces	%requires
aix	No	No	Yes	Yes
bsd	Yes	No	No	Yes
deb	Yes	Yes[1]	Yes	Yes
inst	Yes	No	Yes	Yes
osx	No	No	No	No
pkg	Yes	No	No	Yes
portable	Yes	Yes	Yes	Yes
rpm	Yes	Yes	No	Yes
setld	No	No	No	No
slackware	No	No	No	No
swinstall	No	No	Yes	Yes

1. Debian's package format does not currently support version numbers for `%provides` dependencies.

Software conflicts and requirements are specified using the `%incompat` and `%requires` directives.

If your software replaces another package, you can specify that using the `%replaces` directive. `%replaces` is silently mapped to `%incompat` when the package format does not support package replacement.

If your package provides certain functionality associated with a standard name, the `%provides` directive can be used.

Dependencies are specified using the package name and optionally the lower and upper version numbers:

```
%requires foobar
%requires foobar 1.0
%incompat foobar 0.9
%replaces foobar
%replaces foobar 1.2 3.4
%provides foobar
```

or the filename:

```
%requires /usr/lib/libfoobar.so
%incompat /usr/lib/libfoobar.so.1.2
```

Package dependencies are currently enforced only for the same package format, so a portable distribution that requires package "foobar" will only look for an installed "foobar" package in portable format.

Filename dependencies are only supported by the Debian, portable, and RPM distribution formats.

Scripts

Bourne shell script commands can be executed before or after installation, patching, or removal of the software. Table 4.2 shows the support for scripts in each package format.

The `%preinstall` and `%postinstall` directives specify commands to be run before and after installation, respectively:

```
%preinstall echo Command before installing
%postinstall echo Command after installing
```

Similarly, the `%prepatch` and `%postpatch` directives specify commands to be executed before and after patching the software:

```
%prepatch echo Command before patching
%postpatch echo Command after patching
```

Finally, the `%preremove` and `%postremove` directives specify commands that are run before and after removal of the software:

```
%preremove echo Command before removing
%postremove echo Command after removing
```

Table 4.2: Scripts Support

Format	%preinstall	%postinstall	%prepatch	%postpatch	%preremove	%postremove
aix	Yes	Yes	No	No	Yes	Yes
bsd	No	Yes	No	No	Yes	No
deb	Yes	Yes	No	No	Yes	Yes
inst	Yes	Yes	No	No	Yes	Yes
osx	Yes	Yes	No	No	No	No
pkg	Yes	Yes	No	No	Yes	Yes
portable	Yes	Yes	Yes	Yes	Yes	Yes
rpm	Yes	Yes	No	No	Yes	Yes
setld	Yes	Yes	No	No	Yes	Yes
slackware	No	Yes	No	No	No	No
swinstall	Yes	Yes	No	No	Yes	Yes

To include an external script file, use the `<filename` notation:

```
%postinstall <filename
```

To include multiple lines directly, use the `<<string` notation (a.k.a. a "here" document):

```
%postinstall <<EOF
echo Command before installing
/usr/bin/foo
EOF
```

Note that all commands specified in the list file will use the variable expansion provided by EPM, so be sure to quote any dollar sign ($) characters in your commands. For example, "$foo" is replaced by the value of "foo", but "$$foo" becomes "$foo".

Conditional Directives

The `%system` directive can match or not match specific operating system names or versions. The operating system name is the name reported by `uname` in lowercase, while the operating system version is the major and minor version number reported by `uname -r`:

`%system macos`

> Only include the following files when building a distribution for the macOS operating system.

`%system linux-2.0`

> Only include the following files when building a distribution for Linux 2.0.x.

`%system !macos !linux-2.0`

> Only include the following files when building a distribution for operating systems other than macOS and Linux 2.0.x.

The special name `all` is used to match all operating systems:

```
%system all
```

For format-specific files, the `%format` directive can be used:

`%format rpm`

> Only include the following files when building an RPM distribution.

`%format !rpm`

> Only include the following files when not building an RPM distribution.x.

`%format all`

> Include the following files for all types of distributions.

The `%arch` directive can match or not match specific architectures. The architecture name is the name reported by `uname -m`; "arm" is a synonym for "armv6", "armv7", and "armv8", "intel" is a synonym for "i386", "i486", "i586", and "i686", and "powerpc" is a synonym for "ppc":

`%arch intel`

> Only include the following files when building a package for 32-bit Intel processors.

