Contents

Chapter 1

Mounting the CD

CDs are created to conform to the ISO-9660 standard with Rock Ridge extensions. The examples below assume a particular SCSI ID or device path for the CD-ROM drive. Please adjust for your configuration.

1.1 AIX 4.3.x

Issue the following command to mount the CD-ROM:

```
$ mount -rv cdrfs /dev/cd0 /mnt
```

1.2 Compaq Tru64 UNIX 4.x, 5.x

Tru64 UNIX 4.0D does not, by default, understand the Rock Ridge extensions. Therefore, the `mount` comand must be instructed to interpret the Rock Ridge Extensions to mount the CD. On Tru64 UNIX 4.0D, issue the following command to mount the CD:

```
$ mount -t cdfs -o ro,rrip /dev/rz4c /mnt
```

On Tru64 UNIX 5.x, which does understand the Rock Ridge extension by default, issue the following command to mount the CD:

```
$ mount -t cdfs -o ro /dev/disk/cdrom0c /mnt
```

1.3 HP-UX 10.20

The HP-UX 10.20 `mount(1M)` command does not support the Rock Ridge extension to ISO-9660 CDs.

Because the `mount(1M)` command cannot be patched to support the Rock Ridge formatted CDs, the `pfs` daemons must be started prior to mounting the CD. The `pfs` suite of utilities, shipped with the OS, provide the ability to support the Rock Ridge extensions. To use the `pfs` tools, two daemons must be started, invoked with the following commands:

```
$ pfs_mountd &
```

```
    $ pfsd &
```

Once the above is done, the following commands are used to mount and unmount the CD:

```
    $ pfs_mount -t rrip -o ro /dev/dsk/c0t4d0 /mnt
    $ pfs_umount /mnt
```

The `pfsd` and `pfs_mountd` programs must be killed once the CD is unmounted.

1.4 HP-UX 11.x

The `PHCO_26449`, `PHKL_26450`, and `PHKL_28060` patches must be installed to add Rock Ridge support for HP-UX 11.00 and the `PHKL_26269`, `PHCO_25841`, and `PHKL_28025` patches must be installed for HP-UX 11i (11.11).

Once the above patches have been applied, the following command can be used to mount the CD:

```
    $ mount -o ro,rr /dev/dsk/c0t4d0 /mnt
```

1.5 IRIX 6.x

If the IRIX `mediad` daemon is running, the CD will mount automatically on `/CDROM`. If `mediad` is not running, issue the following command to mount the CD:

```
    $ mount -t iso9660 -o ro /dev/rdsk/dks0d4vol /mnt
```

1.6 Redhat Linux 7.x

During installation of Redhat Linux, if a CD-ROM is detected, it will be given the device name `/dev/cdrom`. An entry will also be created in `/etc/fstab` to mount the CD-ROM on `/mnt/cdrom`. If such an entry exists, executing the following with mount the CD-ROM:

```
    $ mount /mnt/cdrom
```

If a pre-defined entry does not exist for the CD-ROM, it can be mounted as follows:

```
    $ mount -o ro -t iso9660 /dev/scd0 /mnt
```

1.7 Solaris 2.x

If the Solaris `vold` daemon is running, the CD will automatically mount to `/cdrom/cdrom#`. If `vold` is not running, issue the following command to mount the CD:

```
    $ mount -F hsfs -o ro /dev/dsk/c0t4d0s0 /mnt
```

Chapter 2

QuickStart Guide

CDs are organized along the following directory hierarchy:

```
INSTALL.pdf
MANIFEST
README
RELEASE-NOTES.pdf
cd/
cd/depot-db.xml
cd/[platform]/
package-list
package-notes/
support/
```

`INSTALL.pdf`:	Installation documentation
`MANIFEST`:	Description of contents of CD
`RELEASE-NOTES.pdf`:	Release notes for this release
`cd/[platform]/`:	Pre-compiled binaries in the native vendor package-manager format or RPM
`package-list`:	List of packages on the CD
`package-notes/`:	Package notes and licensing information for packages on the CD
`support/`:	Installation and support tools on the first binary CD. Contains the `pbutils`, `pkgutils`, and `sbutils` tools.

The `pkgutils` suite should be the first package installed. Once installed, the `pkg-inst` tool documented in section 4.1 can be used to install the packages on each CD or in a remote repository.

Table 2.1 lists the commands needed to install the packages that comprise the `pkgutils` suite depending on the platform. The instructions assume the CD-ROM is mounted on '`/cdrom`'. The packages are available in the "`support`" directory of the first binary CD-ROM or via ftp at "`ftp://support.thewrittenword.com/dists/5.1/support`". The latest version is always available on the ftp site. If installing RPM packages, the RPM package must be installed first, followed by the `pkgutils` suite. Table 2.2 lists the commands to install RPM. Section 10.1 should be consulted after RPM is installed to configure the RPM database.

The `pkgutils` suite, once installed, will reside in `/opt/TWWfsw/pkgutils15` with symbolic links for the binaries in `/opt/TWWfsw/bin` and the man pages in `/opt/TWWfsw/man`.

Table 2.1: Installing the pkgutils suite

Platform	**Installation command**
AIX 4.3.2	`$ installp -acqQ -d /cdrom/support/TWW.pkgutils15.bff TWW.pkgutils15`
HP-UX 10.20, 11.x	`$ swinstall -s /cdrom/support TWWpkgut15`
IRIX 6.x	`$ inst -a -f /cdrom/support -I TWWpkgutils15`

RPM[1]	`$ cd /cdrom/support`
	`$ rpm -Uv TWWpkgutils15-1.5.11-1.`*arch*[2]`.rpm`
	`$ rpm -Uv TWWpkgutils15-conf-1.5.11-1.`*arch*[2]`.rpm`
	`$ rpm -Uv TWWpkgutils15-man-1.5.11-1.`*arch*[2]`.rpm`

Solaris 2.x	`$ pkgadd -d /cdrom/support TWWpkg15c`
	`$ pkgadd -d /cdrom/support TWWpkg15m`
	`$ pkgadd -d /cdrom/support TWWpkg15u`

Tru64 UNIX 4.0D	`$ cd /cdrom/support`
	`$ setld -l .  TWWPKGUTILS15CONF425`
	`$ setld -l .  TWWPKGUTILS15BIN425`
	`$ setld -l .  TWWPKGUTILS15MAN425`

Tru64 UNIX 5.1	`$ cd /cdrom/support`
	`$ setld -l .  TWWPKGUTILS15CONF510`
	`$ setld -l .  TWWPKGUTILS15BIN510`
	`$ setld -l .  TWWPKGUTILS15MAN510`

[1] The RPM package manager might need to be configured before it can be used. Chapter 10 contains information on configuring RPM.

[2] *arch* is dependent on the platform: AIX (“`ppc`”); IRIX (“`mipseb`”); HP-UX (“`parisc`”); Redhat Linux (“`i386`”); Solaris (“`sparc`”); Tru64 UNIX (“`alpha`”)

Table 2.2: Installing RPM

Platform	**Installation command**
AIX 4.3.2	`$ installp -acqQ -d /cdrom/support/TWW.rpm40.bff TWW.rpm40`
HP-UX 10.20, 11.x	`$ swinstall -s /cdrom/support TWWrpm40`
IRIX 6.x	`$ inst -a -f /cdrom/support -I TWWrpm40`

Solaris 2.x	`$ pkgadd -d /cdrom/support TWWrpm40c`
	`$ pkgadd -d /cdrom/support TWWrpm40m`
	`$ pkgadd -d /cdrom/support TWWrpm40u`

Tru64 UNIX 4.0D	`$ cd /cdrom/support`
	`$ setld -l .  TWWRPM40BIN425`
	`$ setld -l .  TWWRPM40CONF425`
	`$ setld -l .  TWWRPM40MAN425`

Tru64 UNIX 5.1	`$ cd /cdrom/support`
	`$ setld -l .  TWWRPM40BIN510`
	`$ setld -l .  TWWRPM40CONF510`
	`$ setld -l .  TWWRPM40MAN510`

Chapter 3

Package Overview

All packages are installed using either the `pkg-inst` tool or the vendor-supported package management system (the use of `pkg-inst` is strongly recommended). Due to the differences between the package management systems on each of our supported platforms, installation, depending on the platform, is different unless using the `pkg-inst` tool. This document outlines how packages are installed, upgraded, and removed. Some detail is also provided on the workarounds implemented to achieve uniformity in installation for all package management systems supported.

3.1 Installation paths

All packages install into `/opt/TWWfsw/`*`package`*. For example, GNU bash is installed into `/opt/TWWfsw/bash`, less to `/opt/TWWfsw/less`, and GIMP to `/opt/TWWfsw/gimp10`. For some packages, the major and/or minor number might appear in '*package*'. Examples of this are the GTK+ 1.2.6 library installed as `/opt/TWWfsw/gtk+12`, the Glib 1.2.6 library installed as `/opt/TWWfsw/glib12`, and XEmacs 20.4 installed as `/opt/TWWfsw/xemacs20`. Installation for these packages deviates from the norm to support multiple versions.

3.1.1 Package configuration files

Configuration files are installed into one of three directories:

1. `/opt/TWWfsw/`*`package`*`/`*`config`*
2. `/etc/init.d/`*`config`*
3. `/etc/opt/TWWfsw/`*`package`*`/`*`config`*

Example of files in category #1 are the configuration files from the tin news reader, "`/opt/TWWfsw/tin/etc/tin.defaults`", "`/opt/TWWfsw/tin/etc/nntpserver`", and "`/opt/TWWfsw/tin/etc/NNTP_INEWS_DOMAIN`". Examples of files in category #2 are startup scripts for Apache and Samba, "`/etc/init.d/TWWapache1320`" and "`/etc/init.d/TWWsamba221`". Examples of files in category #3 are configuration files for TCP Wrappers, "`/etc/opt/TWWfsw/tcpwrap/hosts.allow`" and "`/etc/opt/TWWfsw/tcpwrap/hosts.deny`".

While a default location is always provided for the location of the configuration files to be used by binaries of the relevant programs, their location can usually be changed by a command-line option or environment variable.

The rationale for selecting whether or not a configuration file should be installed using category #1 or #3 is determined based on how often the configuration file will be used. In the case of Apache, more than one web server might not necessarily share the same '`httpd.conf`' file. However, for a package such as '`tin`', a configuration file containing the name of the NNTP server will be shared by the majority of users running tin. Moreover, if '`tin`' is installed to an NFS-mounted area, a change to the configuration file will allow an immediate update to all clients as the configuration file would also reside on the NFS server. And, for those clients that need a separate configuration file, which should be few, a custom startup script or environment variable can be set.

