Copying and Creating Package Repositories in Oracle® Solaris 11.2

Part No: E36805-02
September 2014

ORACLE®

Contents

Examples

Using This Documentation

- **Overview** – Describes how to create, copy, make accessible, update, and maintain a software package repository using the Oracle Solaris Image Packaging System (IPS) feature.
- **Audience** – System administrators who install and manage software or assist others who install and manage software.
- **Required knowledge** – Experience with the Oracle Solaris Service Management Facility (SMF) feature and experience administering NFS and web servers.

Product Documentation Library

Late-breaking information and known issues for this product are included in the documentation library at http://www.oracle.com/pls/topic/lookup?ctx=E36784.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

Image Packaging System Package Repositories

Oracle Solaris 11 software is distributed in Image Packaging System (IPS) packages. IPS packages are stored in IPS package repositories, which are populated by IPS publishers.

This guide describes how to create a software package repository using the Oracle Solaris Image Packaging System (IPS) feature. IPS tools enable you to easily copy an existing repository or create your own repository for your own packages and easily update the packages in the repository. You can provide a file interface or an HTTP or HTTPS interface for users of the repository. This guide also describes how to automatically update your repository and how to clone a repository, and shows Apache web server configuration such as caching, load balancing, and configuring HTTPS access.

This chapter provides:

- Reasons that you might want to create a local IPS package repository for internal use
- Best practices for creating package repositories
- System requirements for hosting a repository

Local IPS Repositories

You might want a local IPS repository for the following reasons:

- **Performance and security.** You do not want your client systems to go to the Internet to retrieve new software packages or update existing packages.
- **Change control.** You want to ensure that you can perform the same installation next year that you perform today. You want to easily control the versions to which systems can be updated.
- **Custom packages.** You want to deliver custom IPS packages.

Best Practices for Creating and Using Local IPS Package Repositories

Employ the following best practices to maintain repository availability and minimize errors.

Include all content of all Support Repository Updates (SRUs).

Keep local repositories updated with all support updates. Support updates contain security updates and other important fixes. Each minor release and update of the Oracle Solaris OS package repository is released as a full set of packages. SRUs are released as a sparse update of just the changed packages.

- Do not add a subset of packages from a support update to your repository. Add all of the content of the support update to your local repository.
- Do not skip a support update. Accumulate all applicable support updates in each repository.
- Do not remove packages that are delivered by an Oracle publisher.
- Use the `svc:/application/pkg/mirror` Service Management Facility (SMF) service to automatically update the local master repository from the Oracle support repository. See "How to Automatically Copy a Repository From the Internet" on page 22 for instructions.

Users can update to a version earlier than the latest version in the repository by specifying the version of the `entire` incorporation package to install. See Chapter 4, "Updating or Upgrading an Oracle Solaris Image," in "Adding and Updating Software in Oracle Solaris 11.2 ".

Verify every time you update the repository.

Use the `pkgrepo verify` command whenever you change the content or property values of the repository. The `pkgrepo verify` command verifies that the following attributes of the repository content are correct:

- File checksums.
- File permissions. The repository files and directories and the path leading to the repository are checked to ensure that the `pkg5srv` user can read the repository content.
- Package manifest permissions.
- Package manifest content.
- Package signatures.

Create repositories in a shared location.

A shared location is a location that is not in any bootable environment (BE). Examples of shared locations include `/var/share` and `/export`. Creating a repository in a shared location provides the following benefits:

- The repository is easily available from other existing BEs.
- When you create a new BE through upgrading or by cloning an existing BE, you do not waste space by having multiple copies of a repository.
- You do not waste time and I/O resources reapplying repository updates that you have already made in a different BE.

If you are using non-global zones, all locations of publishers configured in non-global zones must be accessible from the global zone even if that publisher is not configured in the global zone.

Create each repository in its own ZFS file system.

Using a separate ZFS file system enables you to do the following:

- Achieve better performance.
- Set separate file system characteristics. For example, set `atime` to `off` for better performance when updating the repository. The `atime` property controls whether the access time for files is updated when the files are read. Turning this property off avoids producing write traffic when reading files.
- Manage resource use. Specify an appropriate disk quota for each repository dataset to ensure that large repository updates do not consume all the space in the pool. This best practice is especially important if you are performing updates automatically as described in "How to Automatically Copy a Repository From the Internet" on page 22.
- Create snapshots.

Snapshot every time you update the repository.

Snapshot the repository file system every time you update the repository to gain the following benefits:

- Roll back to a previous version of the repository from a snapshot.
- Update the repository from a snapshot to minimize user disruption.

Provide high availability.

- Maintain repository clones in different locations. See "Maintaining Multiple Identical Local Repositories" on page 36 for instructions.

- Configure your web server for caching, load balancing, and serving multiple repositories. See Chapter 5, “Running the Depot Server Behind a Web Server” for information.

Secure your local repositories.

See “Configuring HTTPS Repository Access” on page 55 for instructions.

System Requirements

The system that hosts the IPS package repository can be either an x86-based or a SPARC-based system.

Operating system

Repository servers running Oracle Solaris 11 11/11 support all Oracle Solaris 11 update packages.

Disk space

To host a copy of the Oracle Solaris 11.2 release repository, the repository server must have 16 gigabytes of free space.

Because best practice is to keep local repositories updated with all support updates, plan to use 10-15 GB of additional space each year for support updates. Additional software, such as Oracle Solaris Studio or Oracle Solaris Cluster, of course requires additional space in the package repository.

If one system hosts more than one IPS repository, make each repository a separate ZFS file system so that you can rollback and recover each repository separately.

Repository Management Privileges

Use one of the following methods to gain the privilege you need to create and configure package repositories. See “Securing Users and Processes in Oracle Solaris 11.2 ” for more information about profiles and roles, including how to determine which profile or role you need.

Rights profiles

Use the `profiles` command to list the rights profiles that are assigned to you. The following profiles are useful for maintaining local package repositories:

ZFS File System Management

This rights profile enables you to run the `zfs` command.

Software Installation

This rights profile enables you to run the `pkg` command.

Service Management

This rights profile enables you to run SMF commands such as `svccfg`.

Roles

Use the `roles` command to list the roles that are assigned to you. If you have the `root` role, you can use the `su` command with the `root` password to assume the `root` role.

sudo command

Depending on the security policy at your site, you might be able to use the `sudo` command with your user password to execute a privileged command.

♦ ♦ ♦ CHAPTER 2

Copying IPS Package Repositories

This chapter describes two ways to create a copy of the Oracle Solaris IPS package repository: You can use repository files from media or from an Oracle Solaris download site, or you can retrieve the repository content from the Internet manually or automatically. In all cases, first create a separate ZFS file system in a shared location for your local package repository. After the repository is created, verify and snapshot the repository.

This chapter also provides performance and troubleshooting information related to copying repositories.

Performance Considerations for Copying Repositories

If you download repository files from the Oracle Solaris download site, or if you use the `pkgrecv` command shown in "Copying a Repository From the Internet" on page 21 to retrieve repository content from an Internet location, consider the following configuration to improve your transfer performance:

- Ensure that your ZFS storage pool capacity is less than 80%. Use the `zpool list` command to view your pool capacity.
- If you are using a proxy, check the performance of the proxy.
- Close applications that use a large amount of memory.
- Ensure adequate free space is available in the temporary directory. During its operations, the `pkgrecv` command uses `$TMPDIR` as a temporary storage directory. If `TMPDIR` is not set, `pkgrecv` uses `/var/tmp` for this temporary storage. Ensure that `$TMPDIR` or `/var/tmp` has enough free space for the size of the `pkgrecv` operation you are doing.
- If you are using the `pkgrecv` command to copy a large repository, consider using the `--clone` option. Using the `--clone` option is faster and consumes less memory. See "How to Clone a Local IPS Package Repository" on page 37.
- If you are using the `pkgrecv` command to create or update a large repository, consider using an SSD for the destination repository. You can move the repository as needed after the package retrieval is complete.

Troubleshooting Local Package Repositories

The following methods can prevent problems or help find the cause of problems you might encounter:

- Verify repository source files. If you use `.zip` files to create your repository, confirm that the files on your system are correct by using the checksums as described in "How to Copy a Repository From a `zip` File" on page 17.
- Verify the installed repository. Use the `pkgrepo verify` command to check your installed repository.

  The following permissions problems are reported by `pkgrepo verify`:

  - File permissions. To avoid problems with directory and file permissions for file system based repositories, ensure that the `pkg5srv` user has permission to read the repository.
  - Directory permissions. Ensure that all directories in the repository have execute permission.

  If the `pkgrepo verify` command reports other types of errors, try using the `pkgrepo fix` command to fix the errors. See the `pkgrepo`(1) man page for more information.
- Check your publisher origin. Make sure you set the origin for each publisher appropriately in each image. To update installed packages, install packages that depend on installed packages, or install a non-global zone, the repository that you set as the publisher origin must contain at least the same software that is installed in the image where you are setting the publisher. See Step 3 in "How to Enable Users to Retrieve Packages Using a File Interface" on page 27. See "Adding and Updating Software in Oracle Solaris 11.2 " for information about setting publishers and troubleshooting package installation problems.
- Check web server configuration. If you configure an Apache web server to access your repository, configure the web server to not decode encoded forward slashes. See the instructions in "Required Apache Configuration Setting" on page 50. Decoding encoded forward slashes can result in "package not found" errors.
- Do not create a repository that is only accessible from a non-global zone. All locations of publishers configured in non-global zones must be accessible from the global zone even if that publisher is not configured in the global zone.

Copying a Repository From a File

This section describes how to make a local copy of the Oracle Solaris package repository from one or more repository files. The repository files might be on media or might be available on an Oracle Solaris download site. The repository files might be `zip` files or `iso` files.

How to Copy a Repository From a zip File

1. Create a ZFS file system for the new repository.

Create the repository in a shared location. Set `atime` to `off` when you create the repository file system. See "Best Practices for Creating and Using Local IPS Package Repositories" on page 10.

```
$ zfs create -o atime=off rpool/export/IPSpkgrepos
$ zfs create rpool/export/IPSpkgrepos/Solaris
$ zfs get atime rpool/export/IPSpkgrepos/Solaris
NAME                              PROPERTY  VALUE  SOURCE
rpool/export/IPSpkgrepos/Solaris  atime     off    inherited from rpool/export/IPSpkgrepos
```

2. Get the package repository files.

Download the Oracle Solaris IPS package repository `.zip` files from the same location where you downloaded the system installation image, or locate the repository DVD in the media packet. Along with the `.zip` files, download the `install-repo.ksh` script, and the `.txt` files (the `README` and checksum files).

```
$ ls
install-repo.ksh           sol-11_2-ga-repo-3of4.zip
README-zipped-repo.txt     sol-11_2-ga-repo-4of4.zip
sol-11_2-ga-repo-1of4.zip  sol-11_2-ga-repo.txt
sol-11_2-ga-repo-2of4.zip
```

3. Make sure the script file is executable.

```
$ chmod +x install-repo.ksh
```

4. Run the repository installation script.

The repository installation script, `install-repo.ksh`, uncompresses each repository `.zip` file into the specified directory. The script optionally performs the following additional tasks:

- Verify checksums of the downloaded `.zip` files. If you do not specify the `-c` option to verify checksums, verify the checksums manually before you run the repository installation script. Run the following `digest` command, and compare the output with the appropriate checksum from the `.md5` file:

  ```
  $ digest -a md5 file
  ```

- Add the repository content to existing content if the specified destination already contains a repository.
- Verify the final repository. If you do not specify the `-v` option to verify the repository, use the `info`, `list`, and `verify` subcommands of the `pkgrepo` command to verify the repository after you run the repository installation script.

