Copying and Creating Package Repositories in Oracle® Solaris 11.3

Part No: E54747
September 2018

ORACLE®

Copying and Creating Package Repositories in Oracle Solaris 11.3

Part No: E54747

Référence: E54747

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Examples

Using This Documentation

- **Overview** – Describes how to create, copy, access, update, and maintain a software package repository by using the Oracle Solaris 11.3 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

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E53394-01.

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 package repository
- Privileges required for some package repository creation and maintenance tasks

Local IPS Repositories

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

- **Performance and security.** You do not want your 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

The following is an example of the recommended way to use local IPS package repositories:

1. Create a master repository under `/var/share/pkg/repositories`.
2. Duplicate the master repository on other systems as necessary, depending on client demand and distance between client locations.
3. Configure all of these repository locations as publisher origins on every system. IPS automatically selects the repository location with the best connection for each client.
4. Update the master repository monthly to ensure that security fixes are available to client systems. Update the clone repositories monthly from the master repository.

Follow the best practices described below to maintain repository availability and minimize system update errors.

Do Not Subset Repository Content

Include all content from your chosen Oracle Solaris repository source.

Your Oracle Solaris repository source can be any of the following:

- The Oracle Solaris release repository: `http://pkg.oracle.com/solaris/release/`
- The Oracle Solaris support repository: `https://pkg.oracle.com/solaris/support/`
- An Oracle Solaris package repository file

Repository files for Oracle Solaris GA releases and Support Repository Update (SRU) releases are available from various Oracle download sites as described in "How to Copy a Repository From a `zip` File" on page 29.

Do not attempt to select a subset of packages to include in your local repository.

- **Dependencies.** Packages that belong together do not necessarily have the same version numbers or any other common identifier that would enable you to correctly select the required set.
- **Architecture.** Each package includes all content for both SPARC and x86 architectures. Creating a SPARC-only or x86-only repository is not possible.

Do not remove from your repository packages that are delivered by an Oracle publisher. Oracle publishers include, for example, `solaris`, `solarisstudio`, and `ha-cluster`.

Using a subset repository is not a viable way to accomplish any of the following:

- **Limit the software that users can install.** To install or update to a release that is older than the latest release available in the repository, or to otherwise control which software users can install, use one of the solutions described in "Updating to a Version Older Than the Newest Version Allowed" in *Adding and Updating Software in Oracle Solaris 11.3*. Solutions described in that documentation include specifying a version constraint on the command line and using a constraint package.
- **Save disk space.** To minimize the amount of software installed on a system, install the `solaris-minimal-server` package in your initial system installation instead of the `solaris-large-server` package, and then add other packages that you want. You could create a custom package with just this set of packages as `group` dependencies. Other packages that are dependencies of the named packages are installed automatically. For information about dependencies and instructions for creating packages, see *Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.3*.

  To minimize the space required for a package repository, use the guidelines in "Minimal Required Repository" on page 15 to periodically replace your repository, rather than adding packages with every repository update. See the remove and replace instructions in Step 3 in "How to Update a Local IPS Package Repository" on page 56 to update your package repository by replacing the repository rather than simply adding to the repository.
- **Reduce package search and installation time.** Use one of the following strategies to reduce package access time:
  - Install multiple identical package repositories and configure all of these locations as publisher origins on every system. IPS automatically selects the repository location with the best connection for each client. See "Maintaining Multiple Identical Local Repositories" on page 60.
  - Use the `pkg/depot` scalable repository server described in "How to Enable Users to Retrieve Packages Using an HTTP Interface" on page 43.

  See also "Provide Security and High Availability" on page 18 below.

Minimal Required Repository

The Oracle Solaris package repository for each GA release (Oracle Solaris 11 11/11, Oracle Solaris 11.1, Oracle Solaris 11.2, and Oracle Solaris 11.3) is a complete set of packages for that GA release. The Oracle Solaris package repository for each SRU (for example, Oracle Solaris 11.3 SRU 19) includes only packages for that SRU. See also "Oracle Solaris Repository Content" on page 19.

In general, you do not need to add SRU content to your local repository if you add the content of a later GA release to your local repository. The exception to this general rule is if you are updating from or to that SRU release.

The minimal package repository required to support upgrades to new Oracle Solaris releases is the following content for each system that the repository must support:

- The repository content for the currently installed GA release
- The repository content for the currently installed SRU, if any
- The repository content for the GA release to which you want to update
- The repository content for the SRU to which you want to update, if any
- If the currently installed GA release is more than one release behind the release to which you want to update, then your local repository must also contain the repository content for each intervening GA release.
- If the currently installed release is Oracle Solaris 11 11/11 with no SRU or an SRU lower than SRU10, then your local repository must also contain the repository content for Oracle Solaris 11 11/11 SRU10.

For example, to upgrade from Oracle Solaris 11.3 SRU16 to Oracle Solaris 11.3 SRU19, your local IPS package repository must include the following repository content:

- Oracle Solaris 11.3
- Oracle Solaris 11.3 SRU16
- Oracle Solaris 11.3 SRU19

To upgrade from Oracle Solaris 11 11/11 SRU7.5 to Oracle Solaris 11.3 SRU19, your local IPS package repository must include the following repository content:

- Oracle Solaris 11 11/11
- Oracle Solaris 11 11/11 SRU7.5
- Oracle Solaris 11 11/11 SRU10.5
- Oracle Solaris 11.1
- Oracle Solaris 11.2
- Oracle Solaris 11.3
- Oracle Solaris 11.3 SRU19

Repository files for Oracle Solaris GA releases and Support Repository Update (SRU) releases are available from various Oracle download sites as described in "How to Copy a Repository From a `zip` File" on page 29.

Your local repository must contain complete content of the GA and SRU repositories for currently installed software because content from those repositories that is not currently installed might be needed to resolve dependencies for an update.

For more information about keeping your repository as small as possible see the remove and replace instructions in Step 3 in "How to Update a Local IPS Package Repository" on page 56.

Most Complete Repository

If space is not a problem, you can use the `svc:/application/pkg/mirror` Service Management Facility (SMF) service to automatically update your local package repository from the Oracle Solaris support repository. See "How to Automatically Copy a Repository From the Internet" on page 36 for instructions.

Create Repositories in a Separate ZFS File System in a Shared, Persistent Location

An example of a recommended repository location is in a separate ZFS file system in `/var/share/pkg/repositories`.

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. Set the `compression` property to reduce disk space required.
- 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 36.
- Create snapshots.

Create repositories in a shared location.

A shared location is a location that is not in any bootable environment (BE). Everything in `rpool/ROOT` or *pool*`/ROOT` is copied to each new BE and potentially customized in each BE; it is not shared by all BEs. Examples of shared locations include `/var/share` and `/export`. Creating a repository in a shared location provides the following benefits:

- The repository is available from any existing BE. When you boot to a different BE, including an older BE, the same package repository is still available.
- When you create a new BE through upgrading or by cloning an existing BE, you do not waste space by creating multiple copies of the 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 repositories in a persistent location.

Package update or removal usually requires access to a repository that contains the currently installed version of the package. Do not set a publisher origin to a repository in `/tmp` or other nonpersistent location. Move that content to a persistent, shared location. If you install packages from package archive files (`.p5p` files), save those package archive files in a persistent, shared location.

Verify and Snapshot the Repository

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.

Snapshot every time you update the repository.

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

- Use a previously saved snapshot to roll back to a previous version of the repository.
- Minimize user disruption when updating the repository by updating a clone and then replacing the repository with the updated clone.

Provide Security and High Availability

To secure your local repositories, see “Configuring HTTPS Repository Access” on page 86 for instructions.