Configuration files are managed using the following criteria:

1. Configuration files are explicitly marked in a package
2. Some configuration files are treated as "upgradable" (files from category #2 above won't be but those from categories #1 and #3 will be). Configuration files thus marked will be copied from the old installation path to the new installation path when a new version is installed and the installation path changes.
3. If a configuration file is treated as upgradable, copy the version from the old installation path and rename the version installed by the package as *`file`*`.tww-orig` **only** if the original configuration file has changed. This is governed by the installation configuration variable "`CONFIG_UPGRADE`". The default value is "`latest`" which indicates the copy should be performed. Any other value will prevent the copy. When the old configuration file is copied over, replace the string "`/opt/TWWfsw/`*`old package`*" with "`/opt/TWWfsw/`*`new package`*" in the new configuration file.
4. If a configuration file is not treated as upgradable and the package is reinstalled, do not replace the configuration file if it has the same checksum as what was originally installed (if the file changes it is not overwritten).
5. By default, local changes are preserved and new configuration files that would have overwritten the original version are saved as *`file`*`.tww-new`. This file is automatically removed when the package is removed but is available to sync up the new configuration file against the local changed version.

RPM is intelligent enough to remove files from an old package that are not present in the new package. Thus, we let RPM deal with configuration files completely except for new RPM packages where the installation path is different. In this case, a postinstall script will run at package installation time to copy the old configuration file to the new installation path. Due to this "intelligence", we will only mark configuration files as upgradable for new packages which install into a different directory than the older version.

For AIX lpp, Solaris pkgadd, HP-UX depot, Tru64 UNIX setld, and IRIX inst, configuration files are managed by the postinstall script. With the exception of Solaris, the various system tools used to generate a listing of the files for a package will not include the configuration files. The postinstall script creates a file named "`.tww-toc`" in each directory containing a configuration file to store the name of the configuration file and a checksum so the above can be accomplished. For RPM, the file is called "`.tww-config-inst`", and simply indicates if the configuration files from an older version have been copied over.

3.1.2 Startup scripts

Startup scripts are sometimes created in '`/etc/init.d`' or '`/sbin/init.d`' with appropriate links in '`/etc/rc*.d`' or '`/sbin/rc*.d`' depending on the platform. Startup scripts are installed as part of the local configuration package and are automatically removed when the package is removed. Before the local configuration package is removed, the '`stop`' shutdown action in the startup scripts should be invoked to stop any running daemons executed by the package. This is not automatically performed when the local configuration package is removed. On HP-UX, if the daemons invoked by the startup script are not stopped prior to removing the runtime package, the daemons will not be removed because the daemons are still executing.

All startup scripts are marked as configuration files but not marked as upgradable.

3.2 Installation configuration files

A configuration file is used by the package installation scripts for all platforms to read defaults governing package installation. The configuration file is searched for in the following order:

1. `/opt/TWWfsw/etc/tww-inst-configs/`*`package`*
2. `/opt/TWWfsw/etc/tww-inst-configs/default`
3. `/etc/opt/TWWfsw/configs/`*`package`*
4. `/etc/opt/TWWfsw/configs/default`

The configuration files '`/opt/TWWfsw/etc/tww-inst-configs/`*`package`*' and '`/etc/opt/TWWfsw/configs/`*`package`*' allow per-package defaults.

It is important to note that the configuration filename does not match the name of the package but the installation component of the pathname. In addition, the configuration file for programs with 3rd-party packages such as Perl and XEmacs does not contain the '`p`' extension at the end. Similarly for packages with the library runtime component. Thus, the configuration filename for ghostscript 5.50 which installs into '`/opt/TWWfsw/ghostscript`' is '`/etc/opt/TWWfsw/configs/ghostscript`', the configuration filename for ghostscript 7.05 which installs into '`/opt/TWWfsw/ghostscript70`' is '`/etc/opt/TWWfsw/configs/ghostscript60`', and the configuration filename for perl 5.6.1 which installs into '`/opt/TWWfsw/perl561`' and '`/opt/TWWfsw/perl561p`' is '`/etc/opt/TWWfsw/configs/perl561`'. The `pkg-info` program with the '`--print=install-path`' option is useful in determining the installation path component of a package.

```
$ pkg-info --depot=[cd mount point] --print=install-path perl
perl
  depot containing v5.8.0-6 of this package ...
    file:///[cd mount point]
  installation path component for v5.8.0 ... perl580
  instances ... TWWpl580m TWWpl580p TWWpl580u

  depot containing v5.6.1-19 of this package ...
    file:///[cd mount point]
  installation path component for v5.6.1 ... perl561
```

```
    instances ... TWWpl561m TWWpl561p TWWpl561u
```

3.2.1 Common bin, info, man links (`COMMON_BASE`)

When packages are installed, a set of links is made for binaries, info, and man pages. The default path used as the base directory for the links is '`/opt/TWWfsw`'. This default can be altered by adding the following variable assignment to the configuration file:

```
COMMON_BASE=<base directory>
```

A value of '`none`' indicates that common links should not be created.

Solaris packages prompt for the value of this variable except when specified in the configuration file. For RPM packages, the variable name to set the common base directory is '`COMMON_BASE_`*systype*' where the value of '*systype*' is determined by table 3.1. Because RPM provides a cross-platform packaging solution, '`COMMON_BASE`' differs to allow for installation of different architectures on the same platform for access from NFS clients.

Table 3.1: `COMMON_BASE` *systype* definitions

Platform	***systype* Value**
AIX 4.3.2	`powerpc-ibm-aix4.3.2.0`

HP-UX 10.20
HP-UX 11.00
HP-UX 11.11
	`hppa1.1-hp-hpux10.20`
	`hppa1.1-hp-hpux11.00`
	`hppa1.1-hp-hpux11.11`

IRIX 6.5	`mips-sgi-irix6.5`
Redhat Linux	`i686-pc-linux-gnu`

Solaris 2.5.1
Solaris 2.6
Solaris 7
Solaris 8
Solaris 9
	`sparc-sun-solaris2.5.1`
	`sparc-sun-solaris2.6`
	`sparc-sun-solaris2.7`
	`sparc-sun-solaris2.8`
	`sparc-sun-solaris2.9`

Tru64 UNIX 4.0D Tru64 UNIX 5.1	`alpha-dec-osf4.0d` `alpha-dec-osf5.1`

Thus, setting this variable to '`/opt/`' will create links in '`/opt/bin`', '`/opt/sbin`', '`/opt/info`', and '`/opt/man`'. Setting the variable to '`/opt/TWWfsw`' (note the omission of the trailing slash) will create links in '`/opt/TWWfswbin`', '`/opt/TWWfswsbin`', '`/opt/TWWfswinfo`', and '`/opt/TWWfswman`'. The '`bin`', '`sbin`', '`info`', and '`man`' path components are appended to the value of '`COMMON_BASE`'.

Should you install a package without having common directories created and wish to do so in the future, the '`pkg-config`' (cf. section 4.4) program can be used. This is a requirement for Solaris pkgadd and IRIX inst which do not allow separate execution of the postinstall scripts. The package management systems on HP-UX and Tru64 UNIX allow configuration and unconfiguration of a package after it has been loaded. The configuration script included with each package would create the symbolic links while the unconfiguration script would remove the symbolic links. More detail is given below under the respective platform about the commands to achieve this. Once the links are created in the common directories, the links are registered as belonging to the package they create a link to. Because of this, removing packages automatically removes the links created in the common directories during the time of the install. A separate program does not need to be run to prune links in the common directories.

In addition to the creation of symbolic links, info entries are added to the '`dir`' file in the common info directory. Their entries will be removed when the package is deleted. To support this, all packages contain a statically-linked version of the texinfo '`install-info`' utility in the '`/opt/TWWfsw/`*`package`*`/tww-inst`' directory. Some architectures contain more files in this directory to aide in managing the creation and removal of links at package installation and removal time.

3.2.2 Verbose output (`INST_VERBOSE`)

The '`INST_VERBOSE`' variable in the global configuration file specifies verbosity of output during the installation process. Valid values are 0 thru 9 and formatted as:

```
INST_VERBOSE=[0-9]
```

Setting the verbosity to 9 invokes a '`set -x`' on the installation scripts. Currently, only values of 0, 1 and 9 have meaning. The default value is '`1`'. Setting the value to '`0`' generates no output from preinstall and postinstall scripts.

3.2.3 Upgrades

`INST_UPGRADE`

When a package is upgraded from an older release to a newer release, the installation path either remains the same or changes. When the path remains the same, the links created by the initial install for the common directories will be maintained for the new release, providing the destination of the links is still valid. If the installation path changes, the default action of the postinstall script when attempting to create the common links will be to replace the existing common links, pointing to the old path, with links pointing to the new path. A value of '`none`' causes a warning message without replacing the links. The default value of '`INST_UPGRADE`' is '`latest`', indicating the installation of the new release will remove links created in the common directories to all old releases and recreate them pointing to the new release. This is only done for links that conflict between releases.

`CONFIG_UPGRADE`

When a new version of an existing package is installed with a different installation path (e.g. Apache 1.3.14 installs into `/opt/TWWfsw/apache1314` and Apache 1.3.20 installs into `/opt/TWWfsw/apache1320`), this variable governs whether or not the configuration files from the previous version should be copied over for the new version. The default, "`latest`", allows the copy. Any other value will prevent the copy.

Section 3.1.1 contains more details about package configuration files.

3.3 Package dependencies

With the exception of Tru64 UNIX, all packages are built with dependency information. In the case of Tru64 UNIX, the package management system is not robust enough to make dealing with dependencies and upgrades simple. If installing packages with the `pkg-inst` tool, dependencies are automatically handled even if the package has not been built with dependency information as part of the package.

Chapter 4

Package Utilities Suite (pkgutils)

The `pkgutils` suite contains tools to assist in configuring, installing, managing, and upgrading an installation. The tools that comprise the suite work on all supported platforms regardless of the underlying package management system. With these tools, very little knowledge of the package management system is required.

4.1 Installing packages (`pkg-inst`)

The `pkg-inst` tool is part of the `pkgutils` suite. It is used to install and upgrade a package. In the architecture directory of the CD distribution is a package database file named '`pkg-db.xml`' that drives `pkg-inst`. The package database file contains information about all packages in the directory it resides in and what dependencies each package has. With this knowledge, `pkg-inst` is able to install and upgrade packages regardless of the underlying package management system.

General usage for `pkg-inst` is:

```
$ pkg-inst --depot=[depot] [packages ...]
```

Multiple depot directories can be specified, with each expected to contain either a '`pkg-db.xml`' file and/or a '`depot-db.xml`' file. The '`depot-db.xml`' file contains information about subdirectories that are to be descended into to search for additional '`depot-db.xml`' and '`pkg-db.xml`' files. Because of this file, it is possible for `pkg-inst` to recursively descend into directories to search for packages. By default, `pkg-inst` will descend into all subdirectories listed in the '`depot-db.xml`' file. The `--update-type` option can be used to restrict where `pkg-inst` packages are searched. The general layout of a depot is:

```
[depot]/cd/depot-db.xml
[depot]/cd/dists/depot-db.xml
[depot]/cd/dists/5.1
[depot]/cd/dists/5.1/depot-db.xml
[depot]/cd/dists/5.1/errata/
[depot]/cd/dists/5.1/errata/depot-db.xml
[depot]/cd/dists/5.1/errata/sparc-sun-solaris2.8/
[depot]/cd/dists/5.1/errata/sparc-sun-solaris2.8/pkg-db.xml
```

By default, all subpackage components of a package will be installed. When dependencies are required for a package and the '`--ignoredeps`' option is not given, only those subpackage components needed to meet the dependency will be installed.