- Create an ISO image file for mounting and distribution. If you use the `-I` option to create an `.iso` file, the `.iso` file and the `README` file that explains how to use the `.iso` file are in specified destination directory.

5. **Verify the repository content.**

   If you did not specify the `-v` option in the previous step, use the `info`, `list`, and `verify` subcommands of the `pkgrepo` command to check that the repository has been copied correctly. If the `pkgrepo verify` command reports errors, try using the `pkgrepo fix` command to fix the errors. See the `pkgrepo`(1) man page for more information.

6. **Snapshot the new repository.**

```
$ zfs snapshot rpool/export/IPSpkgrepos/Solaris@sol-11_2_0
```

Example 2-1 Creating a New Repository From a `zip` File

In this example, no repository exists until the `zip` files are unpacked. The script can take the following options:

`-s`	Optional. Specifies the full path to the directory where the `.zip` files are located. Default: The current directory.
`-d`	Required. Specifies the full path to the directory where you want the repository.
`-i`	Optional. Specifies the files to use to populate this repository. The source directory could contain multiple sets of `.zip` files. Default: The newest image available in the source directory.
`-c`	Optional. Compares the checksums of the `.zip` files with the checksums in the specified file. If you specify `-c` with no argument, the default file used is the `.md5` file for the `-i` image in the source directory.
`-v`	Optional. Verifies the final repository.
`-I`	Optional. Creates an ISO image of the repository in the source directory. Also leaves a `mkiso.log` log file in the source directory.
`-h`	Optional. Displays a usage message.

```
$ ./install-repo.ksh -d /export/IPSpkgrepos/Solaris -c -v -I
Comparing checksums of downloaded files...done. Checksums match.
Uncompressing sol-11_2-ga-repo-1of4.zip...done.
Uncompressing sol-11_2-ga-repo-2of4.zip...done.
Uncompressing sol-11_2-ga-repo-3of4.zip...done.
Uncompressing sol-11_2-ga-repo-4of4.zip...done.
Repository can be found in /export/IPSpkgrepos/Solaris.
```

```
Initiating repository verification.
Building ISO image...done.
ISO image and instructions for using the ISO image are at:
/tank/downloads/sol-11_2-ga-repo.iso
/tank/downloads/README-repo-iso.txt
$ ls /export/IPSpkgrepos/Solaris
COPYRIGHT         NOTICES           pkg5.repository   publisher         README-iso.txt
```

The repository rebuild and verification can take some time, but the repository content is retrievable after you get the “Repository can be found in” message.

Example 2-2 Adding to an Existing Repository From a `zip` File

In this example, the content of the repository `zip` files is added to the content in an existing package repository.

```
$ pkgrepo -s /export/IPSpkgrepos/Solaris info
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4764     online           2014-03-18T05:30:57.221021Z
$ ./install-repo.ksh -d /export/IPSpkgrepos/Solaris -c -v -I
IPS repository exists at destination /export/IPSpkgrepos/Solaris
Current version: 0.175.2.0.0.35.0
Do you want to add to this repository? (y/n) y
Comparing checksums of downloaded files...done. Checksums match.
Uncompressing sol-11_2-ga-repo-1of4.zip...done.
Uncompressing sol-11_2-ga-repo-2of4.zip...done.
Uncompressing sol-11_2-ga-repo-3of4.zip...done.
Uncompressing sol-11_2-ga-repo-4of4.zip...done.
Repository can be found in /export/IPSpkgrepos/Solaris.
Initiating repository rebuild.
Initiating repository verification.
Building ISO image...done.
ISO image and instructions for using the ISO image are at:
/tank/downloads/sol-11_2-ga-repo.iso
/tank/downloads/README-repo-iso.txt
$ pkgrepo -s /export/IPSpkgrepos/Solaris info
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4768     online           2014-06-02T18:11:55.640930Z
```

▼ How to Copy a Repository From an `iso` File

1. Create a ZFS file system for the new repository.

Create the repository in a shared location. Set `atime` to `off` when you create the repository file system. See “Best Practices for Creating and Using Local IPS Package Repositories” on page 10.

```
$ zfs create -o atime=off rpool/export/IPSpkgrepos
$ zfs create rpool/export/IPSpkgrepos/Solaris
$ zfs get atime rpool/export/IPSpkgrepos/Solaris
```

```
NAME                              PROPERTY  VALUE  SOURCE
rpool/export/IPSpkgrepos/Solaris  atime     off    inherited from rpool/export/IPSpkgrepos
```

2. **Get the package repository image files.**

   Create an `.iso` file from the repository `.zip` files using the `-I` option as described in Example 2-1.

3. **Mount the image file.**

   Mount the repository `.iso` file to access the content.

   ```
   $ mount -F hsfs /path/sol-11_2-repo.iso /mnt
   ```

   To avoid the need to remount the `.iso` image each time the repository server system restarts, copy the repository file content as described in the next step.

4. **Copy the repository content to the new location.**

   To increase the performance of repository accesses and to avoid the need to remount the `.iso` image each time the system restarts, copy the repository files from `/mnt/repo/` to a ZFS file system. You can do this copy with the `rsync` command or with the `tar` command.

   - **Use the `rsync` command.**

     If you use the `rsync` command, be sure to specify `/mnt/repo/` (including the trailing slash character) and not `/mnt/repo` to copy the files and subdirectories in the `repo` directory. See the `rsync`(1) man page.

     ```
     $ rsync -aP /mnt/repo/ /export/IPSpkgrepos/Solaris
     ```

   - **Use the `tar` command.**

     Using the `tar` command as shown in the following example can be a faster way to copy the repository from the mounted file system to the repository ZFS file system.

     ```
     $ cd /mnt/repo; tar cf - . | (cd /export/IPSpkgrepos/Solaris; tar xfp -)
     ```

5. **Unmount the image file.**

   Make sure you are not still in the `/mnt` directory.

   ```
   $ umount /mnt
   ```

6. **Verify the new repository content.**

   Use the `info`, `list`, and `verify` subcommands of the `pkgrepo` command to check that the repository has been copied correctly. If the `pkgrepo verify` command reports errors, try using the `pkgrepo fix` command to fix the errors. See the `pkgrepo`(1) man page for more information.

7. **Snapshot the new repository.**

```
$ zfs snapshot rpool/export/IPSpkgrepos/Solaris@sol-11_2_0
```

Copying a Repository From the Internet

This section describes how to make a local copy of the Oracle Solaris package repository by copying the repository from an Internet location. The first procedure shows issuing the copy command from the command line. The second procedure shows using an SMF service to automatically copy and update a repository.

▼ How to Explicitly Copy a Repository From the Internet

1. **Create a ZFS file system for the new repository.**

   Create the repository in a shared location. Set `atime` to `off` when you create the repository file system. See "Best Practices for Creating and Using Local IPS Package Repositories" on page 10.

```
$ zfs create -o atime=off rpool/export/IPSpkgrepos
$ zfs create rpool/export/IPSpkgrepos/Solaris
$ zfs get atime rpool/export/IPSpkgrepos/Solaris
NAME                              PROPERTY  VALUE  SOURCE
rpool/export/IPSpkgrepos/Solaris  atime     off    inherited from rpool/export/IPSpkgrepos
```

2. **Create the required repository infrastructure.**

   Create the required `pkg`(5) repository infrastructure so that you can copy the repository. The image files used in the previous method include the repository infrastructure, so this step is not needed. When you copy repository content using the `pkgrecv` command as described in this method, you need to create the repository infrastructure and then copy the repository content into that infrastructure. See the `pkg`(5) and `pkgrepo`(1) man pages.

```
$ pkgrepo create /export/IPSpkgrepos/Solaris
```

3. **Copy the repository content to the new location.**

   Use the `pkgrecv` command to copy the repository. This operation could affect your network performance. The time required for this operation to complete depends on your network bandwidth and connection speed. See also "Performance Considerations for Copying Repositories" on page 15. If you update this repository later, only the changes are transferred, and the process can take much less time.

The following command retrieves all versions of all packages from the package repository specified by the `-s` option to the repository specified by the `-d` option. If you are copying from a secure site, ensure that the required SSL certificate and key are installed, and specify the required certificate and key options.

```
$ pkgrecv -s https://pkg.oracle.com/solaris/support -d /export/IPSpkgrepos/Solaris \
--key /path-to-ssl_key --cert /path-to-ssl_cert '*'
```

See the pkgrecv(1) man page for information about the `-m` and `--clone` options. You should not use the `-m latest` option for this purpose. Using a repository that is too sparse can result in errors when users attempt to update their images.

4. **Verify the new repository content.**

   Use the `info`, `list`, and `verify` subcommands of the `pkgrepo` command to check that the repository has been copied correctly. If the `pkgrepo verify` command reports errors, try using the `pkgrepo fix` command to fix the errors. See the pkgrepo(1) man page for more information.

5. **Snapshot the new repository.**

   ```
   $ zfs snapshot rpool/export/IPSpkgrepos/Solaris@sol-11_2_0
   ```

▼ How to Automatically Copy a Repository From the Internet

By default, the `svc:/application/pkg/mirror` SMF service performs a periodic `pkgrecv` operation from the `solaris` publisher origins defined in this image to `/var/share/pkg/repositories/solaris`. This `pkgrecv` operation starts at 2:30am one day each month. To change this default behavior, configure the service as described in this procedure.

At the end of each successful run of this service, the repository catalogs are refreshed. You do not need to refresh the repository to build a search index.

Because this service runs periodically, the repository is created and also kept updated. You do not need to use the manual repository update instructions shown in this document.

Other systems can set their `solaris` publisher origin to this automatically updated repository or to a clone of this repository. Only one system needs to have an Internet publisher origin and run the `mirror` service to automatically receive updates.

1. **Set publisher origins.**

By default, the `mirror` service transfers packages from the `solaris` publisher configured in the image rooted at /. Although you cannot directly specify publisher origins in the `mirror` service configuration, you can configure the image root from which to retrieve this information. In that image root, use `pkg set-publisher` to configure the publisher origins to use as the sources of the `pkgrecv` transfer for the mirror repository.

a. (Optional) Set the image root.

If the publisher configuration you want to use for the mirror service is different from the publisher configuration you want to use in this image, create a user image in a shared location (not contained in any BE) and reset the value of the `config/ref_image` property in the `mirror` service to that new image, as shown in the following example. The `mirror` service will use the publisher configuration from the `config/ref_image` image.

```
$ svccfg -s pkg/mirror:default setprop config/ref_image = /var/share/pkg/
mirror_svc_ref_image
$ pkg image-create /var/share/pkg/mirror_svc_ref_image
```

b. (Optional) Set the publishers.

If you want to update your mirror repository with packages from other publishers in addition to the `solaris` publisher, reset the value of the `config/publishers` property in the `mirror` service, as shown in the following example that shows adding the `ha-cluster` and `solarisstudio` publishers.

```
$ svccfg -s pkg/mirror:default setprop config/publishers = solaris,ha-
cluster,solarisstudio
```

c. Set the publisher origins.