`%arch armv6`

> Only include the following files when building a package for ARMv6 processors.

`%system !powerpc`

> Only include the following files when building a package for processors other than PowerPC.

The special name `all` is used to match all architectures:

```
%arch all
```

Finally, EPM can conditionally include lines using the `%if`, `%elseif`, `%ifdef`, `%elseifdef`, `%else`, and `%endif` directives.

`%if` directives include the text that follows if the named variable(s) are defined to a non-empty string:

```
%if FOO
f 755 root sys /usr/bin/foo foo
%elseif BAR
f 755 root sys /usr/bin/bar bar
%endif
```

`%ifdef` directives only include the text if the named variable(s) are defined to any value:

```
%ifdef OSTYPE
f 755 root sys /usr/bin/program program-$OSTYPE
%else
f 755 root sys /usr/bin/program program.sh
%endif
```

Protecting Object Files from Stripping

The `nostrip()` option can be included at the end of a file line to prevent EPM from stripping the symbols and debugging information from a file:

```
f 755 root sys /usr/lib/libfoo.so libfoo.so nostrip()
```

Software Patches

EPM supports portable software patch distributions which contain only the differences between the original and patch release. Patch files are specified using uppercase letters for the affected files. In the following example, the files */usr/bin/bar* and */etc/foo.conf* are marked as changed since the original release:

```
f 755 root sys /usr/bin/foo foo
F 755 root sys /usr/bin/bar bar
f 755 root sys /usr/share/man/man1/foo.1 foo.man
f 755 root sys /usr/share/man/man1/bar.1 bar.man
C 644 root sys /etc/foo.conf foo.conf
```

Variables

EPM imports the current environment variables for use in your list file. You can also define new variable in the list file or on the command-line when running EPM.

Variables are defined by starting the line with the dollar sign ($) followed by the name and value:

```
$name=value
$prefix=/usr
$exec_prefix=${prefix}
$bindir=$exec_prefix/bin
```

Variable substitution is performed when the variable is defined, so be careful with the ordering of your variable definitions.

Also, any variables you specify in your list file will be overridden by variables defined on the command-line or in your environment, just like with `make`. This can be a useful feature or a curse, depending on your choice of variable names.

As you can see, variables are referenced using the dollar sign ($). As with most shells, variable names can be surrounded by curly braces (`${variable}`) to explicitly delimit the name.

If you need to insert a $ in a filename or a script, use $$:

```
%install echo Enter your name:
%install read $$name
%install echo Your name is $$name.
```

Init Scripts

Initialization scripts are generally portable between platforms, however the location of initialization scripts varies greatly.

The `i` file type can be used to specify and init script that is to be installed on the system. EPM will then determine the appropriate init file directories to use and create any required symbolic links to support the init script:

```
i 755 root sys foo foo.sh
```

The previous example creates an init script named *foo* on the end-user system and will create symbolic links to run levels 0, 2, 3, and 5 as needed, using a sequence number of 00 (or 000) for the shutdown script and 99 (or 999) for the startup script.

To specify run levels and sequence numbers, use the `runlevel()`, `start()`, and `stop()` options:

```
i 755 root sys foo foo.sh "runlevel(02) start(50) stop(30)"
```

Literal Package Data

Sometimes you need to include format-specific package data such as keywords, signing keys, and response data. The `%literal(section)` directive adds format-specific data to the packages you create. Literal data is currently only supported for RPM and PKG packages.

Debian Literal Data

Debian packages have control files that provide metadata for each package. The `%literal(control)` directive can be used to provide this metadata:

```
%literal(control) <<EOF
Section: misc
Priority: extra
EOF
```

PKG Literal Data

PKG packages support request files that are used to do batch installations when installation commands require user input. The `%literal(request)` directive can be used to provide this user input:

```
%literal(request) <<EOF
John Doe
1 Any Lane
Forest Lawn, OH 12345
EOF
```

RPM Literal Data

RPM packages support numerous attributes in the "spec" file that control how the package is created and what metadata is included with the package. The `%literal(spec)` directive can be used to provide attributes for the spec file:

```
%literal(spec) <<EOF
%changelog
* Tue Aug 26 2008 John Doe <johndoe@domain.com>

- Added new feature "bar"

* Fri Aug 1 2008 John Doe <johndoe@domain.com>

- Added new feature "foo"
EOF
```

5 - EPM Packaging Examples

This chapter shows how the EPM and CUPS software is packaged using EPM list files. The EPM list file example highlights the basic features of EPM, while the CUPS list file example shows the more advanced features of EPM.