The following is a quick example of how much easier package installation is with `pkg-inst`. It is a sample run of the installation of GNU zip (gzip) on Solaris. Examples using the native package management system are given in the chapters to follow describing how to install, remove, and updates packages using the native package management system. Comparing the two, it should be easy to see how much simpler `pkg-inst` makes the installation process.

```
$ pkg-inst --depot=/tmp/depot gzip
gzip
  depot containing v1.3.5-1 of this package ...
    file:///tmp/depot
  checking if already installed ... no
    components to install: TWWgzp13m, TWWgzp13u, TWWgzp13d
  checking for dependencies ... none
  installing TWWgzp13m ...
    $ pkgadd -a /opt/TWWfsw/pkgutils15/share/pkgadd/admin -d
    /opt/dist/cd/TWWgzp13m TWWgzp13m
    ...
    updating package database entry for man fileset ... done
  installing TWWgzp13u ...
    $ pkgadd -a /opt/TWWfsw/pkgutils15/share/pkgadd/admin -d
    /opt/dist/cd/TWWgzp13u TWWgzp13u
    ...
    updating package database entry for runtime fileset ... done
  installing TWWgzp13d ...
    $ pkgadd -a /opt/TWWfsw/pkgutils15/share/pkgadd/admin -d
    /opt/dist/cd/TWWgzp13d TWWgzp13d
    ...
    updating package database entry for doc fileset ... done
```

The following command can be used to install the latest versions of all packages in a depot. However, it will not necessarily install all packages in the depot. If more than one version of a package is present in the depot (e.g. Tcl 8.0, 8.1, and 8.2) only the latest version will be installed (e.g. Tcl 8.2).

```
$ pkg-inst --depot=[depot]
```

When upgrading to a new release, it might not be desirable to install all packages. Rather, it might be more desirable to install only newer versions of those packages are already installed. The '`--match-target`' option, with the appropriate argument, will install only those packages in the depot that are already installed on the target. Thus, if the current version of Tcl on the target is 8.0.4 and the depot contains both Tcl 8.0.5 and Tcl 8.1.1, the following command will install only the 8.0.5 version whereas the above command would have installed both the 8.0.5 and 8.1.1 versions:

```
$ pkg-inst --match-target=package --depot=[depot] tcl
```

To upgrade to the latest version of tcl, use '`--match-target=name`'. In the case above, if the target contains Tcl 8.0.5 and Tcl 8.0.5 and Tcl 8.1.1 are available, v8.1.1 would be installed with:

```
$ pkg-inst --match-target=name --depot=[depot] tcl
```

The '`--match-target=package`' option matches against an installed package instance, what the package manager knows the name of the package as. In the case of Tcl 8.0.4, for Solaris/pkgadd this is '`TWWtcl80`'. With the '`--match-target=name`' option, `pkg-inst` bases the upgrade on what it knows the package name as. In the case of Tcl 8.0.4, 8.0.5, and 8.1.1, the name is '`tcl`'.

The '`--remove-prev-ver`' option will remove older versions of all packages installed by `pkg-inst`, providing the older version is not a dependency for another package. It does this by building a running list of successfully installed packages, determining the previous versions using the global `pkg-db.xml` file, and then invoking `pkg-rm` to remove the packages (without the '`--add-inverse-deps`' option). This method has the side effect of requiring more disk space because, if upgrading from 3.2 to 3.3, all 3.2 and 3.3 packages will be installed before package deletion begins. Because `pkg-inst` must wait until all packages are installed to determine which were successfully installed, just-in-time deletion of older packages cannot be performed because a package selected for installation but not yet installed might be dependent on an older version of an already-installed package.

4.1.1 Package installation database

For each subpackage installed, an entry is written to a package database directory specified with the `--pkg-db=`*`path`* entry. Files in the database directory are named after the installed package and in XML format, similar to the '`pkg-db.xml`' file. These files are updated as packages are installed, updated, and removed. The default path for the package database is `/var/opt/TWWfsw/pkgutils15`. Special considerations must be taken with the database path for RPM packages (cf. section 4.1.3). If a package is installed and removed without the `pkg-rm` command, `pkg-inst` will fail with the following warning (this warning is ignored if `--reinstall` is given):

```
installing [package] ...
  A pkg-db entry for this package already exists. This should not
  occur. Maybe the pkg-db entry is corrupt? Please check
  /var/opt/TWWfsw/pkgutils15/jpeg-6b.
```

To create files in the new package database for packages installed with a previous release, use the `pkg-info` command with the '`--update-pkg-db`' option.

```
$ ls -ld /var/opt/TWWfsw/pkgutils15/python-2.1.2
/var/opt/TWWfsw/pkgutils15/python-2.1.2: No such file or directory
$ pkg-info --update-pkg-db --verbose
...
python
  checking if v2.1.2 installed (TWW.python212) ... yes
    installed instances ... TWW.python212.man TWW.python212.doc
    TWW.python212.rte TWW.python212.pkg
    updating package database entry for man fileset ... done
    updating package database entry for doc fileset ... done
    updating package database entry for runtime fileset ... done
    updating package database entry for packages fileset ... done
...
$ ls -ld /var/opt/TWWfsw/pkgutils15/python-2.1.2
-rw-r--r--   1 root     system       839 May 17 00:23 python-2.1.2
```

4.1.2 Extracting native package from .pkg-inst archive

The "`--extract=`*`path`*" option is available to copy the raw package out of the `.pkg-inst` archive to a local directory. The `pkg-inst(1)` man page documents this option and how the package is copied to the destination directory.

4.1.3 Installing RPM packages

`pkg-inst` needs no assistance when installing packages for the native package management systems. However, if installing RPM packages for multiple platforms on the same host, the default database path, '`/var/opt/TWWfsw/rpm40/lib/rpm`' will not be sufficient. A separate database directory must exist for each platform. The following options are of particular importance:

```
--rpm-arch=<arch>         RPM architecture type. Possible types are:
                            alpha (Tru64 UNIX 4.x, 5.x)
                            i386 (Redhat Linux)
                            mipseb (IRIX 6.x)
                            parisc (HP-UX 10.20, 11.x)
                            ppc (AIX 4.3.2)
                            sparc (Solaris 2.x)
--rpm-db=<dir>            RPM database path
```

If installing for a non-native architecture, the '`--rpm-arch=`*`arch`*' option specifies which platform to install for. This is important because, by default, `pkg-inst` will select a default architecture name (from the list above) based on the host platform `pkg-inst` is running on. Because the RPM package name contains the architecture type, `pkg-inst` must know the correct architecture type to select the correct RPM file. The examples below demonstrate the `pkg-inst` equivalent commands for each of the RPM commands. Some of the examples are given in section 10.2.

```
$ rpm -Uv --dbpath /var/rpm/sparc-sun-solaris2.5.1 --relocate \
/opt/TWWfsw=/nfs/sparc-sun-solaris2.5.1 --ignorearch --ignoreos \
/tmp/solaris2.5.1-rpms/TWWtin-1.4.2-1.sparc.rpm
$ pkg-inst --rpm-db=/var/rpm/sparc-sun-solaris2.5.1 -e --relocate
-e /opt/TWWfsw=/nfs/sparc-sun-solaris2.5.1 -e --ignorearch -e --ignoreos
--depot=/tmp/solaris2.5.1-rpms tin

$ rpm -Uv --dbpath /var/rpm/sparc-sun-solaris2.6 --relocate \
/opt/TWWfsw=/nfs/sparc-sun-solaris2.6 --ignorearch --ignoreos \
/tmp/solaris2.6-rpms/TWWtin-1.4.2-1.sparc.rpm
$ pkg-inst --rpm-db=/var/rpm/sparc-sun-solaris2.6 -e --relocate
-e /opt/TWWfsw=/nfs/sparc-sun-solaris2.6 -e --ignorearch -e --ignoreos
--depot=/tmp/solaris2.6-rpms tin

$ rpm -Uv /tmp/solaris2.8-rpms/TWWtin-1.4.2-1.sparc.rpm
$ pkg-inst --depot=/tmp/solaris2.8-rpms tin
```

4.1.4 Installing packages from a remote repository

In addition to installing packages from a local repository, a remote repository, available through the HTTP protocol, can be used. A package format has been created for use with `pkg-inst`. Filenames ending with '`.pkg-inst`' contain the binaries in Zip archive format and filenames ending with '`.pkg-inst.md5`' contain the MD5 checksum of the '`.pkg-inst`' archive. The following options to `pkg-inst` are of importance:

```
--dist=<dist>             Name of distribution to update. Default
                          value is "cd" for the normal distribution.
                          Used mainly when installing from a
                          remote depot.
--local-depot=<depot>     Path to local repository to store
                          files retrieved from remote repositories
```

```
--login=<login>             Login for remote depot
--password=<pass>           Password for remote depot
--proxy-host=<host:port>    Hostname and proxy of proxy server
--proxy-login=<login>       Login for proxy server
--proxy-password=<pass>     Password to proxy server
--update-type=<type>        Limit updates to <type>. Possible types are:
                              errata packages security support updates
                            Default type is "packages". Used mainly
                            when installing from a remote depot.
```

To install remote packages, a local repository must be created to save the downloaded packages. This directory is specified with the '`--local-depot=`*`depot`*' option. `pkg-inst` will create a temporary working directory to extract the '`.pkg-inst`' files when working (usually in '`/var/tmp`').

Installation from a remote depot is similar to installation from a local depot. As with the local depot, `--depot=`*`depot`* specifies the location of the depot. If the target is behind a firewall, the `--proxy-host=`*`host:port`*, `--proxy-login=`*`login`*, and `--proxy-password=`*`password`* options can be used to penetrate the firewall. If a login and password are required for authentication to the depot, the `--login=`*`login`* and `--password=`*`password`* options must be used. HTTP BASIC authentication is used for authentication to remote depots. The `--update-type=`*`type`* option specifies which directory in the depot to install packages from. If not specified and the base of the remote depot contains a '`depot-db.xml`' file, the directories specified in the '`depot-db.xml`' file are searched for packages. The '`--verbose`' option to `pkg-inst` provides information about what directories are being searched. All packages from our quarterly distribution are available in the '`packages`' directory ('`--update-type=packages`'). Updates made available between releases will be available from the '`errata`', '`security`', and '`updates`' directories.

As `pkg-inst` installs packages from a remote depot, it populates the local depot (specified with '`--local-depot=`*`depot`*') with the '`.pkg-inst`' package files. These files are never removed by '`pkg-inst`'. If the packages in the remote depot are updated after the copies are made to the local depot, `pkg-inst` will check the remote depot for updates and download newer versions of the packages. If future installs or reinstallations are necessary, the local depot can be specified as the depot for the '`--depot=`*`depot`*' option to eliminate the download time. It is also possible to mirror the remote repository to local disk and specify the local mirror directory as the depot to `pkg-inst`.

Once the '`.pkg-inst`' file is downloaded, `pkg-inst` will verify the file against its MD5 signature, unpack the archive and the archive's contents, and proceed with installation. If the MD5 signature check fails, `pkg-inst` will re-download the package and proceed to unpack the archive for installation.