Because this service runs periodically, you should set your publisher origins to a repository that provides regular updates. For Oracle products, you probably want to set your publisher origins to a support repository to retrieve Support Repository Updates (SRUs). In the following example, the `-R` option is needed only if you are configuring publishers in an alternate image root. The `-k` and `-c` options might not be needed, depending on the origin URIs.

```
$ pkg -R /var/share/pkg/mirror_svc_ref_image set-publisher \
-g https://pkg.oracle.com/solaris/support/ -k ssl_key -c ssl_cert solaris
$ pkg -R /var/share/pkg/mirror_svc_ref_image set-publisher \
-g https://pkg.oracle.com/ha-cluster/support/ -k ssl_key -c ssl_cert ha-cluster
$ pkg -R /var/share/pkg/mirror_svc_ref_image set-publisher \
-g https://pkg.oracle.com/solarisstudio/support/ -k ssl_key -c ssl_cert solarisstudio
```

Use one of the following commands to verify the publishers configured in the new image:

```
$ pkg -R /var/share/pkg/mirror_svc_ref_image publisher
$ pkg -R /var/share/pkg/mirror_svc_ref_image publisher solaris ha-cluster
 solarisstudio
```

2. **(Optional) Configure other properties of the mirror service.**

   You might want to modify other properties of the `mirror` service, such as the time the service runs or the location of the mirror repository.

   You might want to change the time the service runs to more closely match the time you expect the publisher origins being mirrored to be updated. To change the time the service runs, modify the value of the `config/crontab_period` property.

   To change the location of the mirror repository, modify the value of the `config/repository` property. If you change the location of the mirror repository, keep the repository in a shared location. See "Best Practices for Creating and Using Local IPS Package Repositories" on page 10. The default location, `/var/share/pkg/repositories/solaris`, is a shared location, not contained in any BE.

3. **Enable the mirror service.**

   Use the `svcs mirror` command to check the state of the `mirror` service.

   - **The service is disabled and you want to use this service.**

     a. **Refresh the service instance if you changed the configuration.**

        If you changed any of the configuration of the `mirror` service, as shown in the `svccfg setprop` commands in the previous steps, refresh the service to commit the changed values into the running snapshot. If the output from the `svcprop -p config mirror` command does not show the values you want, make sure the output from the `svccfg -s mirror:default listprop config` command shows the values you want. Use either `svcadm refresh mirror:default` or `svccfg -s mirror:default refresh` to commit the changed values into the running snapshot of the service. Use the `svcprop -p config mirror` command again to confirm that the service is configured the way you want it configured.

     b. **Enable the service instance.**

        Use the following command to enable the mirror service:

        ```
        $ svcadm enable mirror:default
        ```

        Use the `svcs mirror` command to confirm that the `mirror` service is online. The service will run at the time set in the `config/crontab_period` property.

   - **The service is online and you want to run the service now.**

     If the service is online, refresh the service to run the service immediately. You should see the `svc-pkg-mirror` method and the `pkgrecv` command being run by the `pkg5srv` user.

   - **The service is online and you do not want to use this service.**

Use the `svcadm disable mirror` command to disable this service. You might want to run this service on only one system to maintain a master repository. On other systems, you probably want to disable this service.

- **The service is in maintenance or is degraded.**

  Use the `svcs -xvL mirror` command to get more information to diagnose and fix the problem.

4. **Verify the repository content.**

   After the `mirror` service finishes a run, use the `info`, `list`, and `verify` subcommands of the `pkgrepo` command to check that the repository has been copied or updated correctly. If the `pkgrepo verify` command reports errors, try using the `pkgrepo fix` command to fix the errors. See the `pkgrepo`(1) man page for more information.

   Check the value of the `config/crontab_period` property of the `mirror` service to see when the service will run. While the service is running, the `svcs -p mirror` command shows the service state as `online*` and shows the processes started by this service. Wait until the service state shows as `online` and no processes are associated with the service before you verify the repository.

5. **Snapshot the new repository.**

   ```
   $ zfs snapshot rpool/VARSHARE/pkg/repositories/solaris@sol-11_2_0
   ```

Next Steps You might not want to copy content from multiple publishers at the same time. Instead of setting multiple publishers in one `config/publishers` property, you could create multiple instances of the `pkg/mirror` service. For example, the `config/publishers` property could be set to `solaris` for the `default` instance, to `ha-cluster` for a new `pkg/mirror:ha-cluster` instance, and to `solarisstudio` for a new `pkg/mirror:solarisstudio` instance. Similarly, the `config/crontab_period` could be set differently for each instance. You could store the content from each publisher in one repository, as shown in this procedure, or you could set a separate `config/repository` value for each `pkg/mirror` instance.

See Also See “Managing System Services in Oracle Solaris 11.2 ” for more information about SMF commands.

♦ ♦ ♦ CHAPTER 3

Providing Access To Your Repository

This chapter describes how to enable clients to retrieve packages in your local repository by using a file interface or by using an HTTP interface. One repository can be configured for both types of access.

Enabling Users to Retrieve Packages Using a File Interface

This section describes how to serve the local repository packages from a directory on your local network.

▼ How to Enable Users to Retrieve Packages Using a File Interface

1. **Configure an NFS share.**

   To enable clients to access the local repository by using NFS, create and publish an NFS share.

   ```
   $ zfs share -o share.nfs=on rpool/export/IPSpkgrepos%ipsrepo
   ```

   See the `zfs_share`(1M) man page for more information, such as additional `share.nfs` properties that you could set.

2. **Confirm that the share is published.**

   Use one of the following tests to confirm that the share is published:

   - **Search for the repository in the shared file system table.**

     ```
     $ grep repo /etc/dfs/sharetab
     /export/IPSpkgrepos     ipsrepo nfs     sec=sys,rw
     ```

   - **Determine whether the repository is accessible from a remote system.**

     ```
     $ dfshares solaris
     ```

```
RESOURCE                                  SERVER ACCESS  TRANSPORT
   solaris:/export/IPSpkgrepos           solaris  -       -
```

3. Set the publisher origin.

To enable client systems to get packages from your local file repository, set the origin for the publisher.

a. Determine the name of the publisher.

Use the following command to determine the names of publishers in your repository:

```
$ pkgrepo info -s /export/IPSpkgrepos/Solaris
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4768     online           2014-04-02T18:11:55.640930Z
```

b. Check the suitability of this publisher origin.

To update installed packages, install packages that depend on installed packages, or install a non-global zone, the repository that you set as the publisher origin must contain at least the same software that is installed in the image where you are setting the publisher. The repository can also contain older or newer software, but it must contain the same software that is installed in the image.

The following command shows that the specified repository is a not suitable publisher origin for this image:

```
$ pkg list entire
NAME (PUBLISHER)      VERSION                    IFO
entire                0.5.11-0.175.2.0.0.36.0    i--
$ pkgrepo list -Hs http://pkg.oracle.com/solaris/release
 entire@0.5.11-0.175.2.0.0.36.0
pkgrepo list: The following pattern(s) did not match any packages:
        entire@0.5.11-0.175.2.0.0.36.0
```

The following command shows that the specified repository is a suitable publisher origin for this image:

```
$ pkgrepo list -Hs /export/IPSpkgrepos/Solaris entire@0.5.11-0.175.2.0.0.36.0
solaris       entire       0.5.11,5.11-0.175.2.0.0.36.0:20140401T190148Z
```

c. Set the publisher origin.

Using the repository location and publisher name from the previous steps, run the following command to set the origin for the publisher:

```
$ pkg set-publisher -G '*' -M '*' -g /export/IPSpkgrepos/Solaris/ solaris
```

`-G '*'` Removes all existing origins for the `solaris` publisher.

`-M '*'` Removes all existing mirrors for the `solaris` publisher.

`-g` Adds the URI of the newly-created local repository as the new origin for the `solaris` publisher.

See "Configuring Publishers" in "Adding and Updating Software in Oracle Solaris 11.2 " for more information about configuring publishers.

If you reset the publisher origin in other images, perform the suitability test again: Other images might have a different version of software installed and might not be able to use this repository. If you reset the publisher origin in images on other systems, use a full path for the `-g` argument.

Enabling Users to Retrieve Packages Using an HTTP Interface

This section describes how to serve the local repository packages using the package depot server.

▼ How to Enable Users to Retrieve Packages Using an HTTP Interface

The package depot server, `pkg.depotd`, provides network access to the data contained within a package repository. The `svc:/application/pkg/server` SMF service invokes the `pkg.depotd` daemon. To enable clients to access the local repository by using HTTP, this procedure shows how to configure the `pkg/server` service. You could configure the `default` instance of the service. This procedure shows how to create and configure a new instance.

1. **Create a depot server instance.**

   Use the `add` subcommand to add a new instance of the `pkg/server` service named `solaris`.

   ```
   $ svccfg -s pkg/server add solaris
   ```

2. **Set the path to the repository.**

   Set the path where this instance of the service can find the repository data.

   ```
   $ svccfg -s pkg/server:solaris setprop pkg/inst_root=/export/IPSpkgrepos/Solaris
   ```

3. **(Optional) Set the port number.**

Set the port number on which the depot server instance should listen for incoming package requests. By default, `pkg.depotd` listens for connections on port 80. To change the port, reset the `pkg/port` property.

```
$ svccfg -s pkg/server:solaris setprop pkg/port=81
```

4. (Optional) Set other properties.

For a complete list of `pkg/server` properties, see the `pkg.depotd`(1M) man page.

To set multiple service properties, use the following command to edit all of the properties at once. Remember to remove the comment marker (#) from the beginning of any lines that you change.

```
$ svccfg -s pkg/server:solaris editprop
```

5. Start the repository service.

Restart the package depot server service.

```
$ svcadm refresh pkg/server:solaris
$ svcadm enable pkg/server:solaris
```

6. Test that the repository server is working.

To determine whether the repository server is working, open a browser window on the `localhost` location. By default, `pkg.depotd` listens for connections on port 80. If you have changed the port, open a browser window on the `localhost:`*port_number* location.

7. Set the publisher origin.

To enable client systems to get packages from your local file repository, set the origin for the publisher.

a. Determine the name of the publisher.

Use the following command to determine the names of publishers in your repository:

```
$ pkgrepo info -s /export/IPSpkgrepos/Solaris
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4768     online           2014-04-02T18:11:55.640930Z
```

b. Check the suitability of this publisher origin.

To update installed packages, install packages that depend on installed packages, or install a non-global zone, the repository that you set as the publisher origin must contain at least the same software that is installed in the image where you are setting the publisher. The repository can also contain older or newer software, but it must contain the same software that is installed in the image.

The following command shows that the specified repository is a not suitable publisher origin for this image:

```
$ pkg list entire
NAME (PUBLISHER)      VERSION                    IFO
entire                0.5.11-0.175.2.0.0.36.0    i--
$ pkgrepo list -Hs http://pkg.oracle.com/solaris/release
 entire@0.5.11-0.175.2.0.0.36.0
pkgrepo list: The following pattern(s) did not match any packages:
        entire@0.5.11-0.175.2.0.0.36.0
```

The following command shows that the specified repository is a suitable publisher origin for this image:

```
$ pkgrepo list -Hs http://localhost:81/ entire@0.5.11-0.175.2.0.0.36.0
solaris       entire       0.5.11,5.11-0.175.2.0.0.36.0:20140401T190148Z
```

c. Set the publisher origin.

Set the publisher origin to one of the following values:

- The `pkg/inst_root` location.

  ```
  $ pkg set-publisher -G '*' -M '*' -g /export/IPSpkgrepos/Solaris/ solaris
  ```

- The `pkg/port` location.

  ```
  $ pkg set-publisher -G '*' -M '*' -g http://localhost:81/ solaris
  ```

`-G '*'` Removes all existing origins for the `solaris` publisher.

`-M '*'` Removes all existing mirrors for the `solaris` publisher.

`-g` Adds the URI of the newly-created local repository as the new origin for the `solaris` publisher.

See "Configuring Publishers" in "Adding and Updating Software in Oracle Solaris 11.2 " for more information about configuring publishers.

If you reset the publisher origin in other images, perform the suitability test again: Other images might have a different version of software installed and might not be able to use this repository.