To provide high availability, use one or more of the following strategies:

- Maintain repository clones in different locations and configure all of these repository locations as publisher origins on every system. IPS automatically selects the repository location with the best connection for each client. See "Maintaining Multiple Identical Local Repositories" on page 60 for instructions.
- To better serve more clients simultaneously, configure repository service properties as described in "How to Enable Users to Retrieve Packages Using an HTTP Interface" on page 43.
- Configure your web server for caching, load balancing, and serving multiple repositories. See Chapter 5, "Running the Package Depot Server Behind a Web Server" for information.

Oracle Solaris Repository Content

Oracle provides the following IPS package repositories for Oracle Solaris. See also "Comparing Repository Content Sources" on page 25.

- The Oracle Solaris release repository is publicly available at `http://pkg.oracle.com/solaris/release/`. The release repository is updated at every Oracle Solaris GA release. For example, the release repository was updated for Oracle Solaris 11.1, Oracle Solaris 11.2, and Oracle Solaris 11.3. Some FOSS updates are available in the release repository between Oracle Solaris GA releases. See FOSS in Oracle Solaris for more information.

  Previous Oracle Solaris GA release repositories are available as file downloads from My Oracle Support (MOS) and other sites as described in "How to Copy a Repository From a `zip` File" on page 29. For example, on MOS see Where to download Oracle Solaris ISO images and Update Releases (Doc ID 1277964.1) for GA repositories. The first link under each release section is a link to download the IPS repository file for that release.
- The Oracle Solaris support repository is available at `https://pkg.oracle.com/solaris/support/`. To access the support repository, follow the instructions in "How to Install a Certificate and Key for the Oracle Solaris Support Repository" on page 26. The support repository includes the content of the release repository plus fixes, including security fixes, that were made after the release repository was released.
- Support Repository Updates (SRUs) are integrated into the Oracle Solaris support repository and are also available as separate repository updates that you can download from MOS or other sites as described in "How to Copy a Repository From a `zip` File" on page 29.

Updates to the support repository are released approximately monthly. Every third SRU is a CPU (Critical Patch Update) SRU with content typically more focused on security fixes. See Critical Patch Updates, Security Alerts and Third Party Bulletin for information about CPUs. SRU repositories include only packages for that SRU. While each SRU includes all fixes and

enhancements that were delivered by previously released SRUs, an SRU does not contain any other SRUs. An SRU contains only one version of `pkg:/entire`. For best system upgrade results, integrate SRUs into your local Oracle Solaris repository monthly, either from the Oracle Solaris support repository URI or from the downloaded SRU repository files.

- If you update your local repository from the Oracle Solaris support repository URI (as described in "Copying a Repository From the Internet" on page 34), then you receive all packages for each update. If you failed to apply an update one month but updated in a subsequent month, the missed update would still be available in your updated repository. For example, if you updated in March but not in February, you would still be able to update systems to the February update level by using one of the solutions described in "Updating to a Version Older Than the Newest Version Allowed" in *Adding and Updating Software in Oracle Solaris 11.3*.
- If you update your local repository from downloaded SRU files, you can only update systems to those SRU levels. For example, if you applied the January and March SRU repository updates but did not apply the February update to your local repository, you would not be able to update systems to the February SRU level. You could add the February SRU repository content to your local repository, even though you already added the March content, and then you could update systems to the February SRU level. If your repository does not contain the version of `pkg:/entire` for a particular SRU update, then you cannot update to that SRU level.

System Requirements

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

Operating system

A system that hosts an IPS package repository must be running at least Oracle Solaris 11 11/11. Newer versions of Oracle Solaris 11 provide additional repository creation and maintenance features, in addition to other new Oracle Solaris features and security fixes.

- The `pkgrepo verify` and `pkgrepo fix` commands are available only with Oracle Solaris 11.1.7 and later.
- The `pkgrepo contents` and `pkgrepo remove-publisher` commands are available only with Oracle Solaris 11.2 and later.
- The `pkgrecv -m all-timestamps` behavior is the default with Oracle Solaris 11.2 and later.
- The `pkgrecv --clone` option is available only with Oracle Solaris 11.2 and later.
- The `svc:/application/pkg/mirror` SMF service is available only with Oracle Solaris 11.2 and later.

- The scalable repository server, `pkg.depotd`, and the `pkg.depot-config` command are available only with Oracle Solaris 11.2 and later.

Disk space

The amount of disk space required depends on how complete your repository is. See "Comparing Repository Content Sources" on page 25.

- The repository file for the release creates a 9.5GB repository.
- Copying the Oracle Solaris support repository creates a 95GB repository.

Because best practice is to keep local repositories updated with all SRUs since the last Oracle Solaris dot release that you added, plan to use 10-15 GB of additional space each year for SRUs.

Additional software, such as Oracle Developer Studio or Oracle Solaris Cluster, 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 11.3* for more information about roles and profiles, including how to determine which role or profile you need.

Roles

Use the `roles` command to list the roles that are assigned to you. Use the `su` command with the name of the role to assume that role. As this role, you can execute any commands that are permitted by the rights profiles that are assigned to that role.

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 to create new ZFS file systems.

Software Installation

This rights profile enables you to run the `pkg` command to install and update packages, configure publishers and images, and manage boot environments.

Service Management

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

Use one of the following methods to execute commands that your rights profiles permit you to execute:

- Use a profile shell such as `pfbash` or `pfksh`.
- Use the `pfexec` command in front of the command that you want to execute. In general, you must specify the `pfexec` command with each privileged command that you execute.

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 an Oracle Solaris IPS package repository:

- Use repository files from media or from an Oracle Solaris download site.
- 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 the following information:

- Performance and troubleshooting information related to copying repositories.
- A description of the content available from different repository sources.

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 34 to retrieve repository content from an Internet location, consider the following configuration to improve your transfer performance:

- Ensure that your ZFS storage pool capacity (percentage of pool space used) is less than 90%. 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. The `--clone` option makes the destination repository an exact copy of the

source repository, which is significantly faster than the default `pkgrecv` copy mode. See "How to Explicitly Copy a Repository From the Internet" on page 34 and "How to Clone a Local IPS Package Repository" on page 61.

- On the ZFS file system that hosts the repository, 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.

  Set `compression` to `on` or `lzjb`. The `lzjb` compression algorithm is optimized for performance while providing good data compression. For better compression, set `compression` to `lz4`. The `lz4` compression algorithm provides better compression than `lzjb` with lower CPU overhead.

  Note that setting `compression` only affects the package manifest data; the package file data is already compressed. To check the compression, check the `compressratio` value for the dataset. See the `zfs`(1M) man page for more information.

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 29.
- Do not use multiple instances of the `pkgrecv` command simultaneously to republish packages to the same destination repository or archive.

  Be careful not to modify a repository that the `svc:/application/pkg/mirror:default` service is modifying. By default, the `pkg/mirror` service creates and updates repositories at `/var/share/pkg/repositories`. See "How to Automatically Copy a Repository From the Internet" on page 36 for more information about the `pkg/mirror` service.
- 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 and the path to the repository have execute permission.

  These permissions problems prevent users from accessing these repositories.

The `pkgrepo verify` command also checks file checksums, package signatures, and manifest dependencies: The `pkgrepo verify` command verifies that every file referenced in each package manifest is in the repository. If the `pkgrepo verify` command reports errors other than permissions errors, try using the `pkgrepo fix` command to fix the errors. See the `pkgrepo`(1) man page for more information.

- After updating the configuration or content of a repository, restart any `pkg/server` service instance that is managing that repository so that the changes will be reflected in the search responses and browser interface for that repository.
- 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 42. See *Adding and Updating Software in Oracle Solaris 11.3* 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 80. 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.