The following is a sample run under HP-UX installing MGv, the postscript previewer:

```
$ pkg-inst --dist-ver=5.1 --local-depot=/tmp/pkg --login=[login]
--password=[password] --depot=http://updates.thewrittenword.com mgv
mgv
  depot containing v3.1.5-2 of this package ...
    http://updates.thewrittenword.com
  checking if already installed ... no
    components to install: TWWmgv.TWWmgv-MAN, TWWmgv.TWWmgv-RUN
  checking for dependencies ... yes
    TWWxpm.TWWxpm-RUN (v>=3.4k, r>=1) ... to be installed
  reordering packages ...
    adding TWWxpm.TWWxpm-RUN [xpm] (v>=3.4k, r>=1) to package list
```

```
      found v3.4k-3 in depot http://updates.thewrittenword.com
    will reinstall after dependencies installed

  xpm
    depot containing v3.4k-3 of this package ...
      http://updates.thewrittenword.com
    checking if already installed ... no
      components to install: TWWxpm.TWWxpm-RUN
    checking for dependencies ... none
    checking if package exists in local depot ... no
    retrieving xpm-3.4k-3 package from remote depot ... 100%
    verifying MD5 checksum of retrieved package payload ... ok
    extracting package files ...
      extracting data.zip file ... done
      testing data.zip ... done
      unpacking data.zip contents ... done
    installing TWWxpm.TWWxpm-RUN ...
      $ swinstall -s /var/tmp/AAAa26293/TWWxpm.depot TWWxpm.TWWxpm-RUN
      ...
      updating package database entry for runtime fileset ... done
    removing temporary files ... done

  mgv
    depot containing v3.1.5-2 of this package ...
      http://updates.thewrittenword.com
    checking if already installed ... no
      components to install: TWWmgv.TWWmgv-MAN, TWWmgv.TWWmgv-RUN
    checking for dependencies ... yes
      TWWxpm.TWWxpm-RUN (v>=3.4k, r>=1) ... to be installed
    reordering packages ...
      skipping TWWxpm.TWWxpm-RUN as installation already attempted
    checking if package exists in local depot ... no
    retrieving mgv-3.1.5-2 package from remote depot ... 100%
    verifying MD5 checksum of retrieved package payload ... ok
    extracting package files ...
      extracting data.zip file ... done
      testing data.zip ... done
      unpacking data.zip contents ... done
    installing TWWmgv.TWWmgv-MAN ...
      $ swinstall -s /var/tmp/BAAa26293/TWWmgv.depot TWWmgv.TWWmgv-MAN
      ...
      updating package database entry for man fileset ... done
    installing TWWmgv.TWWmgv-RUN ...
      $ swinstall -s /var/tmp/BAAa26293/TWWmgv.depot TWWmgv.TWWmgv-RUN
      ...
      updating package database entry for runtime fileset ... done
    removing temporary files ... done
```

4.1.5 Checking GPG signature of packages

Packages are signed with our GPG key. The '`--check-sig`' option checks the signature of the package against our GPG public key in your keyring (specified with '`--gpg-keyring-path=`*`path`*'). Before running `pkg-inst` with '`--check-sig`', the 'gpg' binary must be available in your search path (`$PATH`). The 'gpg' binary is available from the GnuPG package. Our GPG public key can be found at `https://support.thewrittenword.com/` and in the '`support`' directory of our distribution CDs under the name ‘`public-key.gpg`’. The following demonstrates how to import our public key into a keyring:

```
$ gpg --no-default-keyring --homedir <keyring path> --import tww-public-key.gpg
gpg: key FE93BD4E:public key imported
gpg: Total number processed: 1
gpg:               imported: 1
$ gpg --no-default-keyring --homedir <keyring path> --list-keys
<keyring path>/pubring.gpg
------------------------------
pub  1024D/FE93BD4E 2001-04-08 The Written Word, Inc. <security@thewrittenword.com>
sub  2048g/87A9A2F0 2001-04-08
```

The following is a sample run under HP-UX installing Xpm with the `--check-sig` option to check its GPG signature.

```
$ pkg-inst --check-sig --gpg-keyring-path=/root/tww-gpg --dist-ver=5.1
--local-depot=/tmp/pkg --login=[login] --password=[password]
--depot=http://updates.thewrittenword.com xpm
xpm
  depot containing v3.4k-3 of this package ...
    http://updates.thewrittenword.com
  checking if already installed ... no
    components to install: TWWxpm.TWWxpm-MAN, TWWxpm.TWWxpm-RUN
  checking for dependencies ... none
  checking if package exists in local depot ... no
  retrieving xpm-3.4k-3 package from remote depot ... 100%
  verifying MD5 checksum of retrieved package payload ... ok
  verifying GPG signature of retrieved package payload ... ok
  extracting package files ...
    extracting data.zip file ... done
    testing data.zip ... done
    unpacking data.zip contents ... done
  installing TWWxpm.TWWxpm-MAN ...
    $ swinstall -s /var/tmp/AAAa26284/TWWxpm.depot TWWxpm.TWWxpm-MAN
    ...
    updating package database entry for man fileset ... done
  installing TWWxpm.TWWxpm-RUN ...
    $ swinstall -s /var/tmp/AAAa26284/TWWxpm.depot TWWxpm.TWWxpm-RUN
    ...
    updating package database entry for runtime fileset ... done
  removing temporary files ... done
```

4.1.6 Configuration file

A configuration file can be created to help reduce the number of options on the command-line. The location of the configuration file is ‘`/opt/TWWfsw/pkgutils15/etc/pkgutils.conf`’. The default version installed with pkgutils contains verbose descriptions of all options. The configuration file is also documented in `pkgutils.conf(4)`.

A sample configuration file to retrieve packages from the latest distribution would look like:

```
local-depot = "/tmp/pkg"

depots = [ http://updates.thewrittenword.com ]
```

```
pkg-db = "/var/opt/TWWfsw/pkgutils15"

depot http://updates.thewrittenword.com {
  dist-ver = [ latest ]

  login = <login>
  password = <password>
}
```

Without the configuration file above, the command-line equivalent for `pkg-inst` would be:

```
$ pkg-inst --local-depot=/tmp/pkg --dist-ver=latest --login=<login>
--password=<password> --depot=http://updates.thewrittenword.com
```

With the configuration file, the above would be reduced to executing:

```
$ pkg-inst
```

4.1.7 Inside `pkg-inst`

Local and remote repositories

When specifying a repository with the '`--depot=`*`depot`*' option, the distribution name, specified with the '`--dist=`*`dist`*' option, must exist as a subdirectory of the repository path. Thus, if `pkg-inst` is invoked as:

```
$ pkg-inst --dist-ver=5.1 --depot=http://updates.thewrittenword.com ...
```

the root path of the repository is '`http://updates.thewrittenword.com/cd`' where a '`depot-db.xml`' or '`pkg-db.xml`' file is expected to reside. If `pkg-inst` is invoked as:

```
$ pkg-inst --dist-ver=5.1 --dist=local \
--depot=http://updates.thewrittenword.com ...
```

the root path of the repository is '`http://updates.thewrittenword.com/local`'.

In the repository root, `pkg-inst` searches for either the '`depot-db.xml`' or '`pkg-db.xml`' file. Unlike a remote repository, raw packages can reside in the same directory as the '`pkg-db.xml`' file while with remote repositories, '`.pkg-inst`' package files must reside in the same directory as their corresponding '`pkg-db.xml`' file.

The `depot-db.xml` file

As indicated previously, `pkg-inst` can recursively descend into a repository to search for packages. This is done using the '`depot-db.xml`' file. `pkg-inst` expects to find either a '`depot-db.xml`' or '`pkg-db.xml`' at the root of a repository and at each directory it descends into. The '`depot-db.xml`' file contains a list of directories to recurse into. Therefore, `pkg-inst` will recurse into the depot to the directory level specified by '`depot-db.xml`'. The DTD of the '`depot-db.xml`' file is documented in `depot-db.xml(4)` in the `pkgutils` package.

The `pkg-db.xml` file

The `pkg-inst` tool is simply a front-end to the native package management systems. It contains pluggable backend modules for each supported package manager to handle calls to the native package management system. To unify the differences between the different package management systems and provide a single interface to the backend modules, the '`pkg-db.xml`' file was created which provides all the information `pkg-inst` needs to install and remove a package. The '`pkg-db.xml`' file is at the heart of all utilities in the `pkgutils` suite. The DTD of the '`pkg-db.xml`' file is documented in `pkg-db.xml(4)`.

4.2 Removing packages (`pkg-rm`)

The `pkg-rm` tool is used to remove packages installed by `pkg-inst`. The `pkg-rm` tool makes use of the package database specified with `--pkg-db=`*`path`* and the global package database, '`/opt/TWWfsw/pkgutils15/share/pkg-db.xml`', containing a list of all pkg-inst packages every created. Only packages in the package database directory or global package database can be removed. The global package database was introduced in the 1.3 version and is being deprecated by the package database directory.

Package dependents (reverse dependencies) can be removed when a package is removed using the '`--add-reverse-deps`' option. The `--ignoredeps` option allows `pkg-rm` to remove a package even though the package being removed is a dependency for another package. By default, `pkg-rm` will not remove a package unless all dependents are listed or if the `--add-reverse-deps` option is given. Like the `pkg-inst` tool, `pkg-rm`, when given the `--add-reverse-deps` option, will remove only those components of the dependent package necessary to meet the reverse dependency. An example of this is show below when `xpm` is removed and only the runtime component of its dependent package, `mgv`, is removed, leaving the man component installed.

The package name given as input to `pkg-rm` is the same as that used by `pkg-inst`. A sample run removing the `gzip` package follows:

```
$ pkg-rm gzip
gzip
  checking if v1.3.5-1 installed (TWWgzp13) ... yes
    installed instances ... TWWgzp13m TWWgzp13u TWWgzp13d
  checking for packages dependent on me ... none
  checking for dependencies ... none
  removing TWWgzp13d ...
    $ pkgrm -a /opt/TWWfsw/pkgutils15/share/pkgadd/admin -n TWWgzp13d
    ...
    removing package database entry for doc fileset ... done
  removing TWWgzp13u ...
    $ pkgrm -a /opt/TWWfsw/pkgutils15/share/pkgadd/admin -n TWWgzp13u
    ...
    removing package database entry for runtime fileset ... done
  removing TWWgzp13m ...
    $ pkgrm -a /opt/TWWfsw/pkgutils15/share/pkgadd/admin -n TWWgzp13m
    ...
    removing package database entry for man fileset ... done
```

As an example of the `--add-reverse-deps` option, the following example removes the `xpm` package which automatically removes the `mgv` package (dependent on the `xpm` package):

```
$ pkg-rm xpm
xpm
  checking if v3.4k-3 installed (TWWxpm) ... yes
    installed instances ... TWWxpm.TWWxpm-RUN
  checking for packages dependent on me ... yes
    TWWmgv [mgv] ... installed (v3.1.5-2)
  checking package list ...
    mgv v3.1.5 not found in package list
  checking for dependencies ... none
  skipping due to failed reverse dependencies

$ pkg-rm --add-reverse-deps xpm
xpm
  checking if v3.4k-3 installed (TWWxpm) ... yes
    installed instances ... TWWxpm.TWWxpm-RUN
  checking for packages dependent on me ... yes
    TWWmgv [mgv] ... installed (v3.1.5-2)
  reordering packages ...
    adding TWWmgv [mgv] (v3.1.5) to package list
  checking for dependencies ... none
  will remove after dependent packages removed

mgv
  checking if v3.1.5-2 installed (TWWmgv) ... yes
    installed instances ... TWWmgv.TWWmgv-RUN
  checking for packages dependent on me ... none
  checking for dependencies ... yes
    TWWxpm.TWWxpm-RUN (v>=3.4k, r>=1) ... installed (v3.4k-3)
  removing TWWmgv.TWWmgv-RUN ...
    $ swremove TWWmgv.TWWmgv-RUN
    ...
    removing package database entry for runtime fileset ... done

xpm
  checking if v3.4k-3 installed (TWWxpm) ... yes
    installed instances ... TWWxpm.TWWxpm-RUN
  checking for packages dependent on me ... none
  checking for dependencies ... none
  removing TWWxpm.TWWxpm-RUN ...
    $ swremove TWWxpm.TWWxpm-RUN
    ...
    removing package database entry for runtime fileset ... done
```

4.3 Package Information (`pkg-info`)

The `pkg-info` tool is used to display information about a package. In addition to displaying the installation path component of a file (used in section 3.2) it can be used to display a list of installed packages, what subpackages are installed, if common links have been created for the package, what the common links are, and populate the package installation database. As with the `pkg-inst` tool, `pkg-info` is independent of the underlying package managment system.