See Also

- "Serving Multiple Repositories Using Web Server Access" on page 44 describes how to serve multiple repositories from multiple locations or from a single location.
- "Multiple Repositories Under One Domain" on page 53 describes how to run multiple repositories under one domain name with different prefixes.
- "Configuring HTTPS Repository Access" on page 55 describes how to configure secure repository access.

♦ ♦ ♦ CHAPTER 4

Maintaining Your Local IPS Package Repository

This chapter describes how to update packages in an IPS repository, how to set or update properties of a repository, and how to add packages to a repository from a second source.

Updating Your Local Repository

The procedures shown in this section illustrate the following best practices for updating IPS package repositories:

- Keep each repository updated with all support updates for that release. Support updates contain security updates and other important fixes.
  - Do not try to choose particular fixes to apply from a support update. Do not add a subset of packages from a support update to your repository. Add all of the content of the support update to your local repository. The default behavior of the `pkgrecv` command is to retrieve all versions of all packages.
  - Do not skip a support update. Accumulate all applicable support updates in each repository.

  Users can update to a version earlier than the latest version in the repository by specifying the version of the `entire` incorporation package to install. See Chapter 4, “Updating or Upgrading an Oracle Solaris Image,” in “Adding and Updating Software in Oracle Solaris 11.2 ”.
- Update a copy of the repository. This practice helps ensure that systems do not access the repository while the repository is being updated. Create a snapshot of your repository before you update the repository, clone the snapshot, perform the update, and replace the original repository with the updated clone.

If you are maintaining multiple copies of package repositories with the same content, use the following procedure to update one of those identical repositories. See “Maintaining Multiple Identical Local Repositories” on page 36 for the procedure to update the additional repositories from this master repository.

How to Update a Local IPS Package Repository

Note - You do not need to perform this procedure if you use the `svc:/application/pkg/mirror` SMF service to periodically update your repository. See "How to Automatically Copy a Repository From the Internet" on page 22 for instructions for using the `mirror` service.

1. Make a ZFS snapshot of the package repository.

Make sure you have a current snapshot of the repository to be updated.

```
$ zfs list -t all -r rpool/export/IPSpkgrepos/Solaris
NAME                                          USED  AVAIL  REFER  MOUNTPOINT
rpool/export/IPSpkgrepos/Solaris             17.6G  78.4G    34K  /export/IPSpkgrepos/Solaris
rpool/export/IPSpkgrepos/Solaris@initial         0      -  17.6G  -
```

If you already have a snapshot of the repository, use the `zfs diff` command to check whether the snapshot is the same as the repository dataset.

```
$ zfs diff rpool/export/IPSpkgrepos/Solaris@initial
$
```

If the `zfs diff` command produces no output, then the snapshot is the same as its parent dataset, and you can use that snapshot for the update.

If the `zfs diff` command produces output, or if you do not have a snapshot of the repository, then take a new snapshot as shown in Step 6 in "How to Explicitly Copy a Repository From the Internet" on page 21. Use this new snapshot for the update.

2. Make a ZFS clone of the package repository.

Clone the repository snapshot to create a copy of the repository that you can update.

```
$ zfs clone rpool/export/IPSpkgrepos/Solaris@initial rpool/export/IPSpkgrepos/Solaris_tmp
$ zfs list -r rpool/export/IPSpkgrepos/Solaris/
NAME                                          USED  AVAIL  REFER  MOUNTPOINT
rpool/export/IPSpkgrepos/Solaris             17.6G  78.4G    34K  /export/IPSpkgrepos/Solaris
rpool/export/IPSpkgrepos/Solaris@initial         0      -  17.6G  -
rpool/export/IPSpkgrepos/Solaris_tmp          76K  78.4G  17.6G  /export/IPSpkgrepos/Solaris_tmp
```

3. Update the ZFS clone of the package repository.

Just as you created the original repository either from a file or from an HTTP location, you can update your repository either from a file or from an HTTP location.

- **Update from a `zip` file.**

  See Example 2-2. If the specified destination already contains a package repository, the content of the `zip` file is added to the content of the existing repository.

- **Update from an ISO file.**

  a. **Mount the ISO image.**

  ```
  $ mount -F hsfs ./sol-11_2-incr-repo.iso /mnt
  ```

  b. **Copy the ISO file content to the repository clone.**

  Use either `rsync` or `tar` as shown in "How to Copy a Repository From an `iso` File" on page 19.

  ```
  $ rsync -aP /mnt/repo/ /export/IPSpkgrepos/Solaris_tmp
  ```

  c. **Unmount the ISO image.**

- **Update from a repository.**

  Copy content from another repository to the repository clone. If you are copying from a secure site, ensure that the required SSL certificate and key are installed, and specify the required certificate and key options.

  ```
  $ pkgrecv -s https://pkg.oracle.com/solaris/support \
  -d /export/IPSpkgrepos/Solaris_tmp \
  --key /path-to-ssl_key --cert /path-to-ssl_cert '*'
  ```

  See the `pkgrecv`(1) man page for more information about the `pkgrecv` command. Only packages that have changed are updated, so the time to update your repository can be much less than the time to populate the original repository. See the performance tips in "Performance Considerations for Copying Repositories" on page 15.

  If the `pkgrecv` operation is interrupted, follow the instructions in "Resuming an Interrupted Package Receive" on page 36.

4. **Replace the working repository with the updated clone.**

```
$ svcadm disable -st pkg/server:solaris
$ zfs promote rpool/export/IPSpkgrepos/Solaris_tmp
$ zfs rename rpool/export/IPSpkgrepos/Solaris rpool/export/IPSpkgrepos/Solaris_old
$ zfs rename rpool/export/IPSpkgrepos/Solaris_tmp rpool/export/IPSpkgrepos/Solaris
```

See the `svcadm`(1M) man page for more information about the `svcadm` command.

5. **Verify the updated repository.**

Use the `pkgrepo verify` command to verify the updated repository. See the `pkgrepo`(1) man page for more information about the `pkgrepo verify` and `pkgrepo fix` commands.

6. **Catalog new packages and update search indexes.**

Catalog any new packages found in the newly updated repository and update all search indexes.

```
$ pkgrepo refresh -s rpool/export/IPSpkgrepos/Solaris
```

7. **Make a ZFS snapshot the newly updated clone of the package repository.**

```
$ zfs snapshot rpool/export/IPSpkgrepos/Solaris@S11U2SRU1
```

8. **Restart the SMF service.**

If you are providing the repository through an HTTP interface, restart the SMF service. Be sure to specify the appropriate service instance when you restart the service.

```
$ svcadm restart pkg/server:solaris
```

9. **Remove the old repository.**

When you are satisfied that your updated repository is working correctly, you can remove the old repository.

```
$ zfs destroy rpool/export/IPSpkgrepos/Solaris_old
```

Resuming an Interrupted Package Receive

If the `pkgrecv` operation is interrupted, use the `-c` option to retrieve content that was already downloaded and resume the content download. The value of *cache_dir* is supplied in an informational message when the transfer is interrupted, as shown in the following example:

```
PROCESS                    ITEMS       GET (MB)        SEND (MB)
...
pkgrecv: http protocol error: code: 503 reason: Service Unavailable
URL: 'https://pkg.oracle.com/solaris/support/file/file_hash

pkgrecv: Cached files were preserved in the following directory:
        /var/tmp/pkgrecv-fOGaIg
Use pkgrecv -c to resume the interrupted download.
$ pkgrecv -c /var/tmp/pkgrecv-fOGaIg \
-s https://pkg.oracle.com/solaris/support -d /export/IPSpkgrepos/Solaris_tmp \
--key /path/to/ssl_key --cert /path/to/ssl_cert '*'
Processing packages for publisher solaris ...
Retrieving and evaluating 156 package(s)...
```

Maintaining Multiple Identical Local Repositories

You might want to maintain multiple copies of package repositories with the same content to meet the following goals:

- Increase the availability of the repository by maintaining copies on different nodes.
- Enhance the performance of repository accesses if you have many users or your users are spread across a great distance.

Use the "How to Update a Local IPS Package Repository" on page 34 procedure to update one of your package repositories. Then use the "How to Clone a Local IPS Package Repository" on page 37 procedure to update additional identical repositories from the repository you updated first. These two procedures are very similar, with an important difference in the way you use the `pkgrecv` command. The `pkgrecv` operation shown in the clone procedure copies the source repository files exactly, with the following effects:

- Timestamps for the catalogs of cloned repositories are exactly the same as timestamps for the catalogs of the source repository. If your repositories are load balanced, the catalogs in all of the repositories should be exactly the same to avoid problems when the load balancer switches clients from one node to another. See "Configuring Load Balancing" on page 54 for information about load balancing.
- Packages that are in the destination repository but not in the source repository are removed from the destination repository. Do not use a sparse repository as the source for a clone operation unless your goal is to create an exact copy of only that sparse repository.

▼ How to Clone a Local IPS Package Repository

See "How to Update a Local IPS Package Repository" on page 34 for details of these steps.

1. **Copy the destination repository.**

   Make sure you have a current snapshot of the destination repository. Make a ZFS clone of this snapshot.

2. **Update the copy of the destination repository.**

   Use the `pkgrecv` command to clone your previously updated local package repository to the copy of the destination repository. See the `pkgrecv(1)` man page for more information about the `pkgrecv` clone operation.

   ```
   $ pkgrecv -s /net/host1/export/IPSpkgrepos/Solaris \
   -d /net/host2/export/IPSpkgrepos/Solaris_tmp --clone
   ```

3. **Replace the working destination repository with the updated clone.**

4. **Verify the updated repository.**

   Use the `pkgrepo verify` command to verify the updated destination repository.

5. **Snapshot the newly updated repository.**

6. **Restart the SMF service.**

   If you are providing the repository through an HTTP interface, restart the SMF service. Be sure to specify the appropriate service instance when you restart the service.

7. **Remove the old repository.**

   When you are satisfied that your updated repository is working correctly, remove the old repository.

See Also If you are providing the repository through an HTTP interface, see the following related documentation:

- "Serving Multiple Repositories Using Web Server Access" on page 44 describes how to serve multiple repositories using multiple `pkg.depotd` daemons running on different ports.
- "Multiple Repositories Under One Domain" on page 53 describes how to run multiple repositories under one domain name with different prefixes.

Checking and Setting Repository Properties

This section describes how to display information about an IPS repository and how to change repository property values.

Viewing Properties that Apply to the Entire Repository

The following command displays a list of the package publishers known by the local repository. The STATUS column indicates whether the publisher's package data is currently being processed.

```
$ pkgrepo info -s /export/IPSpkgrepos/Solaris
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4506     online           2013-07-11T23:32:46.379726Z
```

The following command displays property information that applies to the entire repository. See the pkgrepo(1) man page for a complete list of repository properties and their descriptions, including specifications of their values.

```
$ pkgrepo get -s /export/IPSpkgrepos/Solaris
SECTION    PROPERTY                    VALUE
publisher  prefix                      solaris
repository check-certificate-revocation False
repository signature-required-names     ()
repository trust-anchor-directory      /etc/certs/CA/
```

```
repository version                    4
```

`publisher/prefix`

The name of the default publisher. Though a repository can contain packages from multiple publishers, only one of the publishers can be set as the default publisher. This default publisher name is used for the following purposes:

- To identify a package when no publisher is specified in the package FMRI in the `pkg` command
- To assign a publisher to a package when the package is published to the repository (using the `pkgsend`(1) command) and no publisher is specified in the package manifest

`repository/check-certificate-revocation`

A flag for checking the certificate. When set to `True`, the `pkgrepo verify` command attempts to determine whether the certificate has been revoked since being issued. This value must match the value of the `check-certificate-revocation` image property described in "Additional Image Properties" in "Adding and Updating Software in Oracle Solaris 11.2 " and in the `pkg`(1) man page.