Packaging the EPM Software

The EPM software comes with its own autoconf-generated *epm.list* file that is used to package and test EPM. The EPM package consists of the main package plus a "documentation" subpackage for the documentation files and a "man" subpackage for the man pages.

We start by defining variables for each of the autoconf directory variables:

```
$prefix=/usr
$exec_prefix=/usr
$bindir=${exec_prefix}/bin
$datadir=/usr/share
$docdir=${datadir}/doc/epm
$libdir=/usr/lib
$mandir=/usr/share/man
$srcdir=.
```

Then we provide the general product information that is required for all packages; notice the use of `${srcdir}` to reference the COPYING and README files:

```
%product ESP Package Manager
%copyright 1999-2020 by Michael R Sweet, All Rights Reserved.
%copyright 2020 by Jim Jagielski, All Rights Reserved.
%vendor Michael R Sweet
%vendor Jim Jagielski
%license ${srcdir}/COPYING
%readme ${srcdir}/README.md
%description Universal software packaging tool for UNIX.
%version 4.6 460
```

After the product information, we include all of the non-GUI files that are part of EPM:

```
# Executables
%system all
f 0555 root sys ${bindir}/epm epm
f 0555 root sys ${bindir}/epminstall epminstall
f 0555 root sys ${bindir}/mkepmlist mkepmlist

# Documentation
%subpackage documentation
%description Documentation for EPM
f 0444 root sys ${docdir}/README $srcdir/README.md
f 0444 root sys ${docdir}/COPYING $srcdir/COPYING
f 0444 root sys ${docdir}/epm-book.epub $srcdir/doc/epm-book.epub
f 0444 root sys ${docdir}/epm-book.html $srcdir/doc/epm-book.html
f 0444 root sys ${docdir}/epm-book.pdf $srcdir/doc/epm-book.pdf

# Man pages
%subpackage man
%description Man pages for EPM
f 0444 root sys ${mandir}/man1/epm.1 $srcdir/doc/epm.man
f 0444 root sys ${mandir}/man1/epminstall.1 $srcdir/doc/epminstall.1
```

```
f 0444 root sys ${mandir}/man1/mkepmlist.1 $srcdir/doc/mkepmlist.1
f 0444 root sys ${mandir}/man5/epm.list.5 $srcdir/doc/epm.list.5
```

Finally, we conditionally include the GUI files depending on the state of a variable called GUIS:

```
# GUI files...
$GUIS=setup uninst

%if GUIS
%subpackage
f 0555 root sys ${libdir}/epm/setup setup
f 0555 root sys ${libdir}/epm/uninst uninst

%system macos
f 0444 root sys ${datadir}/epm/default.icns default.icns
%system all

%subpackage man
f 0444 root sys ${mandir}/man1/setup.1 $srcdir/doc/setup.1
f 0444 root sys ${mandir}/man5/setup.types.5 $srcdir/doc/setup.types.5

%endif
```

Apache License, Version 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

a. You must give any other recipients of the Work or Derivative Works a copy of this License; and

b. You must cause any modified files to carry prominent notices stating that You changed the files; and

c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

```
Copyright [yyyy] [name of copyright owner] Licensed under the Apache License,
Version 2.0 (the "License"); you may not use this file except in compliance
with the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
or agreed to in writing, software distributed under the License is
```

```
distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied. See the License for the specific language
governing permissions and limitations under the License.
```

B - Command Reference

epm(1)

Name

epm - create software packages.

Synopsis

epm [**-a** *architecture*] [**-f** *format*] [**-g**] [**-k**] [**-m** *name*] [**-n**[**mrs**]] [**-s** *setup.ext*] [**--depend**] [**--help**] [**--keep-files**] [**--output-dir** *directory*] [**--setup-image** *setup.ext*] [**--setup-program** */foo/bar/setup*] [**--setup-types** *setup.types*] [**--uninstall-program** */foo/bar/uninst*] [**-v**] [*name=value ... name=value*] *product* [*listfile*]

Description

epm(1) generates software packages complete with installation, removal, and (if necessary) patch scripts. Unless otherwise specified, the files required for *product* are read from a file named "*product*.list".