If you are using Oracle Enterprise Manager Ops Center, the IPS package repository is called the Oracle Solaris Software Update Library. See the Oracle Solaris sections of the Software Library section of the Oracle Enterprise Manager Ops Center configuration reference manual for information about how to update the library. Use the Configure Parent Repos and Add Content actions in the Library > Oracle Solaris section of the Ops Center BUI (browser user interface) to associate new certificates with the `solaris` publisher. See the Firewall Rules section of the Oracle Enterprise Manager Ops Center Ports and Protocols guide for firewall rules to allow systems to access the external IPS repository. This list can also be useful if you are not using Ops Center.

Comparing Repository Content Sources

The following table offers information to help you decide whether to create and update your repository from the Oracle Solaris support repository or from repository files.

TABLE 1 Repository Content Source Comparison

Repository Source	Support Repository	Repository Files
Description	The support repository includes every package from every Oracle Solaris release starting from Oracle Solaris 11 11/11.	The repository `zip` files that are delivered with each release include packages from the current release plus all packages required to update to that release from any previous release starting from Oracle Solaris 11 11/11.
Installing and Updating Systems	Systems can install or update to any release and to any SRU of any release.	You can update systems to an SRU for the current release only if you added the content of the repository file for that SRU to your local repository. For example, if you failed to add the April SRU repository content and then you added the May SRU repository content, you would have all the fixes and enhancements from the May, April, and earlier SRUs, but you would not be able to update systems to the April SRU level.
Time and Space	The repository takes longer to create and requires more disk space.	The repository requires less time to create and consumes less disk space.

How to Install a Certificate and Key for the Oracle Solaris Support Repository

If you use the Oracle Solaris support repository, you must install a certificate and key. You cannot add the support repository origin to your `solaris` publisher unless you specify a valid certificate and key.

1. **Get certificate and key files.**

   a. **Open the Oracle Certificate Request site.**

      In your browser, navigate to `https://pkg-register.oracle.com/`.

   b. **Sign in to the site.**

      i. **Select the Request Certificates link.**

      ii. **Sign in using your My Oracle Account support credentials.**

   c. **Find the Oracle Solaris 11 Support repository, and select the Request Access button.**

   d. **Read the license and select the Accept button.**

e. Open the Certificate Information page.

On the Product Details page, select the "certificate page" link or navigate to `https://pkg-register.oracle.com/register/certificate/`.

f. Select the Download Certificate button, and save the certificate file on your system.

g. Select the Download Key button, and save the key file on your system.

2. Configure the `solaris` publisher using the new certificate and key files.

The arguments to the `-c` and `-k` options are the certificate and key files that you downloaded in the previous step, as shown in the following example:

```
$ pkg set-publisher -g https://pkg.oracle.com/solaris/support/ \
> -c ~/Downloads/pkg.oracle.com.certificate.pem \
> -k ~/Downloads/pkg.oracle.com.key.pem  solaris
```

Use the following command to show that the certificate and key are installed:

```
$ pkg publisher solaris
            Publisher: solaris
                Alias:
           Origin URI: https://pkg.oracle.com/solaris/support/
              SSL Key: /var/pkg/ssl/keyfile
             SSL Cert: /var/pkg/ssl/certfile
 Cert. Effective Date: July  1, 2015 04:47:13 PM
Cert. Expiration Date: July  8, 2017 04:47:13 PM
          Client UUID: client-uuid
      Catalog Updated: January 18, 2016 06:59:40 PM
              Enabled: Yes
           Properties:
                        proxied-urls = []
                        signature-policy = require-signatures
```

Next Steps See also "SSL Certificate Problem" in *Adding and Updating Software in Oracle Solaris 11.3*.

This process is similar for repositories of other publishers such as `solarisstudio` or `ha-cluster`.

▼ How to Update the Certificate and Key for the Oracle Solaris Support Repository

Use this procedure if you receive a message such as the following when you perform `pkg` commands:

```
Certificate '/var/pkg/ssl/certfile' for publisher 'solaris', needed to access
'https://pkg.oracle.com/solaris/support/', will expire in '7' days.
```

1. **Get new certificate and key files.**

   a. **Open the Oracle Certificate Request site.**

      In your browser, navigate to `https://pkg-register.oracle.com/register/certificate/`.

   b. **Sign in using your My Oracle Account support credentials.**

   c. **For Oracle Solaris 11.3, select the Download Certificate button, and save the certificate file on your system.**

   d. **For Oracle Solaris 11.3, select the Download Key button, and save the key file on your system.**

2. **Configure the `solaris` publisher using the new certificate and key files.**

   Because the support repository is already configured with the old key and certificate, you do not need to specify the origin to update the key and certificate.

   ```
   $ pkg set-publisher -c ~/Downloads/pkg.oracle.com.certificate.pem \
   > -k ~/Downloads/pkg.oracle.com.key.pem solaris
   ```

   The `pkg publisher solaris` command shows the updated “Cert. Effective Date” and “Cert. Expiration Date.”

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. Consider setting the `compression` property. See "Best Practices for Creating and Using Local IPS Package Repositories" on page 14.

```
$ zfs create -o atime=off rpool/VARSHARE/pkgrepos
$ zfs create rpool/VARSHARE/pkgrepos/solaris
$ zfs get atime rpool/VARSHARE/pkgrepos/solaris
NAME                              PROPERTY  VALUE  SOURCE
rpool/VARSHARE/pkgrepos/solaris   atime     off    inherited from rpool/VARSHARE/pkgrepos
```

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

   Download the Oracle Solaris IPS package repository files (`*repo*.zip`) from the same location where you downloaded the system installation image. In addition to the repository files, download the `install-repo.ksh` script and the README and checksum `.txt` files.

```
$ ls
install-repo.ksh           sol-11_3-ga-repo-3of4.zip
README-zipped-repo.txt     sol-11_3-ga-repo-4of4.zip
sol-11_3-ga-repo-1of4.zip  sol-11_3-ga-repo.txt
sol-11_3-ga-repo-2of4.zip
```

   On the Oracle Technology Network (OTN) site, you can download the `install-repo.ksh` script and the README and digest `.txt` files directly. On the My Oracle Support (MOS) and the Oracle Software Delivery Cloud (OSDC) sites, the `install-repo.ksh` script and the README and checksum `.txt` files are part of the Repository Installation Guide. For example, in the IPS Repository column of an SRU Index on MOS, select the Installation Guide document. A new page displays that contains the following buttons:

   - Download. Select this button to retrieve the Repository Installation Guide `.zip` file. The Repository Installation Guide file contains the following files:
     - The `install-repo.ksh` script.
     - The `README-zipped-repo.txt` README file that explains how to use the `install-repo.ksh` script.
     - The `digest.txt` checksums file for the repository files (`*repo*.zip`).
     - Text and HTML versions of a README file that describes this particular SRU.
   - View Readme. Select this button to display the README file that describes this particular SRU.
   - View Digest. Select this button to pop up a new window that displays the SHA-1 and MD5 checksums for the Repository Installation Guide `.zip` file.

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 file (`*repo*.zip`) into the specified directory.

The `install-repo.ksh` script optionally performs the following additional tasks:

- Verifies checksums of the repository 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 `digest.txt` file:

```
$ digest -v -a sha256 *repo*.zip
```

- Adds the repository content to existing content if the specified destination already contains a repository.
- Verifies 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.
- Creates 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 the specified destination directory (`-d`).