`repository/signature-required-names`

A list of names that must be seen as common names of certificates while validating the signatures of a package. This list is used by the `pkgrepo verify` command. This value must match the value of the `signature-required-names` image property described in "Image Properties for Signed Packages" in "Adding and Updating Software in Oracle Solaris 11.2 " and in the `pkg`(1) man page.

`repository/trust-anchor-directory`

The absolute path name of the directory that contains the trust anchors for packages in this repository. The default is `/etc/certs/CA/`. This value must match the value of the `trust-anchor-directory` image property described in "Additional Image Properties" in "Adding and Updating Software in Oracle Solaris 11.2 " and in the `pkg`(1) man page.

`repository/version`

The format version of the repository. This value cannot be set with the `pkgrepo set` command shown in "Modifying Repository Property Values" on page 41. This value

can be set with the `pkgrepo create` command. Version 4 repositories are created by default. Version 4 repositories support storage of packages for multiple publishers.

Viewing Repository Publisher Properties

The following command displays property information about the `solaris` publisher in the local repository. Parentheses indicate that the value can be a list of values.

```
$ pkgrepo get -p solaris -s /export/IPSpkgrepos/Solaris
PUBLISHER SECTION    PROPERTY         VALUE
solaris   publisher  alias
solaris   publisher  prefix           solaris
solaris   repository collection-type  core
solaris   repository description      ""
solaris   repository legal-uris       ()
solaris   repository mirrors          ()
solaris   repository name             ""
solaris   repository origins          ()
solaris   repository refresh-seconds  ""
solaris   repository registration-uri ""
solaris   repository related-uris     ()
```

`publisher/prefix`

The name of the publisher specified in the `-p` option. If no `-p` option is specified, this value is the name of the default publisher for this repository, as described in the previous section.

`repository/collection-type`

The type of packages in this repository. If the value is `core`, this repository contains all of the dependencies declared by packages in the repository. If the value is `supplemental`, this repository does not contain all of the dependencies declared by packages in the repository.

`repository/description`

The purpose and contents of this repository. If this repository is available from an HTTP interface, this value displays in the About section near the top of the main page.

`repository/legal-uris`

A list of locations for documents that provide legal information about the repository.

`repository/mirrors`

A list of locations of repositories that contain the same package content as this repository.

`repository/name`

The name of this repository. If this repository is available from an HTTP interface, this value displays at the top of the main page and in the window title.

`repository/origins`

A list of locations of repositories that contain the same package content and metadata as this repository.

`repository/refresh-seconds`

The number of seconds for clients to wait between checks for updated package data in this repository.

`repository/registration-uri`

The location of a resource that must be used to obtain credentials for access to this repository.

`repository/related-uris`

A list of locations of repositories that contain other packages that might be of interest.

The following command displays information about the specified *section/property* in the `pkg.oracle.com` repository.

```
$ pkgrepo get -p solaris -s http://pkg.oracle.com/solaris/release \
repository/name repository/description
PUBLISHER SECTION    PROPERTY    VALUE
solaris   repository description This\ repository\ serves\ the\ Oracle\ Solaris\ 11\ Package\
 repository.
solaris   repository name        Oracle\ Solaris\ 11\ Package\ Repository
```

Modifying Repository Property Values

"Viewing Repository Publisher Properties" on page 40 shows that the repository name and description property values are not set for the `solaris` publisher in the local repository. If this repository is available from an HTTP interface and you use a browser to view the content of this repository, you see a default name and no description. After you set these values, the publisher `repository/name` value is displayed near the top of the page and as the page title, and the publisher `repository/description` value is displayed in the About section just below the name. You must use the `-p` option to specify at least one publisher when you set these values. If this repository contains content from more than one publisher, you can set different values for each publisher, or you can specify `-p all`.

```
$ pkgrepo set -p solaris -s /export/IPSpkgrepos/Solaris \
```

```
repository/description="Local copy of the Oracle Solaris 11 repository." \
repository/name="Oracle Solaris 11"
$ pkgrepo get -p solaris -s /export/IPSpkgrepos/Solaris repository/name repository/
description
PUBLISHER SECTION    PROPERTY         VALUE
solaris   repository description      Local\ copy\ of\ the\ Oracle\ Solaris\ 11\ repository.
solaris   repository name             Oracle\ Solaris\ 11
```

Customizing Your Local Repository

You can use the `pkgrecv` command to add packages and their publisher data to your repository. You can use the `pkgrepo` command to remove packages and publishers from your repository.

Adding Packages to Your Repository

You can add publishers to a repository. For example, you could maintain `solaris`, `ha-cluster`, and `solarisstudio` packages in one repository.

If you add custom packages, publish those packages under a custom publisher name. Do not publish custom packages as an existing publisher such as `solaris`. If you publish packages that do not have a publisher specified, those packages will be added to the default publisher for the repository. Publish custom packages to a test repository with the correct default publisher. Then use the `pkgrecv` command to add those packages and their publisher information to your production repository. See "Publish the Package" in "Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.2 " for instructions.

In the following example, the `isvpub` publisher data and all of the packages from the `ISVproducts.p5p` package archive are added to the local repository. A *package archive* is a file that contains publisher information and one or more packages provided by that publisher. See "Deliver as a Package Archive File" in "Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.2 ". Most `pkgrepo` operations are not available for package archives. A package archive contains packages but does not contain repository configuration. However, the `pkgrepo list` and `pkgrepo contents` commands work with package archives. The `pkgrepo contents` command is discussed in "Examining Packages In Your Repository" on page 43.

In the `pkgrepo list` output, the publisher is shown because it is not the publisher that is highest ranked in search order in this image.

```
$ pkgrepo -s /tmp/ISVproducts.p5p list
PUBLISHER NAME                                  O VERSION
isvpub    isvtool                                 1.1,5.11:20131120T021902Z
```

```
isvpub    isvtool                                  1.0,5.11:20131120T010105Z
```

The following `pkgrecv` command retrieves all packages from the source repository. If you list names of packages to retrieve, or you specify a pattern other than `'*'`, you should specify the `-r` option to ensure you retrieve all necessary dependency packages.

```
$ pkgrecv -s /tmp/ISVproducts.p5p -d /export/IPSpkgrepos/Solaris '*'
Processing packages for publisher isvpub ...
Retrieving and evaluating 2 package(s)...
PROCESS        ITEMS      GET (MB)      SEND (MB)
Completed        2/2       0.0/0.0          0.0/0
```

After you change the content of a repository, refresh the repository and restart any package depot server service instance configured for this repository.

```
$ pkgrepo -s /export/IPSpkgrepos/Solaris refresh -p isvpub
Initiating repository refresh.
$ svcadm refresh pkg/server:solaris
$ svcadm restart pkg/server:solaris
```

The following `pkgrepo info` command shows one package because the two packages that were retrieved are different versions of the same package. The `pkgrepo list` command shows both packages.

```
$ pkgrepo -s /export/IPSpkgrepos/Solaris info
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4768     online           2014-01-02T19:19:06.983979Z
isvpub    1        online           2014-03-20T23:24:37.196773Z
$ pkgrepo -s /export/IPSpkgrepos/Solaris list -p isvpub
PUBLISHER NAME                                    O VERSION
isvpub    isvtool                                   1.1,5.11:20131120T021902Z
isvpub    isvtool                                   1.0,5.11:20131120T010105Z
```

Add the new repository location for the `isvpub` publisher by using the `pkg set-publisher` command.

If this repository is available from an HTTP interface and you use a browser to view the content of this repository, you can view this new package by specifying the publisher in the location. For example, you can specify `http://localhost:81/isvpub/`.

Examining Packages In Your Repository

In addition to the `pkgrepo info` and `pkgrepo list` commands shown in “Adding Packages to Your Repository” on page 42, you can use the `pkgrepo contents` command to examine the content of packages in your repository.

For a single package, the output from the `pkgrepo contents` command is the same as the output from the `pkg contents -m` command. The `pkgrepo contents` command displays

the output for each matching package in the specified repository, while the `pkg contents` command displays output only for versions of matching packages that are installable in this image. If you specify the `-t` option, the `pkgrepo contents` command shows only the specified actions.

The following example does not need to specify the version of the package because only one version of this package exists in the specified repository. This package contains `depend` actions to provide the set of Oracle Solaris packages required for installation and operation of Oracle Database 12.

```
$ pkgrepo -s http://pkg.oracle.com/solaris/release/ \
contents -t depend oracle-rdbms-server-12cR1-preinstall
depend fmri=x11/library/libxi type=group
depend fmri=x11/library/libxtst type=group
depend fmri=x11/session/xauth type=group
depend fmri=compress/unzip type=require
depend fmri=developer/assembler type=require
depend fmri=developer/build/make type=require
```

Removing Packages From Your Repository

Do not remove packages that are delivered by an Oracle publisher. “Adding and Updating Software in Oracle Solaris 11.2 ” shows methods for installing only the packages you want and avoiding installing packages that you do not want.

You can use the `pkgrepo remove` command to remove packages that were not delivered by an Oracle publisher. You can use the `pkgrepo remove-publisher` command to remove a publisher and all of the packages delivered by that publisher. See the `pkgrepo`(1) man page for details. These operations should be performed on a copy of the repository, as described in “How to Update a Local IPS Package Repository” on page 34.

Serving Multiple Repositories Using Web Server Access

The procedures in this section show how to extend the information provided in “Enabling Users to Retrieve Packages Using an HTTP Interface” on page 29 to support serving multiple repositories.

The following methods are two different ways to serve multiple IPS package repositories using HTTP access. For both methods, start by creating additional instances of the `pkg/server` service with unique repository paths.

- Multiple locations. Users access each repository by viewing pages at separate locations.

- Single location. Users access all repositories from one location.

In addition to providing access to multiple repositories, remember that a single repository can provide packages from multiple publishers, as shown in "Adding Packages to Your Repository" on page 42.

▼ How to Serve Multiple Repositories From Separate Locations

In this example, the `SolarisStudio` repository exists in addition to the `Solaris` repository. The `Solaris` repository is accessible from `http://localhost/` using port 81, as specified in the `solaris` instance of the `pkg/server` service. See "Enabling Users to Retrieve Packages Using an HTTP Interface" on page 29.

1. **Create a new depot server instance.**

   Use the `add` subcommand of the `svccfg` command to add a new instance of the `pkg/server` service.

   ```
   $ svccfg -s pkg/server add studio
   ```

2. **Check that you have added the new instance.**

   ```
   $ svcs pkg/server
   STATE  STIME    FMRI
   online 14:54:16 svc:/application/pkg/server:default
   online 14:54:20 svc:/application/pkg/server:studio
   online 14:54:20 svc:/application/pkg/server:solaris
   ```

3. **Set the path to the repository.**

   Set the path where this instance of the service can find the repository data.

   ```
   $ svccfg -s pkg/server:studio setprop pkg/inst_root=/export/IPSpkgrepos/SolarisStudio
   ```

4. **(Optional) Set the port number for the new instance.**

   ```
   $ svccfg -s pkg/server:studio setprop pkg/port=82
   ```

5. **(Optional) Set the Apache proxy base.**

   See "Configuring a Simple Prefixed Proxy" on page 52 for an example of setting the `pkg/proxy_base`.

6. **Set the repository name and description.**

   Make sure the repository name and description are set as shown in "Modifying Repository Property Values" on page 41.

7. **Start the repository service.**

   Restart the package depot server service.

   ```
   $ svcadm refresh pkg/server:studio
   $ svcadm enable pkg/server:studio
   ```

8. **Test that the repository server is working.**

   Open a browser window on the `http://localhost:82/` location.

   If you did not set the port number, the default is 80. View your repository at `http://localhost:80/` or `http://localhost/`.