Options

The following options are recognized:

-a *architecture*
: Specifies the actual architecture for the software. Without this option the generic processor architecture is used ("intel", "sparc", "mips", etc.)

-f bsd
: Generate a BSD distribution suitable for installation on a FreeBSD, NetBSD, or OpenBSD system.

-f deb
: Generate a Debian distribution suitable for installation on a Debian-based Linux system.

-f native
: Generate a native distribution. This uses *deb* or *rpm* for Linux, *bsd* for FreeBSD, NetBSD, and OpenBSD, and *macos* for macOS. All other operating systems default to the *portable* format.

-f macos
-f macos-signed
: Generate a macOS software package. The *macos-signed* format uses the signing identity in the EPM_SIGNING_IDENTITY environment variable.

-f portable
: Generate a portable distribution based on shell scripts and tar files. The resulting distribution is installed and removed the same way on all operating systems. [default]

-f rpm
-f rpm-signed
: Generate a Red Hat Package Manager ("RPM") distribution suitable for installation on an RPM-based Linux system. The *rpm-signed* format uses the GPG private key you have defined in the ~/.rpmmacros file.

-g
: Disable stripping of executable files in the distribution.

-k

Keep intermediate (spec, etc.) files used to create the distribution in the distribution directory.

-m *name*
Specifies the platform name as a string. The default is to use the auto-generated name from the *-n* option.

-n[*mrs*]
Specifies the operating system and machine information that is included in the package name. Distributions normally are named "product-version-system-release-machine.ext" and "product-version-system-release-machine-patch.ext" for patch distributions. The "system-release-machine" information can be customized or eliminated using the appropriate trailing letters. Using *-n* by itself will remove the "system-release-machine" string from the filename entirely. The letter 'm' includes the architecture (machine). The letter 'r' includes the operating system version (release). The letter 's' includes the operating system name.

-v
Increases the amount of information that is reported. Use multiple v's for more verbose output.

--depend
Lists the dependent (source) files for all files in the package.

--output-dir *directory*
Specifies the directory for output files. The default directory is based on the operating system, version, and architecture.

-s *setup.ext*

--setup-image *setup.ext*
Include the ESP Software Wizard with the specified image file with the distribution. This option is currently only supported by portable distributions.

--setup-program */foo/bar/setup*
Specifies the setup executable to use with the distribution. This option is currently only supported by portable distributions.

--setup-types *setup.types*

Specifies the *setup.types* file to include with the distribution. This option is currently only supported by portable distributions.

--uninstall-program */foo/bar/uninst*
Specifies the uninst executable to use with the distribution. This option is currently only supported by portable distributions.

Environment

The following environment variables are supported by **epm**:

EPM_SIGNING_IDENTITY
The common name that should be used when signing a package.

List Files

The EPM list file format is now described in the *epm.list(5)* man page.

See Also

epminstall(1), **mkepmlist**(1), **epm.list**(5), **setup**(1).

Copyright

epminstall(1)

Name

epminstall - add a directory, file, or symlink to a list file.

Synopsis

epminstall [*options*] *file1file2...fileNdirectory*
epminstall [*options*] *file1file2*
epminstall [*options*] **-d** *directory1directory2...directoryN*

Description

epminstall adds or replaces a directory, file, or symlink in a list file. The default list file is "epm.list" and can be overridden using the *EPMLIST* environment variable or the *--list-file* option.

Entries are either added to the end of the list file or replaced in-line. Comments, directives, and variable declarations in the list file are preserved.

Options

epminstall recognizes the standard Berkeley **install**(8) command options:

-b
: Make a backup of existing files (ignored, default for **epm**.)

-c
: BSD old compatibility mode (ignored.)

-g *group*
: Set the group owner of the file or directory to *group*. The default group is "sys".

-m *mode*
: Set the permissions of the file or directory to *mode*. The default permissions are 0755 for directories and executable files and 0644 for non-executable files.

-o *owner*
: Set the owner of the file or directory to *owner*. The default owner is "root".

-s
: Strip the files (ignored, default for **epm**.)

--list-file *filename.list*
: Specify the list file to update.

See Also

epm(1), mkepmlist(1), epm.list(5).

Copyright

mkepmlist(1)

Name

mkepmlist - make an epm list file from a directory.

Synopsis

mkepmlist [**-g** *group*] [**-u** *user*] [**--prefix** *directory*] *directory* [... *directory*]

Description

mkepmlist(1) recursively generates file list entries for files, links, and directories. The file list is send to the standard output.