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/VARSHARE/pkgrepos/solaris@sol-11_3_0
```

Example 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 `*repo*.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 `*repo*.zip` files. Default: The newest image available in the source directory.

`-c` Optional. Compares the checksums of the `*repo*.zip` files with the checksums in the specified file. If you specify `-c` with no argument, the default file used is the `digest.txt` 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 /var/share/pkgrepos/solaris -c -v -I
Comparing digests of downloaded files...done. Digests match.
Uncompressing sol-11_3-ga-repo-1of4.zip...done.
Uncompressing sol-11_3-ga-repo-2of4.zip...done.
Uncompressing sol-11_3-ga-repo-3of4.zip...done.
Uncompressing sol-11_3-ga-repo-4of4.zip...done.
Repository can be found in /var/share/pkgrepos/solaris.
Initiating repository verification.
Building ISO image...done.
ISO image can be found at:
/tank/downloads/sol-11_3-ga-repo.iso
Instructions for using the ISO image can be found at:
/var/share/pkgrepos/solaris/README-repo-iso.txt
$ ls /var/share/pkgrepos/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.

If you receive a message that the repository verification could not be done, ensure that Oracle Solaris 11.1.7 or later is installed.

Example 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 /var/share/pkgrepos/solaris info
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4764     online           2014-03-18T05:30:57.221021Z
$ ./install-repo.ksh -d /var/share/pkgrepos/solaris -c -v -I
IPS repository exists at destination /var/share/pkgrepos/solaris
Current version: 0.175.2.0.0.35.0
Do you want to add to this repository? (y/n) y
Comparing digests of downloaded files...done. Digests match.
Uncompressing sol-11_3-ga-repo-1of4.zip...done.
Uncompressing sol-11_3-ga-repo-2of4.zip...done.
Uncompressing sol-11_3-ga-repo-3of4.zip...done.
Uncompressing sol-11_3-ga-repo-4of4.zip...done.
Repository can be found in /var/share/pkgrepos/solaris.
Initiating repository rebuild.
Initiating repository verification.
Building ISO image...done.
ISO image can be found at:
/tank/downloads/sol-11_3-ga-repo.iso
Instructions for using the ISO image can be found at:
/var/share/pkgrepos/solaris/README-repo-iso.txt
$ pkgrepo -s /var/share/pkgrepos/solaris info
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4768     online           2016-06-02T18:11:55.640930Z
```

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. Consider setting the compression property. See “Best Practices for Creating and Using Local IPS Package Repositories” on page 14.

   ```
   $ zfs create -o atime=off rpool/VARSHARE/pkgrepos
   $ zfs create rpool/VARSHARE/pkgrepos/solaris
   $ zfs get atime rpool/VARSHARE/pkgrepos/solaris
   NAME                              PROPERTY  VALUE  SOURCE
   rpool/VARSHARE/pkgrepos/solaris   atime     off    inherited from rpool/VARSHARE/pkgrepos
   ```

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

Create an .iso file from the repository .zip files using the -I option as described in Example 1, “Creating a New Repository From a zip File,” on page 30.

3. **Mount the image file.**

   Mount the repository .iso file to access the content.

   ```
   $ mount -F hsfs /path/sol-11_3-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/ /var/share/pkgrepos/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 /var/share/pkgrepos/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/VARSHARE/pkgrepos/solaris@sol-11_3_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. Consider setting the `compression` property. See "Best Practices for Creating and Using Local IPS Package Repositories" on page 14.

   ```
   $ zfs create -o atime=off rpool/VARSHARE/pkgrepos
   $ zfs create rpool/VARSHARE/pkgrepos/solaris
   $ zfs get atime rpool/VARSHARE/pkgrepos/solaris
   NAME                            PROPERTY  VALUE  SOURCE
   rpool/VARSHARE/pkgrepos/solaris  atime     off    inherited from rpool/VARSHARE/pkgrepos
   ```

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

   Create the required IPS 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 /var/share/pkgrepos/solaris
   ```

3. **Set the default publisher.**

   ```
   $ pkgrepo -s /var/share/pkgrepos/solaris get publisher/prefix
   SECTION    PROPERTY                  VALUE
   publisher  prefix                    ""
   $ pkgrepo -s /var/share/pkgrepos/solaris set publisher/prefix=solaris
   ```

```
$ pkgrepo -s /var/share/pkgrepos/solaris get publisher/prefix
SECTION    PROPERTY                       VALUE
publisher  prefix                         solaris
```

4. **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 23. 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 /var/share/pkgrepos/solaris \
   --clone -p '*' --key /path-to/key-file --cert /path-to/certificate-file '*'
   ```

   The `--clone` option makes the destination repository an exact copy of the source repository, which is significantly faster than the default `pkgrecv` copy mode. See the `pkgrecv(1)` man page for information about the `--clone` option.

   You should not use the `-m latest` option for this purpose. Do not try to specify a subset of Oracle Solaris packages to receive. Using a repository that is too sparse can result in errors when users attempt to update their images. See "Do Not Subset Repository Content" on page 14.

   If the `pkgrecv` operation is interrupted, follow the instructions in "Resuming an Interrupted Package Receive" on page 60.

5. **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.

6. **Snapshot the new repository.**

   ```
   $ zfs snapshot rpool/VARSHARE/pkgrepos/solaris@sol-11_3_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 14. 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_3_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 11.3* for more information about SMF commands.

♦ ♦ ♦ CHAPTER 3

Providing Access To Your Repository

This chapter describes how to enable systems to retrieve packages in your local repository by using either of the following methods:

- A file interface
- An HTTP interface

One repository can be configured for both types of access.

This chapter also shows how to use an Oracle Solaris 10 or Linux system to serve Oracle Solaris 11 package repository content.

Comparing Repository Access Interfaces

The following table offers information to help you decide how to provide access to your IPS package repository. A single repository can be configured for both file access and HTTP or HTTPS access.

TABLE 2 Repository Access Interfaces Comparison

Repository Access Types	File	HTTP	HTTPS
Advantages	Easy access for local users.	Easy access for multiple systems without using NFS.	Restricted system access. Secure transfer of content between repository and installable image. Configure with or without client authentication.
Disadvantages	Requires NFS. Less flexibility to configure access controls.	Not secure. Difficult to restrict systems from connecting.	More complex to configure, particularly if you configure client authentication.

Enabling Users to Retrieve Packages Using a File Interface

This section describes how to serve 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 systems to access the local repository by using NFS, create and publish an NFS share.

   ```
   $ zfs share -o share.nfs=on rpool/VARSHARE/pkgrepos%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
     /var/share/pkgrepos     ipsrepo nfs     sec=sys,rw
     ```

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

     ```
     $ dfshares
     RESOURCE                                  SERVER ACCESS  TRANSPORT
        solaris:/var/share/pkgrepos           solaris  -       -
     ```

3. **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 /var/share/pkgrepos/solaris/ solaris
   ```

   See "Configuring Publishers" in *Adding and Updating Software in Oracle Solaris 11.3* for more information about configuring publishers.

Enabling Users to Retrieve Packages Using an HTTP Interface

This section describes how to configure an HTTP interface for a package repository by configuring a `svc:/application/pkg/depot` SMF service instance. The `pkg/depot` instance runs the Apache instance.

TABLE 3 HTTP Repository Access Configuration Methods Comparison

Configuration Method	`pkg/server` service	`pkg/depot` service	Manual Apache configuration
Advantages	Requires the least amount of configuration. See "How to Configure a `pkg/server` Service Instance" on page 46.	Provides a scalable package repository server for simultaneous use by a large number of clients. See "How to Enable Users to Retrieve Packages Using an HTTP Interface" on page 43.	Provides scalability and more fine-grained control such as caching, load balancing, multiple repositories behind one domain, and HTTPS access. See Chapter 5, "Running the Package Depot Server Behind a Web Server".
Disadvantages	Does not scale well. This method is most suitable for package development with access by a small number of clients.	Requires a small amount of additional configuration after first configuring a `pkg/server` instance.	This method is more complex to configure.

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

To enable systems to access the local repository by using HTTP or HTTPS, this procedure shows how to configure a `svc:/application/pkg/depot` SMF service instance. The `pkg/depot` instance provides a scalable package repository server.

Before You Begin To get the `svc:/application/pkg/depot` SMF service and the `pkg.depot-config(1M)` man page, make sure the `pkg:/package/pkg/depot` package is installed.