   If the port number is also being used by another `pkg/server` instance, append the publisher name to the location to see the new packages. For example, view your repository at `http://localhost:81/solarisstudio/`.

9. **Set the publisher origin.**

   Set the publisher origin to one of the following values:

   - The `pkg/inst_root` location.

     ```
     $ pkg set-publisher -G '*' -M '*' -g /export/IPSpkgrepos/SolarisStudio/ \
     solarisstudio
     ```

   - The `pkg/port` location.

     ```
     $ pkg set-publisher -G '*' -M '*' -g http://localhost:82/ solarisstudio
     ```

See Also See "Multiple Repositories Under One Domain" on page 53 for information about running multiple repositories under one domain name with different prefixes such as `http://pkg.example.com/solaris` and `http://pkg.example.com/studio`.

▼ How to Serve Multiple Repositories From a Single Location

Many of the steps in this procedure are the same as the steps in the previous procedure. See the previous procedure for details.

1. **Create a new depot server instance.**

2. **Set the path to the repository.**

   Each `pkg/server` instance that is managed by a particular `pkg/depot` instance must have a unique `pkg/inst_root` value.

3. Check the `readonly` property for the new instance.

The default value of the `pkg/readonly` property is `true`. If this value has been changed, reset the value to `true`.

```
$ svcprop -p pkg/readonly pkg/server:studio
true
```

4. Set the `standalone` property for the new instance.

By default, the value of the `pkg/standalone` property is `true`. Any `pkg/server` instances whose `pkg/standalone` property is set to `false` can be served from the same location by a `pkg/depot` service instance.

```
$ svccfg -s pkg/server:studio
svc:/application/pkg/server:studio> setprop pkg/standalone=false
svc:/application/pkg/server:studio> refresh
svc:/application/pkg/server:studio> select solaris
svc:/application/pkg/server:solaris> setprop pkg/standalone=false
svc:/application/pkg/server:solaris> refresh
svc:/application/pkg/server:solaris> exit
$
```

Make sure the value of the `pkg/inst_root` property is unique for each instance of `pkg/server` whose `pkg/standalone` property is set to `false`.

5. (Optional) Set the port number for the `pkg/depot` instance.

By default, the port number of the `svc:/application/pkg/depot:default` service is 80. This port number can be the same as the port number for any of the `pkg/server` instances that will be managed by this `pkg/depot` instance. To change the port number, set the `config/port` property of `pkg/depot:default`.

6. Restart the `pkg/depot` instance.

```
$ svcadm refresh pkg/depot:default
$ svcadm restart pkg/depot:default
```

7. Test that the repository server is working.

When users open the `http://localhost:80/` location, they see the `http://localhost/solaris` repository listed with the `solaris` publisher, and they see the `http://localhost/studio` repository listed with the `solarisstudio` publisher.

If one repository provides packages for multiple publishers, all publishers are listed. For example, users might see the `http://localhost/solaris` repository listed with the `solaris` and `isvpub` publishers.

8. Set the publisher origin.

Set the publisher origin to one of the following values:

- The unique `pkg/inst_root` location.

```
$ pkg set-publisher -G '*' -M '*' -g /export/IPSpkgrepos/SolarisStudio/ \
solarisstudio
```

- The location defined by the value of `config/port` plus the `pkg/server` instance name.

```
$ pkg set-publisher -G '*' -M '*' -g http://localhost:80/studio/ solarisstudio
```

Next Steps If you change the content of a repository that is managed by a `pkg/depot` instance, as discussed in "Updating Your Local Repository" on page 33 and "Customizing Your Local Repository" on page 42, perform both of the following steps:

- Run `pkgrepo refresh` on the repository.
- Run `svcadm restart` on the `pkg/depot` instance.

You can create additional instances of the `pkg/depot` service where each instance hosts one or more repositories.

To generate a standalone configuration rather than configuring `pkg/server` and `pkg/depot` service instances, see the `pkg.depot-config`(1M) man page.

♦ ♦ ♦ CHAPTER 5

Running the Depot Server Behind a Web Server

Running the depot server behind an Apache web server instance provides the following benefits:

- Allows hosting multiple repositories under one domain name. The `pkg`(5) depot server enables you to easily provide access to a repository in the local network or on the Internet. However, the depot server does not support serving multiple repositories under one domain name or sophisticated prefixes. To host multiple repositories under one domain name, run the depot server behind a web proxy.
- Improves performance and availability. Running the depot server behind a web proxy can improve the performance and availability of the server by enabling load balancing over multiple depots and enabling content caching.
- Enables providing a secure repository server. Run the depot server behind a Secure Sockets Layer (SSL) protocol enabled Apache instance that supports client certificates.

Depot Server Apache Configuration

The examples in this chapter use the Apache web server as the proxy software. Activate the Apache web server by enabling the `svc:/network/http:apache22` service. See Apache HTTP Server Version 2.2 Documentation for additional information.

You should be able to apply the principles shown in these examples to any proxy server software.

The Oracle Solaris 11.2 OS includes the Apache web server in the `web/server/apache-22` package, which delivers a basic `httpd.conf` file in `/etc/apache2/2.2`. In general, you can use the following command to locate the `httpd.conf` file:

```
$ pkg search -Hl -o path ':file:path:*httpd.conf'
etc/apache2/2.2/httpd.conf
etc/apache2/2.2/original/httpd.conf
```

Required Apache Configuration Setting

If you run the package depot server behind an Apache web server instance, include the following setting in your `httpd.conf` file to not decode encoded forward slashes:

```
AllowEncodedSlashes NoDecode
```

Package names can contain URL encoded forward slashes because forward slashes are used to express hierarchical package names. For example, the package name `pkg://solaris/developer/build/make` becomes `http://pkg.oracle.com/solaris/release/manifest/0/developer%2Fbuild%2Fmake` to the web server. To prevent these forward slashes from being interpreted as directory delimiters, instruct Apache not to decode the `%2F` encoded slashes.

Omitting this setting can result in `404 Not Found` errors and can very negatively impact search functionality.

Recommended Generic Apache Configuration Settings

The following settings affect performance and security.

Reduce the over-the-wire size of metadata.

HTTP clients can tell the server that they accept compressed data in an HTTP request. Enabling the Apache DEFLATE filter can dramatically reduce the over-the-wire size of metadata such as catalogs and manifests. Metadata such as catalogs and manifests often compress 90%.

```
AddOutputFilterByType DEFLATE text/html application/javascript text/css text/plain
```

Allow more pipelined requests.

Increase the `MaxKeepAliveRequests` value to allow clients to make a larger number of pipelined requests without closing the connection.

```
MaxKeepAliveRequests 10000
```

Set the maximum wait time for response.

The proxy timeout sets how long Apache waits for the back-end depot to respond. For most operations, 30 seconds is satisfactory. Searches with a very large number of results can take significantly longer. You might want a higher timeout value to accommodate such searches.

```
ProxyTimeout 30
```

Disable forward proxying.

Make sure that forward proxying is disabled.

```
ProxyRequests Off
```

Configuring Caching for the Depot Server

Minimal configuration is required to set up the depot server behind a caching proxy. With the exception of the catalog attributes file (see “Cache Considerations for the Catalog Attributes File” on page 52) and repository search results (see “Cache Considerations for Search” on page 52), all files served are unique and therefore safe to cache indefinitely if necessary. Also, all depot responses contain the appropriate HTTP headers to ensure files in the cache do not become stale by mistake.

See the Apache Caching Guide for more information about configuring Apache as a caching proxy.

Use the `CacheRoot` directive to specify the directory to contain the cached files. Make sure the specified directory is writable by the Apache process. No explicit error message is output if Apache cannot write to this directory.

```
CacheRoot /tank/proxycache
```

Apache allows you to enable caching for specific directories. You probably want your repository server to cache all of the content on the server, as shown in the following directive.

```
CacheEnable disk /
```

Use the `CacheMaxFileSize` directive to set the maximum size of files to be cached. The Apache default of 1 MB might be too small for most repositories. The following directive sets the maximum cached file size to 1 GB.

```
CacheMaxFileSize 1000000000
```

Adjust the directory structure of the on-disk cache for the best performance with the underlying file system. In a ZFS dataset, multiple directory levels affect performance more than the number of files in one directory. Therefore, configure one directory level with a large number of files in each directory. Use the `CacheDirLevels` and `CacheDirLength` directives to control the directory structure. Set `CacheDirLevels` to 1. Set `CacheDirLength` to a value that results in a good balance between the number of directories and the number of files per directory. The value of 2 set below will generate 4096 directories. See the Apache Disk-based Caching documentation for more information.

```
CacheDirLevels 1
CacheDirLength 2
```

Cache Considerations for the Catalog Attributes File

The repository catalog attributes file (`catalog.attrs`) contains the current status of the repository catalog. This file can be large enough to warrant caching. However, this file becomes stale if the catalog of the back-end repository has changed. You can use one of the following two methods to address this issue.

- Do not cache this file. This solution works best if the repository server runs in a high-bandwidth environment where the additional traffic is not an important consideration. The following partial `httpd.conf` file shows how to specify not to cache the `catalog.attrs` file:

  ```
  <LocationMatch ".*/catalog.attrs">
          Header set Cache-Control no-cache
  </LocationMatch>
  ```

- Prune this file from the cache whenever the catalog of the back-end repository is updated.

Cache Considerations for Search

Searching a package repository generates custom responses based on the request. Therefore, search results are not well suited for being cached. The depot server sets the appropriate HTTP headers to make sure search results do not become stale in a cache. However, the expected bandwidth savings from caching are small. The following partial `httpd.conf` file shows how to specify not to cache search results.

```
<LocationMatch ".*/search/\d/.*">
        Header set Cache-Control no-cache
</LocationMatch>
```

Configuring a Simple Prefixed Proxy

This example shows the basic configuration for a non-load-balanced depot server. This example connects `http://pkg.example.com/myrepo` to `internal.example.com:10000`.

See "Serving Multiple Repositories Using Web Server Access" on page 44 for instructions about setting other properties you need that are not described in this example.

Configure the depot server with a `pkg/proxy_base` setting that names the URL where the depot server can be accessed. Use the following commands to set the `pkg/proxy_base`:

```
$ svccfg -s pkg/server add repo
$ svccfg -s pkg/server:repo setprop pkg/proxy_base = astring: http://pkg.example.com/
myrepo
$ svcadm refresh pkg/server:repo
$ svcadm enable pkg/server:repo
```

The pkg(5) client opens 20 parallel connections to the depot server when performing network operations. Make sure the number of depot threads matches the expected connections to the server at any given time. Use the following commands to set the number of threads per depot:

```
$ svccfg -s pkg/server:repo setprop pkg/threads = 200
$ svcadm refresh pkg/server:repo
$ svcadm restart pkg/server:repo
```

Use nocanon to suppress canonicalization of URLs. This setting is important in order for search to work well. Also, limit the number of back-end connections to the number of threads the depot server provides. The following partial `httpd.conf` file shows how to proxy one depot server:

```
Redirect /myrepo http://pkg.example.com/myrepo/
ProxyPass /myrepo/ http://internal.example.com:10000/ nocanon max=200
```

For information about the Oracle Solaris SSL kernel proxy and using SSL to encrypt and accelerate web server communications, see Chapter 3, “Web Servers and the Secure Sockets Layer Protocol,” in “Securing the Network in Oracle Solaris 11.2 ”.

Multiple Repositories Under One Domain

The most important reason to run the depot server behind a proxy is to easily run several repositories under one domain name with different prefixes. The example from “Configuring a Simple Prefixed Proxy” on page 52 can be easily extended to support multiple repositories.