Options

mkepmlist supports the following options:

-g *group*
: Overrides the group ownership of the files in the specified directories with the specified group name.

-u *user*
: Overrides the user ownership of the files in the specified directories with the specified user name.

--prefix *directory*
: Adds the specified directory to the destination path. For example, if you installed files to "/opt/foo" and wanted to build a distribution that installed the files in "/usr/local", the following command would generate a file list that is installed in "/usr/local":

```
mkepmlist --prefix=/usr/local /opt/foo >foo.list
```

See Also

epm(1), **epminstall**(1), **epm.list**(5).

Copyright

setup(1)

Name

setup - graphical setup program for the esp package manager

Synopsis

setup [*directory*]

Description

setup(1) provides a graphical installation interface for EPM-generated portable installation packages. It presents a step-by-step dialog for collecting a list of packages to install and accepting any license agreements for those packages.

setup searches for products in the current directory or the directory specified on the command-line.

Installation Types

The default type of installation is "custom". That is, users will be able to select from the list of products and install them.

setup also supports other types of installations. The *setup.types* file, if present, defines the other installation types.

See Also

epm(1), **setup.types**(5).

Copyright

C - List File Reference

This appendix provides a complete reference for the EPM list file and setup types formats.

The EPM List File Format

Each *EPM* product has an associated list file that describes the files to include with the product. Comment lines begin with the "#" character and are ignored. All other non-blank lines must begin with a letter, dollar sign ("$"), or the percent sign ("%").

List File Directives

The following list describes all of the list file directives supported by *EPM*:

$name=value

Sets the named variable to *value*. **Note:** Variables set in the list file are overridden by variables specified on the command-line or in the current environment.

%copyright *copyright notice*

Sets the copyright notice for the file.

%description *description text*

Adds a line of descriptive text to the distribution. Multiple lines are supported.

%format *format [... format]*

Uses following files and directives only if the distribution format is the same as *format*.

%format !*format [... format]*

Uses following files and directives only if the distribution format is not the same as *format*.

%include *filename*

Includes files listed in *filename*.

%incompat *product*
%incompat *filename*

Indicates that this product is incompatible with the named product or file.

%if *variable [... variable]*
%if !*variable [... variable]*
%ifdef *variable [... variable]*
%ifdef !*variable [... variable]*
%elseif *variable [... variable]*

%elseif !*variable [... variable]*
%elseifdef *variable [... variable]*
%elseifdef !*variable [... variable]*
%else
%endif

Conditionally includes lines in the list file. The *%if* lines include the lines that follow if the named variables are (not) defined with a value. The *%ifdef* lines include the lines that follow if the named variables are (not) defined with any value. These conditional lines cannot be nested.

%install *script or program*

Specifies a script or program to be run after all files are installed. (This has been obsoleted by the %postinstall directive)

%license *license file*

Specifies the file to display as the software license.

%packager *name of packager*

Specifies the name of the packager.

%patch *script or program*

Specifies a script or program to be run after all files are patched. (This has been obsoleted by the %postpatch directive)

%postinstall *script or program*
%postinstall <*scriptfile*
%postinstall <<*string*

Specifies a script or program to be run after all files are installed.

%postpatch *script or program*
%postpatch <*scriptfile*
%postpatch <<*string*

Specifies a script or program to be run after all files are patched.

%postremove *script or program*
%postremove <*scriptfile*
%postremove <<*string*

Specifies a script or program to be run after removing files.

%preinstall *script or program*
%preinstall <*scriptfile*
%preinstall <<*string*

Specifies a script or program to be run before all files are installed.

%prepatch *script or program*
%prepatch *<scriptfile*
%prepatch *<<string*

Specifies a script or program to be run before all files are patched.

%preremove *script or program*
%preremove *<scriptfile*
%preremove *<<string*

Specifies a script or program to be run before removing files.

%product *product name*

Specifies the product name.

%provides *product name*

Indicates that this product provides the named dependency.

%readme *readme file*

Specifies a README file to be included in the distribution.

%remove *script or program*

Specifies a script or program to be run before removing files. (This has been obsoleted by the %preremove directive)

%release *number*

Specifies the release or build number of a product (defaults to 0).

%replaces *product*

Indicates that this product replaces the named product.

%requires *product*
%requires *filename*

Indicates that this product requires the named product or file.