1. **Create a `pkg/server` instance.**

   See "How to Configure a `pkg/server` Service Instance" on page 46. The `svc:/application/pkg/server` instance for each repository provides the file system location of the repository (in the `pkg/inst_root` property). Each `pkg/server` instance serves a single local repository. A single `pkg/depot` instance can serve multiple repositories.

By default, repository content is served by an instance of the `pkg.depotd` package depot server. If you run `svcs -p` on the `pkg/server` instance, you see a `pkg.depotd` process associated with this `pkg/server` instance:

```
$ svcs -p pkg/server:solaris
STATE          STIME    FMRI
online         Jun_26   svc:/application/pkg/server:solaris
               Jun_26       16129 pkg.depotd
```

Running a `pkg.depotd` process under a `pkg/server` instance works well when a small number of clients need to access the repository simultaneously. For greater scalability, configure the repository to be served directly by the Apache depot server by associating a `pkg/depot` instance, as described in the following steps.

2. **Ensure that the value of the `readonly` property is `true`.**

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

3. **Set the `standalone` property.**

   To configure the repository to be served directly by the Apache depot server, set the value of the `pkg/standalone` property for the appropriate `pkg/server` service instance to `false`.

   - If the `pkg/standalone` property of a particular `pkg/server` instance is set to `true`, then the repository is served by the `pkg.depotd` process.
   - If the `pkg/standalone` property of a particular `pkg/server` instance is set to `false` and the `pkg/readonly` property is set to `true`, then the repository is served by the `pkg/depot:default` instance, which runs the Apache instance.

   Each `pkg/server` instance either runs `pkg.depotd` or instructs the `pkg/depot` service to serve this repository and provides the file system location of the repository.

   Use the `-c` option with the `svcprop` command because you have not yet refreshed the service:

   ```
   $ svccfg -s pkg/server:solaris setprop pkg/standalone=false
   $ svcprop -c -p pkg/standalone pkg/server:solaris
   pkg/standalone boolean false
   ```

   After you restart this `pkg/server` service instance, you no longer see processes associated with this `pkg/server` instance. If you run `svcs -p` on the `pkg/depot` service, you see Apache worker processes associated with the `pkg/depot` service:

   ```
   $ svcs -p pkg/depot pkg/server:solaris
   STATE          STIME    FMRI
   online         12:26:38 svc:/application/pkg/server:default
   online         12:26:40 svc:/application/pkg/depot:default
   ```

```
12:26:39     6266 httpd.worker
12:26:39     6269 httpd.worker
12:26:39     6270 httpd.worker
12:26:39     6272 httpd.worker
12:26:39     6273 httpd.worker
```

4. **Set the host name.**

   Set the value of the `config/host` property of the `pkg/depot:default` instance to the IP address or host name of the system that hosts the repository file system.

   ```
   $ svccfg -s pkg/depot:default setprop config/host=pkg.example.com
   ```

   If you need to access this repository URI from other systems, specify the host name to use as the argument to the Apache `ServerName` directive. See `ServerName` directive examples in "How to Configure Secure Repositories" on page 95 and "Adding SSL Configuration to the Apache Configuration File" on page 91.

5. **(Optional) Set other properties.**

   For descriptions of `pkg/depot` properties, see the Options section of the `pkg.depot-config(1M)` man page.

   The following command lists the depot configuration properties and their values:

   ```
   $ svcprop -p config pkg/depot
   ```

   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/depot:default editprop
   ```

6. **Start the `pkg/depot` instance.**

   - **If the `pkg/depot` instance is `disabled`, refresh and enable the instance:**

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

     If the `pkg/depot` instance is not `online`, use the `svcs -xvL pkg/depot` command to get more information to diagnose and fix the problem.

   - **If the `pkg/depot` instance is `online`, refresh and restart the instance:**

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

7. **Restart the `pkg/server` instance.**

   The `pkg/server` instance should already be `online`.

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

See Also

- The `pkg.depot-config(1M)` man page provides more information about web server configuration for `pkg` package repositories.
- "Serving Multiple Repositories Using Web Server Access" on page 64 describes how to serve multiple repositories from multiple locations or from a single location.
- "Multiple Repositories Under One Domain" on page 84 describes how to run multiple repositories under one domain name with different prefixes.
- Chapter 5, "Running the Package Depot Server Behind a Web Server" describes how to configure your web server for caching, load balancing, and serving multiple repositories.
- "Configuring HTTPS Repository Access" on page 86 describes how to configure secure repository access.

▼ How to Configure a `pkg/server` Service Instance

You always need to configure a `svc:/application/pkg/server` SMF service instance. A `pkg/server` instance is required to provide the file system location of the repository to either a `pkg/depot` instance or a manual Apache configuration.

1. **Create a `pkg/server` depot server instance.**

   You could configure the `default` instance of the `pkg/server` service. This procedure shows how to create and configure a new 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=/var/share/pkgrepos/solaris
   ```

3. **(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
```

Note - When the value of the pkg/standalone property is false and the value of the pkg/readonly property is true (see "How to Enable Users to Retrieve Packages Using an HTTP Interface" on page 43), then only the pkg/inst_root property of the pkg/server instance is used. All other configuration comes from properties on the pkg/depot service.

The following are examples of other properties that you could set:

- **Set the port property.**

  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.

- **Increase the threads property value to increase scalability for a small number of clients.**

  To increase concurrent client access for a repository served by the pkg.depotd depot server, increase the value of the pkg/threads property.

  The default value of the pkg/threads property is 60. The pkg/threads value is limited by available memory. Allow for 20 threads used per concurrent client. Scaling efficiency peaks at about 20 clients.

  ```
  $ svccfg -s pkg/server:solaris setprop pkg/threads=400
  ```

4. **Start the repository service.**

   Refresh and enable the pkg/server instance.

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

   If the pkg/server:solaris instance is not online, use the svcs -xvL pkg/server:solaris command to get more information to diagnose and fix the problem.

5. **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.

6. **Set the publisher origin.**

   Set the publisher origin to the pkg/port location:

   ```
   $ pkg set-publisher -g http://localhost:81/ solaris
   ```

   See "Configuring Publishers" in *Adding and Updating Software in Oracle Solaris 11.3* for more information about configuring publishers.

Next Steps

Note - A pkg/server instance for repository access is suitable for use by a small number of clients. For simultaneous access by a large number of clients, implement a pkg/depot instance. A pkg/depot instance provides much greater scalability than a pkg/server instance. See "How to Enable Users to Retrieve Packages Using an HTTP Interface" on page 43.

Updating Repositories with HTTP Locations

You cannot add packages to or remove packages from a repository by using the http location of the repository if the value of the pkg/readonly property is true. Do not change the value of the pkg/readonly property. Instead, to update a repository that has an http location configured, use the file system location for that repository. The file system location is the value of the pkg/inst_root property of the pkg/server service instance for this repository.

The following commands show that the repository at /var/share/pkgrepos/solaris can be accessed at pkg.example.com:80:

```
$ svcprop -t -p pkg/inst_root -p pkg/port -p pkg/readonly -p pkg/standalone \
pkg/server:solaris
pkg/standalone boolean false
pkg/readonly boolean true
pkg/port count 80
pkg/inst_root astring /var/share/pkgrepos/solaris
$ svcprop -p config/host pkg/depot:default
pkg.example.com
```

The following commands show that you can get information about the repository by using the http location. The solaris in the -s and -g arguments is the name of the pkg/server service instance: pkg/server:solaris. If the value of pkg/standalone were true, the value of the -s and -g arguments would be simply http://pkg.example.com/.