In this example, three different prefixes of one domain name are connected to three different package repositories:

- `http://pkg.example.com/repo_one` is connected to `internal.example.com:10000`
- `http://pkg.example.com/repo_two` is connected to `internal.example.com:20000`
- `http://pkg.example.com/xyz/repo_three` is connected to `internal.example.com:30000`

The pkg(5) depot server is an SMF managed service. Therefore, to run multiple depot servers on the same host, simply create a new service instance:

```
$ svccfg -s pkg/server add repo1
$ svccfg -s pkg/server:repo1 setprop pkg/property=value
$ ...
```

Like the previous example, each depot server runs with 200 threads.

```
Redirect /repo_one http://pkg.example.com/repo_one/
ProxyPass /repo_one/ http://internal.example.com:10000/ nocanon max=200

Redirect /repo_two http://pkg.example.com/repo_two/
ProxyPass /repo_two/ http://internal.example.com:20000/ nocanon max=200

Redirect /xyz/repo_three http://pkg.example.com/xyz/repo_three/
ProxyPass /xyz/repo_three/ http://internal.example.com:30000/ nocanon max=200
```

Configuring Load Balancing

You might want to run depot servers behind an Apache load balancer. One benefit of load balancing is to increase the availability of your repository. This section shows two examples of load balancing.

If your repositories are load balanced, the catalogs in all of the repositories should be exactly the same to avoid problems when the load balancer switches clients from one node to another. To ensure the catalogs are exactly the same, clone the repositories that participate in load balancing as described in “Maintaining Multiple Identical Local Repositories” on page 36.

One Repository Server With Load Balancing

This example connects `http://pkg.example.com/myrepo` to `internal1.example.com:10000` and `internal2.example.com:10000`.

Configure the depot server with an appropriate `proxy_base` setting as shown in “Configuring a Simple Prefixed Proxy” on page 52.

Limit the number of back-end connections to the number of threads each depot is running divided by the number of depots in the load-balancer setup. Otherwise, Apache opens more connections to a depot than are available and they stall, which can decrease performance. Specify the maximum number of parallel connections to each depot with the `max=` parameter. The following example shows two depots, each running 200 threads. See “Configuring a Simple Prefixed Proxy” on page 52 for an example of how to set the number of depot threads.

```
<Proxy balancer://pkg-example-com-myrepo>
        # depot on internal1
        BalancerMember http://internal1.example.com:10000 retry=5 max=100

        # depot on internal2
        BalancerMember http://internal2.example.com:10000 retry=5 max=100
</Proxy>

Redirect /myrepo http://pkg.example.com/myrepo/
ProxyPass /myrepo/ balancer://pkg-example-com-myrepo/ nocanon
```

One Load-Balanced and One Non-Load-Balanced Repository Server

This example includes all of the directives you need to add to the `httpd.conf` file for a repository server that hosts a load-balanced and a non-load-balanced depot server setup.

In this example, two different prefixes of one domain name are connected to three different package repositories:

- `http://pkg.example.com/repo_one` is connected to `internal1.example.com:10000` and `internal2.example.com:10000`
- `http://pkg.example.com/repo_two` is connected to `internal1.example.com:20000`

```
AddOutputFilterByType DEFLATE text/html application/javascript text/css text/plain

AllowEncodedSlashes NoDecode

MaxKeepAliveRequests 10000

ProxyTimeout 30

ProxyRequests Off

<Proxy balancer://pkg-example-com-repo_one>
        # depot on internal1
        BalancerMember http://internal1.example.com:10000 retry=5 max=100

        # depot on internal2
        BalancerMember http://internal2.example.com:10000 retry=5 max=100
</Proxy>

Redirect /repo_one http://pkg.example.com/repo_one/
ProxyPass /repo_one/ balancer://pkg-example-com-repo_one/ nocanon
Redirect /repo_two http://pkg.example.com/repo_two/
ProxyPass /repo_two/ http://internal.example.com:20000/ nocanon max=200
```

Configuring HTTPS Repository Access

Any client can download packages from a repository that is configured to serve packages over HTTP. In some cases, you need to restrict access. One way to restrict access to the repository is to run the depot server behind an SSL-enabled Apache instance that supports client certificates.

Using SSL provides the following benefits:

- Ensures encrypted transfer of package data between the client and the server
- Enables you to grant access to repositories based on the certificate the client presents to the server

To set up a secure repository server, you must create a custom certificate chain:

1. Create a certificate authority (CA), which is the head of the certificate chain.
2. Issue certificates from this CA to the clients that are allowed to access the repository.

One copy of the CA is stored on the repository server. Whenever a client presents a certificate to the server, that client certificate is verified against the CA on the server to determine whether to grant access.

This section describes the following steps to create the certificate chain and configure the Apache front end to verify client certificates:

- Create a keystore
- Create a certificate authority for client certificates
- Add SSL configuration to the Apache configuration file
- Create a self-signed server certificate authority
- Create a PKCS12 keystore

For information about Apache web server privileges in Oracle Solaris, see “Locking Down Resources by Using Extended Privileges” in “Securing Users and Processes in Oracle Solaris 11.2 ”.

Creating a Keystore

To manage certificates and keys, create a keystore. The keystore stores the CA, the CA key, and client certificates and keys.

The tool used for keystore management is `pktool`. See the `pktool`(1) man page for more information.

The default keystore location for `pktool` is `/var/user/`*username*, where *username* is the name of the current system user. This keystore default location can be problematic when a keystore is managed by multiple users. In addition, IPS package repository management should have a dedicated keystore to avoid confusing certificates. To set a custom location for the `pktool` keystore for the IPS package repository, set the environment variable `SOFTTOKEN_DIR`. Reset the `SOFTTOKEN_DIR` variable as necessary to manage multiple keystores.

Use the following commands to create a directory for the keystore. Set the owner, group, and permissions appropriately if multiple users need to manage the keystore.

```
$ mkdir /path-to-keystore
$ export SOFTTOKEN_DIR=/path-to-keystore
```

Access to the keystore is protected by a passphrase that you must enter every time you invoke the `pktool` command. The default passphrase for a newly created keystore is `changeme`. Be sure to change the `changeme` passphrase to a more secure passphrase.

Use the following command to set the passphrase (PIN) for the keystore:

```
$ pktool setpin
Enter token passphrase: changeme
Create new passphrase:
Re-enter new passphrase:
Passphrase changed.
$ ls /path-to-keystore
pkcs11_softtoken
```

Creating a Certificate Authority for Client Certificates

The CA is the top-level certificate in your certificate chain. The CA is required to generate client certificates and to validate the certificates presented by clients to access a repository.

Third-party CAs are managed by a handful of trusted companies such as VeriSign. This trusted management enables clients to verify the identity of a server against one of their CAs. The example in this section does not include verifying the identity of the repository server. This example only shows verifying client certificates. Therefore, this example uses a self-signed certificate to create the CA and does not use any third-party CAs.

The CA requires a common name (CN). If you run only one repository, you might want to set the CN to the name of your organization (for example, “Oracle Software Delivery”). If you have multiple repositories, each repository must have its own CA. In this case, set the CN to a name that uniquely identifies the repository for which you are creating the CA. For example, if you have a release repository and a support repository, only certificates from the release CA will allow access to the release repository, and only certificates from the support CA will allow access to the support repository.

To identify the certificate in the keystore, set a descriptive label for the certificate. A good practice is to set the certificate label to *CN*_`ca`, where *CN* is the CN of the certificate.

Use the following command to create the CA certificate, where *name* is the certificate CN and *CAlabel* is the certificate label:

```
$ pktool gencert label=CAlabel subject="CN=name" serial=0x01
```

The CA will be stored in your keystore. Use the following command to show the contents of your keystore:

```
$ pktool list
```

You will need to extract the CA certificate from the keystore when you configure Apache as described in “Add SSL Configuration to the Apache Configuration File” on page 60. Use the following command to extract the CA certificate to a file named `ca_file.pem`:

```
$ pktool export objtype=cert label=CAlabel outformat=pem \
```

```
outfile=ca_file.pem
```

Creating Client Certificates Used for Accessing the Repository

After you have generated the CA, you can generate client certificates.

Generating a Certificate Signing Request

To generate a client certificate, generate a Certificate Signing Request (CSR). The CSR contains all of the information that you need to pass securely to the server.

If you only want to check whether the client possesses a valid certificate issued by you, you do not need to encode any information. When the client presents its certificate to the server, the server validates the certificate against the CA, verifying whether that client certificate was generated by you. However, SSL requires a `subject` for the CSR. If you do not need to pass any other information to the server, you can just set the `subject` to the country where the certificate has been issued. For example, you could set the `subject` to `C=US`.

A good practice is to encode the user name of the client into the certificate to enable the server to identify the client. The user name is the name of the user to whom you are giving access to the repository. You can use the CN for this purpose. Specify a label for this CSR so that you can find and extract the key for the final certificate as described in "Extracting the Certificate Key" on page 59.

Use the following command to generate the CSR:

```
$ pktool gencsr subject="C=US,CN=username" label=label format=pem \
outcsr=cert.csr
```

Use the following OpenSSL command to inspect the CSR in the file `cert.csr`:

```
$ openssl req -text -in cert.csr
```

Signing the CSR

The CSR must be signed by the CA to create a certificate. To sign the CSR, provide the following information:

- Set the `issuer` of the certificate to the same string that you used for the `subject` when you created the CA using the `gencert` command, as shown in "Creating a Certificate Authority for Client Certificates" on page 57.

- Set a hexadecimal serial number. In this example, the CA serial number was specified as `0x01`, so the first client certificate should be given the serial number `0x02`. Increment the serial number for each new client certificate that you generate.

  Each CA and its descendant client certificates has its own set of serial numbers. If you have multiple CAs configured in your keystore, be careful to set client certificate serial numbers correctly.
- Set the `signkey` to the label of the CA in the keystore.
- Set `outcert` to the name of the certificate file. A good practice is to name the certificate and key after the repository to be accessed.

Use the following command to sign the CSR:

```
$ pktool signcsr signkey=CAlabel csr=cert.csr  \
serial=0x02 outcert=reponame.crt.pem issuer="CN=name"
```

The certificate is created in the file *reponame*`.crt.pem`. Use the following OpenSSL command to inspect the certificate:

```
$ openssl x509 -text -in reponame.crt.pem
```

Extracting the Certificate Key

Extract the key for this certificate from the keystore. Set the `label` to the same label value you specified when you ran `gencsr` to generate the CSR in "Generating a Certificate Signing Request" on page 58. Use the following command to export the key from the keystore:

```
$ pktool export objtype=key label=label outformat=pem \
outfile=reponame.key.pem
```

Transfer the certificate and key to the client systems that need to access the SSL-protected repository.

Enabling Client Systems to Access the Protected Repository

To access the SSL-protected repository, client systems must have a copy of the certificate and key and must specify the certificate and key in the publisher configuration.

Copy the certificate (*reponame*`.crt.pem`) and key (*reponame*`.key.pem`) to each client system. For example, you could copy them to the `/var/pkg/ssl` directory on each client.

Use the following command to specify the generated certificate and key in your publisher configuration:

```
$ pkg set-publisher -k reponame.key.pem -c reponame.crt.pem \
-p https://repolocation
```

Note that SSL authentication is only supported for HTTPS repository URIs. SSL authentication is not supported for file repository URIs.