The `--link-listing` option displays the subpackage components that have common links associated with them. A sample run of `pkg-info` with `--link-listing=short` on Solaris follows:

```
$ pkg-info --link-listing=short python
```

```
python
  checking if v2.1.2-12 installed (TWW.python212) ... yes
    installed instances ... TWW.python212.man TWW.python212.pkg
    TWW.python212.doc TWW.python212.rte
    searching for common links ... found
  common links ...
    bin: TWW.python212.pkg TWW.python212.rte
    man: TWW.python212.man
```

A sample run with `--link-listing=long` to display the common links created follows:

```
$ pkg-info --link-listing=long python
python
  checking if v2.1.2-12 installed (TWW.python212) ... yes
    installed instances ... TWW.python212.man TWW.python212.pkg
    TWW.python212.doc TWW.python212.rte
    searching for common links ... found
    links/wrappers in common directories ...
      l /opt/TWWfsw/man/man1/python.1 -> /opt/TWWfsw/python212/man/man1/python.1
      l /opt/TWWfsw/bin/gfplus -> /opt/TWWfsw/python212p/bin/gfplus
      l /opt/TWWfsw/bin/gfserver -> /opt/TWWfsw/python212p/bin/gfserver
      l /opt/TWWfsw/bin/xmlproc_parse -> /opt/TWWfsw/python212p/bin/xmlproc_parse
      l /opt/TWWfsw/bin/xmlproc_val -> /opt/TWWfsw/python212p/bin/xmlproc_val
      W /opt/TWWfsw/bin/pydoc
      W /opt/TWWfsw/bin/python
      W /opt/TWWfsw/bin/python2.1
```

If a description entry exists for a package, a description can be printed using the `--print=description` option:

```
$ pkg-info --print=description python
python
  checking if v2.1.2-12 installed (TWW.python212) ... yes
    installed instances ... TWW.python212.man TWW.python212.pkg
    TWW.python212.doc TWW.python212.rte
    description ...
      Python is an interpreted, interactive, object-oriented
      programming language. Python combines remarkable power with very
      clear syntax. It has modules, classes, exceptions, very high
      level dynamic data types, and dynamic typing. There are
      interfaces to many system calls and libraries, as well as to
      various windowing systems (X11, Motif, Tk, Mac, MFC). New
      built-in modules are easily written in C or C++. Python is also
      usable as an extension language for applications that need a
      programmable interface. It is also usable as an extension
      language for applications that need a programmable interface.
```

A repository can also be queried for information about a package:

```
$ pkg-info --depot=http://updates.thewrittenword.com --dist-ver=5.1
--login=<login> --password=<password> --print=description python-2.2.2
python
  depot containing v2.2.2-6 of this package ...
    http://updates.il.thewrittenword.com
  installation path component for v2.2.2 ... python222
  instances ... TWW.python222.man TWW.python222.pkg TWW.python222.doc
  TWW.python222.rte
  dependencies ...
    TWW.openssl097.rte [openssl] (v>=0.9.7.3)
  description ...
    Python is an interpreted, interactive, object-oriented programming
    language. Python combines remarkable power with very clear syntax.
    It has modules, classes, exceptions, very high level dynamic data
    types, and dynamic typing. There are interfaces to many system
    calls and libraries, as well as to various windowing systems (X11,
    Motif, Tk, Mac, MFC). New built-in modules are easily written in C
    or C++. Python is also usable as an extension language for
    applications that need a programmable interface. It is also usable
    as an extension language for applications that need a programmable
    interface.
```

While each package management system has a utility to display a list of installed packages, the `pkg-info` tool was designed primarily to provide a list of the installed packages independent of those already installed on the operating system.

The package installation database (cf. section 4.1.1) is used to obtain a list of installed packages. The old pre-1.4 global package database, '`/opt/TWWfsw/pkgutils15/share/pkg-db.xml`', is used as a fallback if the package is not found in the installation database. All packages in the installation database and global package database are queried to determine which packages are installed. In time, the global package database will be deprecated and the package installation database will become the sole source of installed packages. To populate the package installation database from an existing installation, use the `--update-pkg-db` option as follows:

```
  $ ls -ld /var/opt/TWWfsw/pkgutils15/libpng-1.0.14
  /var/opt/TWWfsw/pkgutils15/libpng-1.0.14: No such file or directory
  $ pkg-info --update-pkg-db --verbose
  ...
  libpng
    checking if v1.0.14 installed (TWWpng10) ... yes
      installed instances ... TWWpng10u TWWpng10m
      updating package database entry for runtime fileset ... done
      updating package database entry for man fileset ... done
  ...
  $ ls -ld /var/opt/TWWfsw/pkgutils15/libpng-1.0.14
  -rw-r--r--   1 root     root         510 Jul 16 18:27 libpng-1.0.14
```

4.4 Package Configuration (`pkg-config`)

The `pkg-config` tool is used to execute the post-install and pre-remove procedure script for a package. Usually, the post-install script will create the set of common links in the bin, info, and man directories. The pre-remove script will remove these links. This command is useful if more than one version of a package is installed and the primary version (the package with links in the common directory) needs to change. It is also useful if a package was initially installed with `COMMON_BASE=none` in the configuration file (cf. section 3.2.1).

The default action of `pkg-config` is to execute the post-install script. The `--uninstall` option executes the pre-remove script. Note the list of possibilities for the `--subpkgs` option is more restrictive than with the other commands in the `pkgutils` suite. Because the primary purpose is to create/remove the common links and because common links are only created for the "`libruntime`", "`man`", "`packages`", and "`runtime`" subpackages, the list is more restrictive.

A sample run of `pkg-config` to remove and then recreate the common links on Solaris follows:

```
$ pkg-config --uninstall python-2.1.2
python
  checking if v2.1.2-12 installed (TWW.python212) ... yes
    installed instances ... TWW.python212.man TWW.python212.pkg
    TWW.python212.doc TWW.python212.rte
    searching for common links ... found
    removing links in man for TWW.python212.man by running:
      $ sh /opt/TWWfsw/python212/tww-inst/links-rm.man
      ...
    removing links in bin for TWW.python212.pkg by running:
      $ sh /opt/TWWfsw/python212p/tww-inst/links-rm.pkg
      ...
    removing links in bin for TWW.python212.rte by running:
      $ sh /opt/TWWfsw/python212/tww-inst/links-rm.rte
      ...

$ pkg-info --verbose --link-listing=short python
python
  checking if v2.1.2-12 installed (TWW.python212) ... yes
    installed instances ... TWW.python212.man TWW.python212.pkg
    TWW.python212.doc TWW.python212.rte
    searching for common links ... none found

$ pkg-config python-2.1.2
python
  checking if v2.1.2-12 installed (TWW.python212) ... yes
    installed instances ... TWW.python212.man TWW.python212.pkg
    TWW.python212.doc TWW.python212.rte
    installing links in man for TWW.python212.man by running:
      $ sh /opt/TWWfsw/python212/tww-inst/links-inst.man
      ...
    installing links in bin for TWW.python212.pkg by running:
      $ sh /opt/TWWfsw/python212p/tww-inst/links-inst.pkg
      ...
    installing links in bin for TWW.python212.rte by running:
      $ sh /opt/TWWfsw/python212/tww-inst/links-inst.rte
      ...

$ pkg-info --link-listing=short python
python
  checking if v2.1.2-12 installed (TWW.python212) ... yes
    installed instances ... TWW.python212.man TWW.python212.pkg
    TWW.python212.doc TWW.python212.rte
    searching for common links ... found
  common links ...
    bin: TWW.python212.pkg TWW.python212.rte
    man: TWW.python212.man
```

4.5 Checking for updates (`chk-pkg-updates`)

The `chk-pkg-updates` tool compares the list of packages installed against a depot and indicates which packages are up-to-date and which have updates available.

A sample run of `chk-pkg-updates` on Solaris follows:

```
$ chk-pkg-updates --dist-ver=5.1 --login=<login> --password=<password>
--depot=http://updates.thewrittenword.com python-2.1.2
python
  checking if v2.2.2-6 installed ... no
    depot containing this package ...
      http://updates.il.thewrittenword.com
    checking if update available ... yes (packages)
      TWW.python222.man (v2.2.2.6)
      TWW.python222.pkg (v2.2.2.6)
      TWW.python222.doc (v2.2.2.6)
      TWW.python222.rte (v2.2.2.6)
  checking if v2.2.1-8 installed ... no
    depot containing this package ...
      http://updates.il.thewrittenword.com
    checking if update available ... yes (packages)
      TWW.python221.man (v2.2.1.8)
      TWW.python221.pkg (v2.2.1.8)
      TWW.python221.doc (v2.2.1.8)
      TWW.python221.rte (v2.2.1.8)
  checking if v2.1.2-12 installed ... no
    depot containing this package ...
      http://updates.il.thewrittenword.com
    checking if update available ... yes (packages)
      TWW.python212.man (v2.1.2.10 < v2.1.2.12)
      TWW.python212.doc (v2.1.2.10 < v2.1.2.12)
      TWW.python212.rte (v2.1.2.10 < v2.1.2.12)
      TWW.python212.pkg (v2.1.2.10 < v2.1.2.12)
```

Chapter 5

Solaris 2.x (pkgadd)

NOTE: We strongly recommend using the **pkg-inst** tool, documented in chapter 4, to install packages.