```
$ pkgrepo -s http://pkg.example.com/solaris info
PUBLISHER PACKAGES STATUS           UPDATED
solaris   1404     online           2017-06-30T19:01:49.247256Z
$ pkgrepo -s http://pkg.example.com/solaris list entire
```

```
PUBLISHER NAME                                            O VERSION
solaris   entire                                            0.5.11,5.11-0.175.3.18.0.6.0:
20170322T192451Z
$ pkg list -g http://pkg.example.com/solaris entire
NAME (PUBLISHER)                                    VERSION                    IFO
entire                                              0.5.11-0.175.3.18.0.6.0    ---
```

The following commands show that you cannot add packages to or remove packages from the repository by using the `http` location, but you can update the repository by using the underlying file system location. Best practice is to always leave `pkg/readonly` set to `true` and use the repository file system location to update the repository.

```
$ pkgrecv -s testrepo -d http://pkg.example.com/solaris testpkg
Processing packages for publisher solaris ...
Retrieving and evaluating 1 package(s)...
PROCESS                                       ITEMS    GET (MB)   SEND (MB)
testpkg                                         0/1     0.0/0.0     0.0/0.0
pkgrecv: 'open' failed for transaction ID 'id': Publisher 'solaris' has no
repositories that support the 'open/0' operation.
$ pkgrecv -s testrepo -d /var/share/pkgrepos/solaris testpkg
Processing packages for publisher solaris ...
Retrieving and evaluating 1 package(s)...
PROCESS                                       ITEMS    GET (MB)   SEND (MB)
Completed                                       1/1     0.0/0.0     0.0/0.0
$ pkg list -g http://pkg.example.com/solaris testpkg
NAME (PUBLISHER)                                    VERSION                    IFO
testpkg                                             1.0                        ---
$ pkgrepo -s http://pkg.example.com/solaris remove testpkg
pkgrepo remove: Network repositories are not currently supported for this operation.
$ pkgrepo -s /var/share/pkgrepos/solaris remove testpkg
Removing packages for publisher solaris ...
Deleting search index                           Done
Updating catalog                                Done
Analyzing removed packages                       1/1
Analyzing repository packages              1430/1430
Removing package manifests                       1/1
Removing package files                           1/1
$ pkg list -g http://pkg.example.com/solaris testpkg
pkg list: no packages matching 'testpkg' known
$ pkg list -g /var/share/pkgrepos/solaris testpkg
pkg list: no packages matching 'testpkg' known
```

Using an Oracle Solaris 10 or Linux System to Serve Repository Content

In addition to accessing your repository through an Oracle Solaris 11 system, you can access your IPS package repository through an Oracle Solaris 10 system or a Linux system.

The repository must be created on an Oracle Solaris 11 system. The repository can remain on the Oracle Solaris 11 system and be accessed from an Oracle Solaris 10 or Linux system, or you can copy the repository directory to the Oracle Solaris 10 or Linux system.

A file system repository can be hosted on almost any operating system and then shared over NFS, SMB, or any other remote file system mechanism that Oracle Solaris supports.

The following procedure shows how to provide HTTP access to an IPS repository from an Oracle Solaris 10 or Linux system. In addition to Oracle Solaris 10 and Linux repository hosts, you could use this partial configuration as an Apache configuration for Oracle Solaris 11 11/11, Oracle Solaris 11.1, Oracle Solaris 11.2, or Oracle Solaris 11.3 repository hosts. You might want to do this, for example, if you have an existing web server and do not want to run a separate web server instance as part of the `svc:/application/pkg/depot` service.

▼ How to Provide HTTP Access to an IPS Repository From an Oracle Solaris 10 or Linux System

When HTTP access to a repository is provided from a system that is not an Oracle Solaris 11 system, `pkg search` and BUI support are not available. Users can install and update packages and get information about packages through the `pkg list`, `pkg info`, and `pkg contents` commands.

Before You Begin This procedure uses the `pkg.depot-config` command, which is available only in Oracle Solaris 11.2 or later. You must install the `package/pkg/depot` package to get the `pkg.depot-config` command.

1. **Create a partial web server configuration file.**

   Use the `pkg.depot-config` command to create the configuration to enable a web server on the Oracle Solaris 10 or Linux system to serve basic `pkg` installation operations for systems using an existing web service.

   `/usr/lib/pkg.depot-config -F -d` *prefix*=*repository_dir* **`-r`** *runtime_dir*

The *prefix* is a prefix into the `depot-config` web server namespace where the *repository_dir* repository can be accessed. The *runtime_dir* is created if it does not already exist. See the `pkg.depot-config`(1M) man page for more information about how to use the `pkg.depot-config` command.

This command creates the following output in the *runtime_dir* directory, as shown in Example 3, “Serving Package Content From a Linux System Using an HTTP Interface,” on page 51:

- A filed named `depot.conf` that contains rewrite rules.
- An `htdocs` directory.

2. Install the partial web server configuration file.

Copy the partial configuration that is output by the `pkg.depot-config` command to the Apache configuration directory. On an Oracle Solaris 10 repository host, copy the `depot.conf` configuration fragment file to `/etc/apache2/2.2/conf.d`, for example. For a Linux repository host, consult your OS documentation to determine how to use this partial configuration file. The `htdocs` files must be accessible from the `DocumentRoot` of the repository host.

3. Set the publisher origin.

On each Oracle Solaris system that needs to access the repository, set the publisher origin to the `http` address of the repository host system, as shown in the following example. Use `pkg list` or `pkg info` to test your access to the repository.

Example 3 Serving Package Content From a Linux System Using an HTTP Interface

In this example, the repository is in `/var/share/pkgrepos` on an Oracle Solaris 11.3 system. This repository contains one package from the `mypub` publisher.

```
$ pkgrepo -s /var/share/pkgrepos/myrepo info
PUBLISHER PACKAGES STATUS           UPDATED
mypub     1        online           2015-06-08T05:23:00.489687Z
```

The following command creates the configuration.

```
$ /usr/lib/pkg.depot-config -F -d customconfig=/var/share/pkgrepos/myrepo \
-r /tmp/runtime
Created /tmp/runtime/depot.conf
```

The `-F` option produces a partial Apache configuration file that can be installed on the Linux system to enable access to the IPS package repository from the Linux system. This configuration information is only a partial configuration, designed to be included from an existing Apache configuration file. For a complete `httpd.conf` configuration, you need many additional Apache settings that are not included in this partial configuration.

The `-d` option specifies the path to the repository. This repository directory will be included in the depot server configuration. The `customconfig` portion of the argument is used as a prefix into the `depot-config` web server namespace where this repository can be accessed.

The `-r` option specifies the output directory for the configuration files.

The `pkg.depot-config` command created the following configuration files:

```
/tmp/runtime/depot.conf
/tmp/runtime/htdocs
/tmp/runtime/htdocs/customconfig
/tmp/runtime/htdocs/customconfig/mypub
/tmp/runtime/htdocs/customconfig/mypub/publisher
/tmp/runtime/htdocs/customconfig/mypub/publisher/1
/tmp/runtime/htdocs/customconfig/mypub/publisher/1/index.html
/tmp/runtime/htdocs/customconfig/publisher
/tmp/runtime/htdocs/customconfig/publisher/1
/tmp/runtime/htdocs/customconfig/publisher/1/index.html
/tmp/runtime/htdocs/customconfig/status
/tmp/runtime/htdocs/customconfig/status/0
/tmp/runtime/htdocs/customconfig/status/0/index.html
/tmp/runtime/htdocs/versions
/tmp/runtime/htdocs/versions/0
/tmp/runtime/htdocs/versions/0/index.html
```

The `pkg.depot-config` command created the following `/tmp/runtime/depot.conf` file:

```
RewriteEngine on
RewriteLog "/var/log/pkg/depot/rewrite.log"
RewriteLogLevel 0

# Allow these because they are encoded in the package/manifest names
# when looking up v4 repositories.
AllowEncodedSlashes On
# The default of 500 MaxKeepAliveRequests is too low to be useful.
MaxKeepAliveRequests 10000