Add SSL Configuration to the Apache Configuration File

To use client certificate based authentication for your repository, first set up a generic depot server Apache configuration as described in "Depot Server Apache Configuration" on page 49. Then add the following SSL configuration at the end of your `httpd.conf` file:

```
# Let Apache listen on the standard HTTPS port
Listen 443

# VirtualHost configuration for request on port 443
<VirtualHost 0.0.0.0:443>
        # DNS domain name of the server, needs to match your server certificate
        ServerName pkg-sec.example.com

        # enable SSL
        SSLEngine On

        # Location of the server certificate and key.
        # You either have to get one from a certificate signing authority like
        # VeriSign or create your own CA for testing purposes (see "Creating a
        # Self-Signed CA for Testing Purposes")
        SSLCertificateFile /path/to/server.crt
        SSLCertificateKeyFile /path/to/server.key

        # Intermediate CA certificate file. Required if your server certificate
        # is not signed by a top-level CA directly but an intermediate authority
        # Comment out this section if you are using a test certificate or your
        # server certificate doesn't require it.
        # For more info:
        # http://httpd.apache.org/docs/2.2/mod/mod_ssl.html#sslcertificatechainfile
        SSLCertificateChainFile /path/to/ca_intermediate.pem

        # CA certs for client verification.
        # This is where the CA certificate created in step 3 needs to go.
        # If you have multiple CAs for multiple repos, just concatenate the
        # CA certificate files
        SSLCACertificateFile /path/to/ca_cert.pem

        # If the client presents a certificate, verify it here. If it doesn't,
        # ignore.
        # This is required to be able to use client-certificate based and
        # anonymous SSL traffic on the same VirtualHost.
        # This statement could also go into the <Location> tags but putting it
        # here avoids re-negotiation which can cause security issues with older
        # servers/clients:
        # http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2009-3555
        SSLVerifyClient optional
```

```
        <Location /repo>
                SSLVerifyDepth 1
                # This is the SSL requirement for this location.
                # Requirements can be made based on various information encoded
                # in the certificate. Two variants are the most useful for use
                # with IPS repositories:
                # a) SSLRequire ( %{SSL_CLIENT_I_DN_CN} =~ m/reponame/ )
                #    only allow access if the CN in the client certificate matches
                #    "reponame", useful for different certificates for different
                #    repos
                #
                # b) SSLRequire ( %{SSL_CLIENT_VERIFY} eq "SUCCESS" )
                #    grant access if clients certificate is signed by one of the
                #    CAs specified in SSLCACertificateFile
                SSLRequire ( %{SSL_CLIENT_VERIFY} eq "SUCCESS" )

                # proxy request to depot running at internal.example.com:12345
                ProxyPass http://internal.example.com:12345 nocanon max=500
        </Location>
</VirtualHost>
```

Creating a Self-Signed Server Certificate Authority

For testing purposes, you can use a self-signed server certificate authority (CA) rather than a third-party CA. The steps to create a self-signed server CA for Apache are very similar to the steps to create a CA for client certificates described in “Creating a Certificate Authority for Client Certificates” on page 57.

Use the following command to create a server CA. Set the `subject` to the DNS name of the server.

```
$ pktool gencert label=apacheCA subject="CN=apachetest" \
serial=0x01
```

Use the following command to create a CSR for a server CA. If the server is accessible under several names or you want to make it available under its IP address directly, use the `subjectAltNames` directive as described in Subject Alternative Name in the OpenSSL documentation.

```
$ pktool gencsr label=apache subject="CN=pkg-sec.internal.example.com" \
altname="IP=192.168.1.1,DNS=pkg-sec.internal.example.com" \
format=pem outcsr=apache.csr
```

Use the following command to sign the CSR. Use `server.crt` for `SSLCertificateFile`.

```
$ pktool signcsr signkey=apacheCA csr=apache.csr serial=0x02 \
outcert=server.crt issuer="CN=apachetest"
```

Use the following command to extract the key. Use `server.key` for `SSLCertificateKeyFile`.

```
$ pktool export objtype=key label=apache outformat=pem \
outfile=server.key
```

To ensure that your client will accept this server key, add the CA certificate (`apacheCA`) to the accepted CA directory on the client system and restart the `ca-certificates` service to create the required links for OpenSSL.

Use the following command to extract the CA certificate:

```
$ pktool export label=apacheCA objtype=cert outformat=pem \
outfile=test_server_ca.pem
```

Copy the CA certificate to the CA certificate directory on the client machine:

```
$ cp /path-to/test_server_ca.pem /etc/certs/CA/
```

Restart the CA certificates service:

```
$ svcadm refresh ca-certificates
```

Before you proceed, ensure that your new CA cert has been linked. After refreshing, the `ca-certificate` service rebuilds the links in the `/etc/openssl/certs` directory. Run the following command to check whether your new CA cert has been linked:

```
$ ls -l /etc/openssl/certs | grep test_server_ca.pem
lrwxrwxrwx   1 root     root          40 May  1 09:51 e89d96e0.0 -> ../../certs/CA/
test_server_ca.pem
```

The hash value, `e89d96e0.0`, might be different for you since it is based on the subject of your certificate.

Creating a PKCS12 Keystore to Access a Secure Repository With Firefox

The PEM certificates created in "Creating Client Certificates Used for Accessing the Repository" on page 58 will work to access the secured repository with the `pkg` client. However, to access the browser user interface (BUI), you must convert the certificate and key to a format that Firefox can import. Firefox accepts PKCS12 keystores.

Use the following OpenSSL command to create the PKCS12 keystore for Firefox:

```
$ openssl pkcs12 -export -in /path-to/certificate.pem \
-inkey /path-to/key.pem -out name.p12
```

To import the PKCS12 keystore that you just created, select the following Firefox menus, tabs, and buttons: Edit > Preferences > Advanced > Encryption > View certificates > Authorities > Import.

Import one certificate at a time.

Complete Secure Repositories Example

This example configures three secure repositories named `repo1`, `repo2`, and `repo3`. The `repo1` and `repo2` repositories are configured with dedicated certificates. Therefore, certificates for `repo1` will not work on `repo2`, and certificates for `repo2` will not work on `repo1`. The `repo3` repository is configured to accept either certificate.

The example assumes you have a proper server certificate for your Apache instance already available. If you do not have a server certificate for your Apache instance, see the instructions for creating a test certificate in "Creating a Self-Signed Server Certificate Authority" on page 61.

The three repositories are set up under `https://pkg-sec.example.com/repo1`, `https://pkg-sec.example.com/repo2`, and `https://pkg-sec.example.com/repo3`. These repositories point to depot servers set up at `http://internal.example.com` on ports 10001, 10002, and 10003 respectively. Make sure the `SOFTTOKEN_DIR` environment variable is set correctly as described in "Creating a Keystore" on page 56.

▼ How to Configure Secure Repositories

1. Create a CA certificate for `repo1`.

```
$ pktool gencert label=repo1_ca subject="CN=repo1" serial=0x01
$ pktool export objtype=cert label=repo1_ca outformat=pem \
outfile=repo1_ca.pem
```

2. Create a CA certificate for `repo2`.

```
$ pktool gencert label=repo2_ca subject="CN=repo2" serial=0x01
$ pktool export objtype=cert label=repo2_ca outformat=pem \
outfile=repo2_ca.pem
```

3. Create a combined CA certificate file.

```
$ cat repo1_ca.pem > repo_cas.pem
$ cat repo2_ca.pem >> repo_cas.pem
$ cp repo_cas.pem /path-to-certs
```

4. Create one client certificate/key pair to allow user `myuser` to access repository `repo1`.

```
$ pktool gencsr subject="C=US,CN=myuser" label=repo1_0001 format=pem \
outcsr=repo1_myuser.csr
$ pktool signcsr signkey=repo1_ca csr=repo1_myuser.csr  \
serial=0x02 outcert=repo1_myuser.crt.pem issuer="CN=repo1"
$ pktool export objtype=key label=repo1_0001 outformat=pem \
```

```
outfile=repo1_myuser.key.pem
$ cp repo1_myuser.key.pem /path-to-certs
$ cp repo1_myuser.crt.pem /path-to-certs
```

5. **Create one client certificate/key pair to allow user `myuser` to access repository `repo2`.**

```
$ pktool gencsr subject="C=US,CN=myuser" label=repo2_0001 format=pem \
outcsr=repo2_myuser.csr
$ pktool signcsr signkey=repo2_ca csr=repo2_myuser.csr  \
serial=0x02 outcert=repo2_myuser.crt.pem issuer="CN=repo2"
$ pktool export objtype=key label=repo2_0001 outformat=pem \
outfile=repo2_myuser.key.pem
$ cp repo2_myuser.key.pem /path-to-certs
$ cp repo2_myuser.crt.pem /path-to-certs
```

6. **Configure Apache.**

Add the following SSL configuration at the end of your `httpd.conf` file:

```
# Let Apache listen on the standard HTTPS port
Listen 443

<VirtualHost 0.0.0.0:443>
        # DNS domain name of the server
        ServerName pkg-sec.example.com

        # enable SSL
        SSLEngine On

        # Location of the server certificate and key.
        # You either have to get one from a certificate signing authority like
        # VeriSign or create your own CA for testing purposes (see "Creating a
        # Self-Signed CA for Testing Purposes")
        SSLCertificateFile /path/to/server.crt
        SSLCertificateKeyFile /path/to/server.key

        # Intermediate CA certificate file. Required if your server certificate
        # is not signed by a top-level CA directly but an intermediate authority.
        # Comment out this section if you don't need one or if you are using a
        # test certificate
        SSLCertificateChainFile /path/to/ca_intermediate.pem

        # CA certs for client verification.
        # This is where the CA certificate created in step 3 needs to go.
        # If you have multiple CAs for multiple repos, just concatenate the
        # CA certificate files
        SSLCACertificateFile /path/to/certs/repo_cas.pem

        # If the client presents a certificate, verify it here. If it doesn't,
        # ignore.
        # This is required to be able to use client-certificate based and
        # anonymous SSL traffic on the same VirtualHost.
        SSLVerifyClient optional
```

```
        <Location /repo1>
                SSLVerifyDepth 1
                SSLRequire ( %{SSL_CLIENT_I_DN_CN} =~ m/repo1/ )
                # proxy request to depot running at internal.example.com:10001
                ProxyPass http://internal.example.com:10001 nocanon max=500
        </Location>

        <Location /repo2>
                SSLVerifyDepth 1
                SSLRequire ( %{SSL_CLIENT_I_DN_CN} =~ m/repo2/ )
                # proxy request to depot running at internal.example.com:10002
                ProxyPass http://internal.example.com:10002 nocanon max=500
        </Location>

        <Location /repo3>
                SSLVerifyDepth 1
                SSLRequire ( %{SSL_CLIENT_VERIFY} eq "SUCCESS" )
                # proxy request to depot running at internal.example.com:10003
                ProxyPass http://internal.example.com:10003 nocanon max=500
        </Location>

</VirtualHost>
```

7. **Test access to `repo1`.**

```
$ pkg set-publisher -k /path-to-certs/repo1_myuser.key.pem \
-c /path-to-certs/repo1_myuser.crt.pem \
-p https://pkg-sec.example.com/repo1/
```

8. **Test access to `repo2`.**

```
$ pkg set-publisher -k /path-to-certs/repo2_myuser.key.pem \
-c /path-to-certs/repo2_myuser.crt.pem \
-p https://pkg-sec.example.com/repo2/
```

9. **Test access to `repo3`.**

   Use the `repo1` certificate to test access to `repo3`.

```
$ pkg set-publisher -k /path-to-certs/repo1_myuser.key.pem \
-c /path-to-certs/repo1_myuser.crt.pem \
-p https://pkg-sec.example.com/repo3/
```

   Use the `repo2` certificate to test access to `repo3`.

```
$ pkg set-publisher -k /path-to-certs/repo2_myuser.key.pem \
-c /path-to-certs/repo2_myuser.crt.pem \
-p https://pkg-sec.example.com/repo3/
```

Index

I

K

M

N

O

P

V

W

Z