Solaris packages are provided in pkgadd format. The `pkgadd` command, found in '`/usr/sbin`', is used to install packages. All packages are built and stored in directories rather than individual files because installation of multiple packages is easier. The last character of the package name denotes its content:

```
c     Local configuration files
d     Additional support documentation
m     Info and manual pages
p     3rd-party packages
r     Library runtime
u     Executables, includes, and libraries
```

Most packages contain the '`m`' and '`u`' extensions. Packages such as Perl, XEmacs, and teTeX contain the '`p`' extension which installs into a different parent directory than the main program (e.g. installation into '`/opt/TWWfsw/perl5005p`' rather than '`/opt/TWWfsw/perl5005`' and '`/opt/TWWfsw/xemacs21p`' rather than '`/opt/TWWfsw/xemacs21`'). The 3rd-party additions are installed independent of the program to make upgrades to 3rd-party modules easier and for them not to affect the main program. Packages with documentation not in GNU info or man format will have a separate package for this documentation installed into either '`/opt/TWWfsw/`*`package`*`/docs`' or '`/opt/TWWfsw/`*`package`*`/doc`' and available in the '`d`' extension. Packages with the '`c`' extension contain files to be installed on the local filesystem. Examples of packages with this extension include Apache, with configuration files in '`/etc/opt/TWWfsw/apache139`' and log files in '`/var/opt/TWWfsw/apache139`', Samba, with configuration files in '`/etc/opt/TWWfsw/samba206`' and log files in '`/var/opt/TWWfsw/samba206`', and XFce, with CDE configuration files in '`/etc/dt`'. The library runtime package, packages ending with '`r`', contain shared libraries for dependency purposes. With these packages, the full runtime package does not need to be installed to meet a dependency. GCC is an example of one such package with the library runtime component.

Chapter 6

IRIX 6.x (inst)

NOTE: We strongly recommend using the **pkg-inst** tool, documented in chapter 4, to install packages.

IRIX packages are provided in inst format. The inst command, found in '`/usr/sbin`', is used to install packages. Each package is comprised of at most four files, '*package*', '*package*`.idb`', '*package*`.sw`', and sometimes '*package*`.man`'. Packages are built with the following subsets:

`sw.base`	`Base software`
`sw.config`	`Local configuration files`
`sw.librun`	`Library runtime`
`sw.packages`	`3rd-party Packages`
`man.doc`	`Extra online documentation`
`man.info`	`GNU info pages`
`man.pages`	`Manual pages`

Most packages contain the subsets '`sw.base`', '`man.info`', and '`man.pages`'. Packages such as Perl, XEmacs, and teTeX contain the '`sw.packages`' subset which installs into a different parent directory than the main program (e.g. installation into '`/opt/TWWfsw/perl5005p`' rather than '`/opt/TWWfsw/perl5005`' and '`/opt/TWWfsw/xemacs21p`' rather than '`/opt/TWWfsw/xemacs21`'). The 3rd-party additions are installed independent of the program to make upgrades to 3rd-party modules easier and for them not to affect the main program. Packages with documentation not in GNU info or man format will have a separate package for this documentation installed into either '`/opt/TWWfsw/`*`package`*`/docs`' or '`/opt/TWWfsw/`*`package`*`/doc`' and available in the '`man.doc`' subset. Packages with the '`sw.config`' subset contain files to be installed on the local filesystem. Examples of packages with this extension include Apache, with configuration files in '`/etc/opt/TWWfsw/apache139`' and log files in '`/var/opt/TWWfsw/apache139`', Samba, with configuration files in '`/etc/opt/TWWfsw/samba206`' and log files in '`/var/opt/TWWfsw/samba206`', and XFce, with CDE configuration files in '`/etc/dt`'. The library runtime package, packages ending with '`.sw.librun`', contain shared libraries for dependency purposes. With these packages, the full runtime package does not need to be installed to meet a dependency. GCC is an example of one such package with the library runtime component.

6.1 Procedure scripts

If the installation configuration file indicates that symbolic links should be created after the installation of a package, the package postinstall script will create symbolic links to the common bin, info, and man directories and add entries to the '`dir`' file for all GNU info files. Once the package is removed, a script is run to remove the entries from the '`dir`' file for all GNU info files and to remove the symbolic links from the common bin, info, and man directories.

Because the IRIX package management system does not provide commands similar to the Solaris '`installf/removef`' or the HP-UX '`swmodify`' commands, some hacks were required to make the above work. When the package '`bin`', '`info`', or '`man`' directories are installed, a script is run to create symbolic links in a common tree. This is done by specifying an '`exitop`' command as part of the package to run '`/opt/TWWfsw/`*`package`*`/tww-inst/make-links`'. The argument to '`exitop`' is a sequence of shell commands. Because the size of the command is limiting, a separate script was created. If '`versions long` *`package`*' is executed to view the list of files that comprise the package, this file, and others under '`/opt/TWWfsw/`*`package`*`/tww-inst`', will not be listed. These files are not part of the package file listing because they have the '`nohist`' attribute set in the package IDB file.

As the symbolic links and wrapper scripts are created, information about the link is saved to '`/opt/TWWfsw/`*`package`*`/tww-inst/links`'. This file stores the type of link, the directory the link was created for, the source of the link, and the destination. Unlike other package management systems, it is not possible to register a file as part of a package that is not part of the IDB file. Thus, if the link information was not recorded, removal of the package would not remove the links. If a wrapper script is created, the entry for the wrapper in the file contains the pathname of the wrapper and its checksum (calculated with `sum(1))`. If a package is updated with a newer instance, this file is read and verified against what is present in the target installation tree. If any of the symbolic links are invalid or the wrapper script has changed (because the checksum has changed), the entry in the '`links`' file will be updated to reflect the current state.

Once the package is removed, a script is run to remove the symbolic links, remove wrapper scripts, and, if necessary, remove any entries from the common '`dir`' file for any GNU info files. This is handled by the script '`/opt/TWWfsw/`*`package`*`/tww-inst/rm-links`'. Because a record of the existing links has been created, this script reads the list and removes entries as appropriate. Once complete, it removes the '`/opt/TWWfsw/`*`package`*`/tww-inst`' directory, because it is not part of the package database.

Chapter 7

HP-UX 10.20, 11.x (depot)

NOTE: We strongly recommend using the **pkg-inst** tool, documented in chapter 4, to install packages.

HP-UX packages are provided in depot format. The `swinstall` command, found in '`/usr/sbin`', is used to install packages. Packages are categorized as follows:

```
CONF    Local configuration files
HELP    Additional support documentation
LIBR    Library runtime
MAN     Info and manual pages
PKG     3rd-party packages
RUN     Executables, includes, and libraries
```

Most packages contain the '`RUN`' and '`MAN`' subproducts. Packages such as Perl, XEmacs, and teTeX contain the '`PKG`' extension which installs into a different parent directory than the main program (e.g. installation into '`/opt/TWWfsw/perl5005p`' rather than '`/opt/TWWfsw/perl5005`' and '`/opt/TWWfsw/xemacs21p`' rather than '`/opt/TWWfsw/xemacs21`'). The 3rd-party additions are installed independent of the program to make upgrades to 3rd-party modules easier and for them not to affect the main program. Packages with documentation not in GNU info or man format will have a separate package for this documentation installed into either '`/opt/TWWfsw/`*`package`*`/docs`' or '`/opt/TWWfsw/`*`package`*`/doc`' and available in the '`HELP`' subproduct. Files in the '`CONF`' subproduct are installed on the local filesystem. Examples of packages with this extension include Apache, with configuration files in '`/etc/opt/TWWfsw/apache139`' and log files in '`/var/opt/TWWfsw/apache139`', Samba, with configuration files in '`/etc/opt/TWWfsw/samba206`' and log files in '`/var/opt/TWWfsw/samba206`', and XFce, with CDE configuration files in '`/etc/dt`'. The library runtime package, packages ending with '`LIBR`', contain shared libraries for dependency purposes. With these packages, the full runtime package does not need to be installed to meet a dependency. GCC is an example of one such package with the library runtime component.

Chapter 8

Compaq Tru64 UNIX 4.0D, 5.1 (setld)

NOTE: We strongly recommend using the **pkg-inst** tool, documented in chapter 4, to install packages.

Tru64 UNIX packages are provided in setld format. The `setld` command, found in '`/usr/sbin`', is used to install packages. Packages are categorized as follows:

```
BIN     Executables, includes, and libraries
CONF    Local configuration files
DOC     Additional support documentation
LIBR    Library runtime
MAN     Info and manual pages
PKG     3rd-party packages
```

Most packages contain the '`BIN`' and '`MAN`' subproducts. Packages such as Perl, XEmacs, and teTeX contain the '`PKG`' extension which installs into a different parent directory than the main program (e.g. installation into '`/opt/TWWfsw/perl5005p`' rather than '`/opt/TWWfsw/perl5005`' and '`/opt/TWWfsw/xemacs21p`' rather than '`/opt/TWWfsw/xemacs21`'). The 3rd-party additions are installed independent of the program to make upgrades to 3rd-party modules easier and for them not to affect the main program. Packages with documentation not in GNU info or man format will have a separate package for this documentation installed into either '/opt/TWWfsw/*package*/docs' or '/opt/TWWfsw/*package*/doc' and available in the '`DOC`' subproduct. Files in the '`CONF`' subproduct are installed on the local filesystem. Examples of packages with this extension include Apache, with configuration files in '`/etc/opt/TWWfsw/apache139`' and log files in '`/var/opt/TWWfsw/apache139`', and Samba, with configuration files in '`/etc/opt/TWWfsw/samba206`' and log files in '`/var/opt/TWWfsw/samba206`'. The library runtime package, packages ending with '`LIBR`', contain shared libraries for dependency purposes. With these packages, the full runtime package does not need to be installed to meet a dependency. GCC is an example of one such package with the library runtime component.

8.1 Patch requirement for Tru64 UNIX 5.1

Tru64 UNIX 5.1 requires the installation of patches #260 and #262 from patch kit #3 to provide an important feature enhancement to the Tru64 UNIX loader, `/sbin/loader`. Versions of Tru64 up to this patch will not load dependent libraries when dlopen()'ing shared libraries. This is an inconvenience for 3rd-party modules loaded dynamically by Apache, Perl, Python, etc. Our workaround prior to this patch was to build the 3rd-party modules statically to reduce the need for setting the `LD_LIBRARY_PATH` environment variable. With patch kit #3 for Tru64 UNIX 5.1, this limitation is removed.

Chapter 9

AIX 4.3.2 (lpp)

NOTE: We strongly recommend using the **pkg-inst** tool, documented in chapter 4, to install packages.

AIX packages are provided in AIX backup (`bff`) format. The `installp` command, found in '`/usr/sbin`', is used to install and remove packages. Packages are categorized as follows:

`.rte`	`Executables, includes, and libraries`
`.conf`	`Local configuration files`
`.doc`	`Additional support documentation`
`.librun`	`Library runtime`
`.man`	`Info and manual pages`
`.pkg`	`3rd-party packages`

According to the IBM "Packaging Software for Installation" documentation, the "usr" portion of a fileset contains files that can be shared among several machines while the "root" portion of a fileset contains files that cannot be shared among machines. This usually implies that files installed to `/usr` are part of the "usr" portion and files installed outside of `/usr` are part of the "root" portion. Because packages install to `/opt/TWWfsw`, filesets contain either the "usr" portion only or both the "usr" and "root" portion with files destined for `/opt/TWWfsw/` making up the "usr" portion and files destined for `/etc/opt/TWWfsw` or `/var/opt/TWWfsw` making up the "root" portion.