# Per-repository versions, publishers, and status responses.
RewriteRule ^/customconfig/versions/0 /versions/0/index.html [PT,NE]
RewriteRule ^/customconfig/publisher/0 /customconfig/publisher/1/index.html [PT,NE]
RewriteRule ^/customconfig/status/0 /customconfig/status/0/index.html [PT,NE]

# Rules to handle responses for default publishers.

RewriteRule ^/mypub/manifest/0/.*$ %{THE_REQUEST} [NE,C]
RewriteRule ^GET\ /mypub/manifest/0/([^@]+)@([^\ ]+)(\ HTTP/1.1)$ /customconfig/mypub/
publisher/mypub/pkg/$1/$2 [NE,PT,C]
RewriteRule ^/customconfig/mypub/(.*)$ %{DOCUMENT_ROOT}/customconfig/mypub/$1 [NE,L]
RewriteRule ^/mypub/file/(.*$) /customconfig/mypub/file/$1 [NE]
```

```
RewriteRule ^/customconfig/catalog/1/(.*$) /customconfig/mypub/publisher/mypub/catalog/
$1 [NE,PT]

# Write per-publisher rules for publisher, version, file, and manifest responses.

# Serve the static versions and publisher responses.
RewriteRule ^/customconfig/mypub/versions/0 %{DOCUMENT_ROOT}/versions/0/index.html
 [L,NE]
RewriteRule ^/customconfig/mypub/publisher/0 %{DOCUMENT_ROOT}/customconfig/mypub/
publisher/1/index.html [L,NE]

RewriteRule ^/customconfig/mypub/catalog/1/(.*)$ /customconfig/mypub/publisher/mypub/
catalog/$1 [NE,PT]
RewriteRule ^/customconfig/mypub/file/1/(..)(.*)$ /customconfig/mypub/publisher/mypub/
file/$1/$1$2 [NE,PT]
RewriteRule ^/customconfig/mypub/manifest/0/.*$ %{THE_REQUEST} [NE,C]
RewriteRule ^GET\ /customconfig/mypub/manifest/0/([^@]+)@([^\ ]+)(\
 HTTP/1.1)$ /customconfig/mypub/publisher/mypub/pkg/$1/$2 [NE,PT,C]
RewriteRule ^/customconfig/mypub/(.*)$ %{DOCUMENT_ROOT}/customconfig/mypub/$1 [NE,L]

# Create an alias to serve /var/share/pkgrepos/myrepo content.
# Map the web server URI namespace to a location on the file system.
Alias /customconfig/mypub /var/share/pkgrepos/myrepo
# Enable the web server to see that file system location.
<Directory "/var/share/pkgrepos/myrepo">
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>

# Versions response.
RewriteRule ^/.*[/]?versions/0/?$ %{DOCUMENT_ROOT}/versions/0/index.html [L]
# Allow for 'OPTIONS * HTTP/1.0' requests.
RewriteCond %{REQUEST_METHOD} OPTIONS [NC]
RewriteRule \* - [L]

# Location matches based on the final Rewrite paths for file,
# manifest, catalog, and publisher responses.
<LocationMatch ".*/file/../[a-zA-Z0-9]+$">
        Header set Cache-Control "must-revalidate, no-transform, max-age=31536000"
        Header set Content-Type application/data
</LocationMatch>
<LocationMatch ".*/publisher/.*/pkg/.*">
        Header set Cache-Control "must-revalidate, no-transform, max-age=31536000"
        Header set Content-Type text/plain;charset=utf-8
</LocationMatch>
<LocationMatch ".*/catalog/catalog.*.C">
        Header set Cache-Control "must-revalidate, no-transform, max-age=86400"
```

```
        Header set Content-Type text/plain;charset=utf-8
</LocationMatch>
<LocationMatch ".*/catalog.attrs">
        Header set Cache-Control no-cache
</LocationMatch>
<LocationMatch ".*/publisher/\d/.*">
        Header set Cache-Control "must-revalidate, no-transform, max-age=31536000"
        Header set Content-Type application/vnd.pkg5.info
</LocationMatch>
```

Copy `/tmp/runtime` to the Linux system. The `depot.conf` configuration fragment file must be added to the Apache configuration on the Linux system, and the `htdocs` files must be accessible from the `DocumentRoot` of the Linux system.

On Oracle Solaris systems that need to access this repository, set the origin of the `mypub` publisher to the `http` address of the Linux system.

```
$ pkg set-publisher -g http://linux-system-name:8080/mypub mypub
```

You could have used the `-P` option to the `pkg.depot-config` command to root the repository further down the URI namespace. For example, the following `pkg.depot-config` command enables you to have a `mypub` origin of `http://`*linux-system-name*`:8080/ips-repos/mypub`:

```
$ /usr/lib/pkg.depot-config -F -P ips-repos -d customconfig=/var/share/pkgrepos/myrepo -
r /tmp/runtime
```

♦ ♦ ♦ CHAPTER 4

Maintaining Your Local IPS Package Repository

This chapter describes how to do the following tasks:

- Update packages in an IPS repository
- Maintain multiple identical repositories
- Serve multiple different repositories using HTTP access
- Set or update properties of a repository
- 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. See "Best Practices for Creating and Using Local IPS Package Repositories" on page 14 and "Oracle Solaris Repository Content" on page 19 for more information.

  Users can update to a version earlier than the latest version in the repository by specifying the version of the `pkg:/entire` constraint package to install. See "Updating to a Version Older Than the Newest Version Allowed" in *Adding and Updating Software in Oracle Solaris 11.3*.

  For information about keeping your repository as small as possible see "Minimal Required Repository" on page 15 and the remove and replace instructions in Step 3 in "How to Update a Local IPS Package Repository" on page 56.
- 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 60 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 36 for instructions for using the `mirror` service.

Use the following steps to minimize disruption to clients. With this procedure, the update takes place out of sight, replacing the repository with the updated repository is fast, and clients do not need to change their `solaris` publisher origin or add a new origin.

1. **Make a ZFS snapshot of the package repository.**
   Make sure you have a current snapshot of the repository to be updated.

   Note - This step is especially important if you are going to use `pkgrecv --clone` to update the repository, because the clone operation leaves the destination repository altered if an error occurs.

   ```
   $ zfs list -rt all rpool/VARSHARE/pkgrepos/solaris
   NAME                                          USED  AVAIL  REFER  MOUNTPOINT
   rpool/VARSHARE/pkgrepos/solaris              17.6G  78.4G    34K  /var/share/pkgrepos/
   solaris
   rpool/VARSHARE/pkgrepos/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/VARSHARE/pkgrepos/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 34. Use this new snapshot for the update.

2. **Make a ZFS clone of the snapshot.**

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

```
$ zfs clone rpool/VARSHARE/pkgrepos/solaris@initial rpool/VARSHARE/pkgrepos/solaris_tmp
```

The `%ipsrepo` in the following listing is the share from "How to Enable Users to Retrieve Packages Using a File Interface" on page 42.

```
$ zfs list -rt all rpool/VARSHARE/pkgrepos
NAME                                          USED  AVAIL  REFER  MOUNTPOINT
rpool/VARSHARE/pkgrepos                      17.6G  78.4G    33K  /var/share/pkgrepos
rpool/VARSHARE/pkgrepos%ipsrepo                 -      -      -  /var/share/pkgrepos
rpool/VARSHARE/pkgrepos/solaris              17.6G  78.4G  17.6G  /var/share/pkgrepos/
solaris
rpool/VARSHARE/pkgrepos/solaris@initial          0      -  17.6G  -
rpool/VARSHARE/pkgrepos/solaris_tmp           76K  78.4G  17.6G  /var/share/pkgrepos/
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.

(Optional) To minimize the size of the repository, replace the repository rather than add packages with every repository update. Use one of the following two methods to replace the repository in the snapshot clone:

- Use the `--clone` option of the `pkgrecv` command as described in "Update from a repository" below.
- Remove all `solaris` content from the repository and add only the content that is needed to support current systems. Do not try to select a subset of repository content to remove. Remove all `solaris` content and then add all content that you need for current systems according to the list in "Minimal Required Repository" on page 15. Do not remove all content from the clone; remove only the content under the `publisher/solaris` directory as shown in the following command:

  ```
  $ rm -rf /var/share/pkgrepos/solaris_tmp/publisher/solaris/*
  ```

Use one of the following methods to update the repository in the snapshot clone:

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

  a. **Follow the instructions in Example 2, "Adding to an Existing Repository From a `zip` File," on page 32.**

  Specify the `-v` option to the `install-repo.ksh` script to verify the repository.

If the specified destination already contains a package repository, the content of the `zip` file is added to the content of the existing repository, and a `pkgrepo rebuild` is performed to update catalogs and indexes.

b. Continue with Step 6 below.

- **Update from an ISO file.**

  a. Create the ISO file.

  Create an `.iso` file from the repository `.zip` files using the `-I` option as described in Example 1, “Creating a New Repository From a `zip` File,” on page 30.

  b. Mount the ISO image.