%system *system[-release] [... system[-release]]*

Specifies that the following files should only be used for the specified operating systems and releases.

%system !*system[-release] [... system[-release]]*

Specifies that the following files should not be used for the specified operating systems and releases.

%system all

Specifies that the following files are applicable to all operating systems.

%vendor *vendor or author name*

Specifies the vendor or author of the product.

%version *version number*

Specifies the version number of the product.

c *mode user group destination source*
C *mode user group destination source*

Specifies a configuration file for installation. The second form specifies that the file has changed or is new and should be included as part of a patch. Configuration files are installed as "destination.N" if the destination already exists.

d *mode user group destination -*
D *mode user group destination -*

Specifies a directory should be created when installing the software. The second form specifies that the directory is new and should be included as part of a patch.

f *mode user group destination source [nostrip()]*
F *mode user group destination source [nostrip()]*

Specifies a file for installation. The second form specifies that the file has changed or is new and should be included as part of a patch. If the "nostrip()" option is included, the file will not be stripped before the installation is created.

f *mode user group destination source/pattern [nostrip()]*
F *mode user group destination source/pattern [nostrip()]*

Specifies one or more files for installation using shell wildcard patterns. The second form specifies that the files have changed or are new and should be included as part of a patch. If the "nostrip()" option is included, the file will not be stripped before the installation is created.

i *mode user group service-name source ["options"]*
I *mode user group service-name source ["options"]*

Specifies an initialization script for installation. The second form specifies that the file has changed or is new and should be included as part of a patch. Initialization scripts are stored in */etc/software/init.d* and are linked to the appropriate system-specific directories for run levels 0, 2, 3, and 5. Initialization scripts **must** accept at least the *start* and *stop* commands. The optional *options* following the source filename can be any of the following:

order(*string*)
Specifies the relative startup order compared to the required and used system functions. Supported values include First, Early, None, Late, and

Last (macOS only).

provides(*name(s)*)
: Specifies names of system functions that are provided by this startup item (macOS only).

requires(*name(s)*)
: Specifies names of system functions that are required by this startup item (macOS only).

runlevels(*levels*)
: Specifies the run levels to use.

start(*number*)
: Specifies the starting sequence number from 00 to 99.

stop(*number*)
: Specifies the ending sequence number from 00 to 99.

uses(*name(s)*)
: Specifies names of system functions that are used by this startup item (macOS only).

l *mode user group destination source*
L *mode user group destination source*

: Specifies a symbolic link in the installation. The second form specifies that the link has changed or is new and should be included as part of a patch.

R *mode user group destination*

: Specifies that the file is to be removed upon patching. The *user* and *group* fields are ignored. The *mode* field is only used to determine if a check should be made for a previous version of the file.

List Variables

EPM maintains a list of variables and their values which can be used to substitute values in the list file. These variables are imported from the current environment and taken from the command-line and list file as provided. Substitutions occur when the variable name is referenced with the dollar sign ($):

```
%postinstall <<EOF
echo What is your name:
read $$name
echo Your name is $$name
EOF

f 0555 root sys ${bindir}/foo foo
f 0555 root sys $datadir/foo/foo.dat foo.dat
```

Variable names can be surrounded by curly brackets (${name}) or alone ($name); without brackets the name is terminated by the first slash (/), dash (-), or whitespace. The dollar sign can be inserted using $$.

The setup.types File

The EPM **setup** program normally presents the user with a list of software products to install, which is called a "custom" software installation.

If a file called *setup.types* is present in the package directory, the user will instead be presented with a list of installation types. Each type has an associated product list which determines the products that are installed by default. If a type has no products associated with it, then it is treated as a custom installation and the user is presented with a list of packages to choose from.

The *setup.types* file is an ASCII text file consisting of type and product lines. Comments can be inserted by starting a line with the pound sign (#). Each installation type is defined by a line starting with the word TYPE. Products are defined by a line starting with the word INSTALL:

```
# Pre-select the user packages
TYPE Typical End-User Configuration
INSTALL foo
INSTALL foo-help

# Pre-select the developer packages
TYPE Typical Developer Configuration
INSTALL foo
INSTALL foo-help
INSTALL foo-devel
INSTALL foo-examples

# Allow the user to select packages
TYPE Custom Configuration
```

In the example above, three installation types are defined. Since the last type includes no products, the user will be presented with the full list of products to choose from.