Most packages contain the '`.rte`' and '`.man`' filesets. Packages such as Perl, XEmacs, and teTeX contain the '`.pkg`' fileset which installs into a different parent directory than the main program (e.g. installation into '`/opt/TWWfsw/perl5005p`' rather than '`/opt/TWWfsw/perl5005`' and '`/opt/TWWfsw/xemacs21p`' rather than '`/opt/TWWfsw/xemacs21`'). The 3rd-party additions are installed independent of the program to make upgrades to 3rd-party modules easier and for them not to affect the main program. Packages with documentation not in GNU info or man format will have a separate package for this documentation installed into either '`/opt/TWWfsw/`*`package`*`/docs`' or '`/opt/TWWfsw/`*`package`*`/doc`' and available in the '`.doc`' fileset. Packages with the '`.conf`' fileset contain files to be installed on the local filesystem. Examples of packages with this extension include

Apache, with configuration files in '`/etc/opt/TWWfsw/apache139`' and log files in '`/var/opt/TWWfsw/apache139`', and Samba, with configuration files in '`/etc/opt/TWWfsw/samba206`' and log files in '`/var/opt/TWWfsw/samba206`'. The library runtime fileset, packages ending with '`.librun`', contain shared libraries for dependency purposes. With these packages, the full runtime package does not need to be installed to meet a dependency. GCC is an example of one such package with the library runtime component.

9.1 Procedure scripts

If the installation configuration file indicates that symbolic links should be created after the installation of a package, the package postinstall script will create symbolic links to the common bin, info, and man directories and add entries to the '`dir`' file for all GNU info files. Once the package is removed, a script is run to remove the entries from the '`dir`' file for all GNU info files and to remove the symbolic links from the common bin, info, and man directories.

Because the AIX package management system does not provide commands similar to the Solaris '`installf/removef`' or the HP-UX '`swmodify`' commands, some hacks were required to make the above work. When the package '`bin`', '`info`', or '`man`' directories are installed, a postinstall script is run to create symbolic links in a common tree.

As the symbolic links and wrapper scripts are created, information about the link is saved to '`/opt/TWWfsw/`*`package`*`/tww-inst/links`'. This file stores the type of link, the directory the link was created for, the source of the link, and the destination. Unlike other package management systems, it is not possible to register a file as part of a package that is not part of the fileset. Thus, if the link information was not recorded, removal of the package would not remove the links. If a wrapper script is created, the entry for the wrapper in the file contains the pathname of the wrapper and its checksum (calculated with `sum(1)`). If a package is updated with a newer instance, this file is read and verified against what is present in the target installation tree. If any of the symbolic links are invalid or the wrapper script has changed (because the checksum has changed), the entry in the '`links`' file will be updated to reflect the current state.

Once the package is removed, a script is run to remove the symbolic links, wrapper scripts, and, if necessary, any entries from the common '`dir`' file for any GNU info files. This is handled by the postremove script, the opposite of the postinstall script. Because a record of the existing links has been created, this script reads the list and removes entries as appropriate. Once complete, it removes the '`/opt/TWWfsw/`*`package`*`/tww-inst`' directory, because it is not part of the package database.

9.2 Handling non-portable files

The AIX package manager restricts files that are part of a fileset to the portable filename character set standard. This "standard" does not include the colon (':') character in pathnames. This is a problem for Perl packages which contain '::' in the pathname for man pages to 3rd-party modules. Our workaround is **not** to save "invalid" files in the LPP package but to embed them in the postinstall script for the package fileset and unpack them at postinstall time. The side effect of this is that the AIX package management system will not be aware of the files. The files are removed at package removal time by the preremove script. The only other solution to this issue was to replace the invalid characters with valid characters. However, this would have affected multi-platform customers so we opted for our workaround.

Chapter 10

RPM (Redhat Linux and others)

> **NOTE:** We strongly recommend using the **pkg-inst** tool, documented in chapter 4, to install packages.

The rpm package manager provides a cross-platform package management tool. RPM packages can be installed on all supported platforms and is the default package management system for Redhat Linux. The design of the postinstall scripts for the RPM packages make it easy to install RPM packages into an NFS-mounted area for remote clients. RPM packages consist of the following subpackages:

```
(no suffix) Executables, includes, and libraries
-conf   Local configuration files
-doc    Additional support documentation
-librun Library runtime
-man    Info and manual pages
-pkg    3rd-party packages
```

Most packages contain the executable package and the '`-man`' subpackage. Packages such as Perl, XEmacs, and teTeX contain the '`-pkg`' subpackage which installs into a different parent directory than the main program (e.g. installation into '`/opt/TWWfsw/perl5005p`' rather than '`/opt/TWWfsw/perl5005`' and '`/opt/TWWfsw/xemacs21p`' rather than '`/opt/TWWfsw/xemacs21`'). The 3rd-party additions are installed independent of the program to make upgrades to 3rd-party modules easier and for them not to affect the main program. Packages with documentation not in GNU info or man format will have a separate package for this documentation installed into either '`/opt/TWWfsw/`*`package`*`/docs`' or '`/opt/TWWfsw/`*`package`*`/doc`' and available in the '`-doc`' subpackage. Files in the '`-conf`' subpackage are installed on the local filesystem. Examples of packages with this extension include Apache, with configuration files in '`/etc/opt/TWWfsw/apache139`' and log files in '`/var/opt/TWWfsw/apache139`', and Samba, with configuration files in '`/etc/opt/TWWfsw/samba206`' and log files in '`/var/opt/TWWfsw/samba206`'. The library runtime package, packages ending with '`-librun`', contain shared libraries for dependency purposes. With these packages, the full runtime package does not need to be installed to meet a dependency. GCC is an example of one such package with the library runtime component.

10.1 Configuring RPM

This section can be skipped if using Redhat Linux where RPM is already configured.

The default location for the RPM database is '/var/opt/TWWfsw/rpm40/lib/rpm'. The location of the database can be changed with the '--dbpath' switch to the rpm command or by editing the value of the '%_dbpath' variable in '/opt/TWWfsw/rpm40/lib/rpm/macros'. Each architecture must have its own database as rpm does not support multiple architectures in one database. If installing rpm packages on multiple hosts, this is not an issue. However, if installing multiple platforms on a single host, a different path must be used for the RPM database for each platform (section 11.1 might be of interest).

After a path to the RPM database has been selected, it must be initialized for usage with the following command:

```
$ rpm --initdb [--dbpath=[db path]]
```

When an RPM package is installed, postinstall and postremove scripts are created in a temporary directory. The default path for this directory is '/var/opt/TWWfsw/rpm40/tmp'. To change this path, edit the value of the '%_tmppath' variable in /opt/TWWfsw/rpm40/lib/rpm/macros. The temporary directory must exist prior to installing any RPM package. If not, postinstall scripts will not run (though the package will be installed).

10.2 Package installation

The gzip package will be used to demonstrate how to install a package. The command to issue if installing the Tru64 UNIX gzip binaries on a Tru64 UNIX host is:

```
$ rpm -Uv --dbpath=/var/rpm/alpha-dec-osf4.0d TWWos-1.0-1.alpha.rpm
$ rpm -Uv --dbpath=/var/rpm/alpha-dec-osf4.0d TWWtin-1.4.2-1.alpha.rpm
```

The '--dbpath' option specifies the location of the rpm database.

The '-U' option installs the indicated package (rpm also accepts a '-i' argument as a replacement for '-U' for initial installs but because '-U' works for the initial installation and upgrades, it is used). Because of the way RPM packages are built, each package is dependent on '/bin/sh'. The package 'TWWos' provides this dependency and therefore must be installed prior to installing any package. The 'TWWos' package is not needed on Redhat Linux if using the default RPM database directory but will be needed if specifying a database directory differing from the default. The package does not replace '/bin/sh' but simply records in the RPM database that '/bin/sh' is available and provided by 'TWWos'.

Once the package is installed, scripts will execute following installation to create links in the common bin, info, and man directories. rpm will not prompt for the location of the common directories.

Because RPM packages are cross-platform, it is possible to install non-native binaries. The installation instructions are similar to the above but it is important that the RPM database directory differ for each architecture. Thus, to install the gzip binaries for Solaris 2.5.1 and Solaris 2.6 on a Tru64 UNIX host, the following would be done:

```
$ rpm -Uv --dbpath=/var/rpm/sparc-sun-solaris2.5.1 --relocate \
/opt/TWWfsw=/nfs/sparc-sun-solaris2.5.1 --ignorearch --ignoreos \
/tmp/solaris2.5.1-rpms/TWWos-1.0-1.sparc.rpm
```

```
$ rpm -Uv --dbpath=/var/rpm/sparc-sun-solaris2.5.1 --relocate \
/opt/TWWfsw=/nfs/sparc-sun-solaris2.5.1 --ignorearch --ignoreos \
/tmp/solaris2.5.1-rpms/TWWtin-1.4.2-1.sparc.rpm
$ rpm -Uv --dbpath=/var/rpm/sparc-sun-solaris2.6 --relocate \
/opt/TWWfsw=/nfs/sparc-sun-solaris2.6 --ignorearch --ignoreos \
/tmp/solaris2.6-rpms/TWWos-1.0-1.sparc.rpm
$ rpm -Uv --dbpath=/var/rpm/sparc-sun-solaris2.6 --relocate \
/opt/TWWfsw=/nfs/sparc-sun-solaris2.6 --ignorearch --ignoreos \
/tmp/solaris2.6-rpms/TWWtin-1.4.2-1.sparc.rpm
```

The '`--relocate`' option installs all files with a given prefix to an alternate directory. This is necessary if installing multiple architectures on the same platform. It is also possible to relocate '`/etc/opt/TWWfsw`' and '`/var/opt/TWWfsw`'. Only directories specified in the RPM package as relocatable can be relocated.

10.3 Package removal

Removing packages is done by replacing the '`-U`' option with the '`-e`' option. To remove the gzip package from the first example above:

```
$ rpm -ev --dbpath=/var/rpm/alpha-dec-osf4.0d TWWtin-1.4.2
$ rpm -ev --dbpath=/var/rpm/alpha-dec-osf4.0d TWWos-1.0-1
```

To remove the gzip package from the second example:

```
$ rpm -ev --dbpath=/var/rpm/sparc-sun-solaris2.5.1 TWWtin-1.4.2
$ rpm -ev --dbpath=/var/rpm/sparc-sun-solaris2.5.1 TWWos-1.0
$ rpm -ev --dbpath=/var/rpm/sparc-sun-solaris2.6 TWWtin-1.4.2
$ rpm -ev --dbpath=/var/rpm/sparc-sun-solaris2.6 TWWos-1.0
```

10.4 Package upgrade

When a package is upgraded, with the new version replacing the old by installing into the same directory, `rpm` adds new files that are part of the new package but not part of the old package and removes old files that are not part of the new package. If symbolic links have been created in a common bin, info, and man directory, the links will be preserved during the upgrade. If the package being overwritten contains configuration files, `rpm` will preserve changes to the old configuration files.

10.5 Procedure scripts

If the configuration file indicates that symbolic links should be created after the installation of a package, the package postinstall script will create symbolic links to the common bin, info, and man directories and add entries to the '`dir`' file for all GNU info files. Once the package is removed, a script is run to remove the entries from the '`dir`' file for all GNU info files and to remove the symbolic links from the common bin, info, and man directories.

The '`COMMON_BASE`' variable in the configuration file (cf. section 3.2.1) indicates the base directory to use when creating the links. Unlike the native package-management programs though, RPM postinstall scripts use '`COMMON_BASE_`*`systype`*'. Because RPM is cross-platform, a separate variable was needed to specify the common base directory in the event multiple architectures were installed on one platform. The possibility also existed that, depending on the architecture, the common base directory might be different. Therefore, RPM postinstall scripts do not honor the ‘`COMMON_BASE`’ variable but use ‘`COMMON_BASE_`*`systype`*’. Thus, using the cross-platform examples above, the global configuration file ‘`/etc/opt/TWWfsw/configs/default`’ might look like:

```
COMMON_BASE_sparc-sun-solaris2.5.1=/nfs/sparc-sun-solaris2.5.1/
COMMON_BASE_sparc-sun-solaris2.6=/nfs/sparc-sun-solaris2.6/
INST_UPGRADE=latest
INST_VERBOSE=1
```

Because the RPM package management system does not provide commands similar to the Solaris ‘`installf/removef`’ or the HP-UX ‘`swmodify`’ commands, some hacks were required to maintain the links to the common directories. When the package ‘`bin`’, ‘`info`’, or ‘`man`’ directories are installed, the postinstall script creates symbolic links in a common directory and registers the links in a file underneath the package installation directory. This file, ‘*`package`*`/tww-inst/links`’ is not registered with the RPM package management system nor are the links registered. Therefore, when invoking `rpm` with the ‘`-ql`’ option to list the files in the package, the ‘`links`’ file will not be registered. This file stores the type of link, the directory the link was created for, the source of the link, the destination, and the local destination (might be different from the destination of the link if ‘`--relocate`’ is used).

Unlike other package management systems, it is not possible to register a file as part of a package that is not part of the spec file. Thus, if the link information was not recorded, removal of the package would not remove the links. If a wrapper script is created, the entry for the wrapper in the file contains the pathname of the wrapper and its checksum (calculated with `sum(1)`). If a package is updated with a newer instance, this file is read and verified against what is present in the target installation tree. If any of the symbolic links are invalid or the wrapper script has changed (because the checksum has changed), the entry in the ‘`links`’ file will be updated to reflect the current state.

Once the package is removed, a preremove script is run to remove the symbolic links, remove wrapper scripts, and, if necessary, remove any entries from the common ‘`dir`’ file for any GNU info files. Because a record of the existing links has been created, this script reads the list and removes entries as appropriate.

Chapter 11

Installation Scenarios

11.1 Installation for multiple platforms onto a central server using RPM

Using RPM, it is possible to host packages for multiple platforms on a central server. This is often done when deploying the software to client workstations using NFS. While there are some deficiencies with RPM when installing on non-native platforms, it provides a solution to this problem.

By default, the RPM database resides in '`/var/opt/TWWfsw/rpm40/lib/rpm`'. This location works well if RPM is only being used to install packages for the system on which RPM is run. However, if using RPM to install packages from multiple platforms, a separate RPM database must be created for each platform. Because packages for each platform have the same name, RPM will be unable to differentiate between a package for one platform over another. The '`--dbpath=`*`path`*' option to RPM can be used to specify an alternate location for the RPM database.

In the examples to follow, RPM packages for the following platforms will be installed onto a Solaris 7 server: Solaris 2.5.1, HP-UX 10.20, and IRIX 6.5. Additional platforms such as Tru64 UNIX are installed in a similar manner. Before initializing the RPM database for each of the platforms, we must bootstrap the RPM installation process by installing the RPM package. As our host platform is Solaris 7, a Solaris pkgadd package for RPM must be installed.

The Solaris packages for RPM v4.0.4 are named `TWWrpm40u` for the RPM runtime and `TWWrpm40m` for the man pages. Installation is done as follows:

```
$ pkgadd -d . TWWrpm40u TWWrpm40m
```

As RPM is now configured, the location of the RPM databases must be selected as RPM will be used to install all future packages. For the examples to follow, '`/nfs/TWWfsw/`*`arch`*`/rpmdb`' will be the designated location, where '*arch*' will be '`sol251`' for Solaris 2.5.1, '`hpux10`' for HP-UX 10.20, and '`irix65`' for IRIX 6.5. RPM packages that would otherwise be installed into '`/opt/TWWfsw`' will be installed into '`/nfs/TWWfsw/`*`arch`*'. While the software must still run from '`/opt/TWWfsw`', the client workstations will mount the appropriate directory from the Solaris 7 server onto '`/opt/TWWfsw`' for execution. To initialize the RPM database for each of the platforms we intend to install for:

```
$ rpm --initdb --dbpath=/nfs/TWWfsw/hpux10/rpmdb
$ rpm --initdb --dbpath=/nfs/TWWfsw/irix65/rpmdb
$ rpm --initdb --dbpath=/nfs/TWWfsw/sol251/rpmdb
```

Because all RPM packages have postinstall scripts, they depend on `/bin/sh`. As the Solaris sytem provides `/bin/sh`, a package must be added to RPM that provides `/bin/sh` without copying in a new binary and overwriting the system `/bin/sh` binary. This is done through the '`Provides`' variable in an RPM spec file. The package that provides this is '`TWWos`'. It exists in the '`support`' directory of the first CD. It must be the first RPM package installed. Installation is done as follows:

```
$ rpm -Uv --dbpath=/nfs/TWWfsw/sol251/rpmdb --ignoreos --ignorearch
/tmp/pkgs/sol251/TWWos-1.0-1.sparc.rpm
$ rpm -Uv --dbpath=/nfs/TWWfsw/hpux10/rpmdb --ignoreos --ignorearch
/tmp/pkgs/hpux10/TWWos-1.0-1.sparc.rpm
$ rpm -Uv --dbpath=/nfs/TWWfsw/irix65/rpmdb --ignoreos --ignorearch
/tmp/pkgs/irix65/TWWos-1.0-1.sparc.rpm
```

Finally, installation of RPM packages can begin. The `--ignoreos` and `--ignorearch` arguments to RPM allow installation of non-native packages on the host platform. These options will be used in all of the following examples. Another option to be used will be the `--relocate` option. This option allows us to install all packages destined for '`/opt/TWWfsw/`*`package`*' to '`/nfs/TWWfsw/`*`arch`*`/`*`package`*'. It will also allow us to relocate the local configuration files that are installed into '`/var/opt/TWWfsw`' and '`/etc/opt/TWWfsw`'. Only `-conf` packages are installed in these directories. For NFS installs, these package components are usually ignored or the package is not installed.

To install the first package, GNU zip (gzip), the following commands are issued:

```
$ rpm -Uv --dbpath=/nfs/TWWfsw/sol251/rpmdb --ignoreos --ignorearch
  --relocate /opt/TWWfsw=/nfs/TWWfsw/sol251
  /tmp/pkgs/sol251/TWWgzip-1.3-1.sparc.rpm
  /tmp/pkgs/sol251/TWWgzip-man-1.3-1.sparc.rpm
$ rpm -Uv --dbpath=/nfs/TWWfsw/hpux10/rpmdb --ignoreos --ignorearch
  --relocate /opt/TWWfsw=/nfs/TWWfsw/hpux10
  /tmp/pkgs/sol251/TWWgzip-1.3-1.parisc.rpm
  /tmp/pkgs/sol251/TWWgzip-man-1.3-1.parisc.rpm
$ rpm -Uv --dbpath=/nfs/TWWfsw/irix65/rpmdb --ignoreos --ignorearch
  --relocate /opt/TWWfsw=/nfs/TWWfsw/irix65
  /tmp/pkgs/sol251/TWWgzip-1.3-1.mipseb.rpm
  /tmp/pkgs/sol251/TWWgzip-man-1.3-1.mipseb.rpm
```

If using **pkg-inst**, the equivalent commands are:

```
$ pkg-inst --pkg-type=rpm --rpm-db=/nfs/TWWfsw/sol251/rpmdb
  --rpm-arch=sparc -e --ignoreos -e --ignorearch
  -e --relocate -e /opt/TWWfsw=/nfs/TWWfsw/sol251
  --depot=/tmp/pkgs/sol251 gzip
$ pkg-inst --pkg-type=rpm --rpm-db=/nfs/TWWfsw/hpux10/rpmdb
  --rpm-arch=parisc -e --ignoreos -e --ignorearch
  -e --relocate -e /opt/TWWfsw=/nfs/TWWfsw/hpux10
  --depot=/tmp/pkgs/hpux10 gzip
$ pkg-inst --pkg-type=rpm --rpm-db=/nfs/TWWfsw/irix65/rpmdb
  --rpm-arch=mipseb -e --ignoreos -e --ignorearch
  -e --relocate -e /opt/TWWfsw=/nfs/TWWfsw/irix65
  --depot=/tmp/pkgs/irix65 gzip
```

Please note that even though installation does not occur to '`/opt/TWWfsw`', '`COMMON_BASE`' must still be configured in '/etc/opt/TWWfsw/configs/default' following the description given in section 3.2.1 (see also sections 3.2ff). Based on our configuration, our copy of '/etc/opt/TWWfsw/configs/default' would look like:

```
        COMMON_BASE_sparc_sun_solaris2_5_1=/nfs/TWWfsw/sol251/
        COMMON_BASE_hppa1_1_hp_hpux10_20=/nfs/TWWfsw/hpux10/
        COMMON_BASE_mips_sgi_irix6_5=/nfs/TWWfsw/irix65/
        INST_UPGRADE=latest
        INST_VERBOSE=1
```

In addition, even though the installation directory has been relocated, the destination of links created by the postinstall scripts will remain the same. Thus, if installing as normal to '/opt/TWWfsw' with 'COMMON_BASE_sparc-sun-solaris2.5.1' set to '/opt/TWWfsw/':

```
  $ cd /opt/TWWfsw/bin
  $ ls -l
  lrwxrwxrwx   1 root     other          27 Feb  2 08:59 gunzip -> /opt/TWWfsw/gzip/bin/gunzip
  lrwxrwxrwx   1 root     other          25 Feb  2 08:59 gzip -> /opt/TWWfsw/gzip/bin/gzip
```

And, if installing into '/nfs/TWWfsw/sol251' with 'COMMON_BASE_sparc-sun-solaris2.5.1' set to '/nfs/TWWfsw/sol251/':

```
  $ cd /nfs/TWWfsw/sol251/bin
  $ ls -l
  lrwxrwxrwx   1 root     other          27 Feb  2 08:59 gunzip -> /opt/TWWfsw/gzip/bin/gunzip
  lrwxrwxrwx   1 root     other          25 Feb  2 08:59 gzip -> /opt/TWWfsw/gzip/bin/gzip
```

Removing packages is similar to the method used to install a package. To remove the GNU zip (gzip) package, the following commands are used:

```
  $ rpm -ev --dbpath=/nfs/TWWfsw/sol251/rpmdb TWWgzip TWWgzip-man
  $ rpm -ev --dbpath=/nfs/TWWfsw/hpux10/rpmdb TWWgzip TWWgzip-man
  $ rpm -ev --dbpath=/nfs/TWWfsw/irix65/rpmdb TWWgzip TWWgzip-man
```

Package Installation Instructions

THE WRITTEN WORD

2003 November 14