  ```
  $ mount -F hsfs ./sol-11_3-incr-repo.iso /mnt
  ```

  c. 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 32.

  ```
  $ rsync -aP /mnt/repo/ /var/share/pkgrepos/solaris_tmp
  ```

  d. 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 /var/share/pkgrepos/solaris_tmp \
  --key /path-to/key-file --cert /path-to/certificate-file '*'
  ```

  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 23.

  The `--clone` option of the `pkgrecv` command makes the repository copy significantly faster by performing a differences-only update of the destination repository instead of a series of incremental additions. The `--clone` option makes an exact copy of the source repository, including deleting packages from the destination repository that are not in the

source repository. If you use the `--clone` option, make sure the source repository contains everything you need according to the list in "Minimal Required Repository" on page 15. If an error occurs, roll back to the most recent repository snapshot.

If the `pkgrecv` operation is interrupted, follow the instructions in "Resuming an Interrupted Package Receive" on page 60.

4. **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.

5. **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 /var/share/pkgrepos/solaris_tmp
```

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

```
$ zfs snapshot rpool/VARSHARE/pkgrepos/solaris_tmp@S11U3SRU2
```

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

```
$ svcadm disable -st pkg/server:solaris
$ zfs promote rpool/VARSHARE/pkgrepos/solaris_tmp
$ zfs rename rpool/VARSHARE/pkgrepos/solaris rpool/VARSHARE/pkgrepos/solaris_old
$ zfs rename rpool/VARSHARE/pkgrepos/solaris_tmp rpool/VARSHARE/pkgrepos/solaris
```

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

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/VARSHARE/pkgrepos/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 /var/share/pkgrepos/solaris_tmp \
--key /path-to/key-file --cert /path-to/certificate-file '*'
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 achieve 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.
- Configure load balancing, as described in "Configuring Load Balancing" on page 84.

Make sure that the repository that will be the source for the clone repositories is current. If necessary, use the "How to Update a Local IPS Package Repository" on page 56 procedure to update the source repository. Then use one of the following clone procedures:

- "How to Clone a Local IPS Package Repository" on page 61 – Create clones of the source repository
- "How to Update a Cloned Local IPS Package Repository" on page 63 – Update existing clone repositories

The clone procedures are very similar to the "How to Explicitly Copy a Repository From the Internet" on page 34 and "How to Update a Local IPS Package Repository" on page 56

procedures. The `pkgrecv` operation shown in both of the clone procedures copies the source repository files exactly (`--clone`), 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 must be exactly the same to avoid problems when the load balancer switches clients from one node to another. See "Configuring Load Balancing" on page 84 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.
- The repository copy operation is significantly faster than the default repository copy operation.

Note - The goal and result of the clone procedure is that all repositories are exact duplicates. Configure all clone repository locations as origins on every system. IPS automatically selects the location with the best connection for each client.

▼ How to Clone a Local IPS Package Repository

This procedure assumes that you are creating new duplicate repositories. If you are updating duplicate repositories that already exist, use the "How to Update a Cloned Local IPS Package Repository" on page 63 procedure.

This procedure is very similar to the "How to Explicitly Copy a Repository From the Internet" on page 34 procedure. Some of the steps in this procedure are abbreviated. For more detail, see "How to Explicitly Copy a Repository From the Internet" on page 34.

Before You Begin Make sure that the repository that will be the source for the clone repositories is current. If necessary, use the "How to Update a Local IPS Package Repository" on page 56 procedure to update the source repository before you create the clone repositories.

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

Create the repository in a shared location. Set `atime` to `off` when you create the repository file system. Consider setting the `compression` property. See "Best Practices for Creating and Using Local IPS Package Repositories" on page 14.

If other repositories exist on the same system as this clone repository, you might need to use a different ZFS *filesystem* name.

```
$ zfs create -o atime=off rpool/VARSHARE/pkgrepos
$ zfs create rpool/VARSHARE/pkgrepos/solaris
```

2. **Create the required repository infrastructure.**
   Create the required IPS repository infrastructure so that you can copy the source repository.

   ```
   $ pkgrepo create /var/share/pkgrepos/solaris
   ```

3. **Set the default publisher.**

   ```
   $ pkgrepo -s /var/share/pkgrepos/solaris set publisher/prefix=solaris
   ```

4. **Copy the repository content to the new location.**
   Use the `pkgrecv` command to duplicate the updated source repository to the new clone repository.

   ```
   $ pkgrecv -s /net/host1/var/share/pkgrepos/solaris \
   -d /net/host2/var/share/pkgrepos/solaris --clone -p '*'
   ```

   The `-p` option clones content from all publishers, in case content was added from a new publisher when the source repository was updated. See the `pkgrecv(1)` man page for information about the `--clone` option.
   If the `pkgrecv` operation is interrupted, follow the instructions in "Resuming an Interrupted Package Receive" on page 60.

5. **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.

6. **Snapshot the new repository.**

   ```
   $ zfs snapshot rpool/VARSHARE/pkgrepos/solaris@sol-11_3_19
   ```

7. **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.

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 64 describes how to serve multiple repositories using multiple `pkg.depotd` daemons running on different ports.
- "Multiple Repositories Under One Domain" on page 84 describes how to run multiple repositories under one domain name with different prefixes.

▼ How to Update a Cloned Local IPS Package Repository

This procedure assumes that you are updating duplicate repositories that already exist. If you are creating new duplicate repositories, use the "How to Clone a Local IPS Package Repository" on page 61 procedure.

This procedure is very similar to the "How to Update a Local IPS Package Repository" on page 56 procedure except that this procedure copies the source repository exactly by using the `--clone` option. The steps in this procedure are very abbreviated. For detail, see "How to Update a Local IPS Package Repository" on page 56.

Before You Begin Make sure that the repository that will be the source for the clone repositories is current. If necessary, use the "How to Update a Local IPS Package Repository" on page 56 procedure to update the source repository before you update the clone repositories.

1. Copy the destination repository.

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

This step is especially important when making a clone, because the clone (`--clone`) operation leaves the destination repository altered if an error occurs.

2. Update the copy of the destination repository.

Use the `pkgrecv` command to duplicate the updated master repository to the copy of the destination repository.

```
$ pkgrecv -s /net/host1/var/share/pkgrepos/solaris \
-d /net/host2/var/share/pkgrepos/solaris_tmp --clone -p '*'
```

The `-p` option clones content from all publishers, in case content was added from a new publisher when the source repository was updated. See the `pkgrecv(1)` man page for information about the `--clone` option.

If an error occurs, roll back to the most recent repository snapshot.

If the `pkgrecv` operation is interrupted, follow the instructions in "Resuming an Interrupted Package Receive" on page 60.

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 64 describes how to serve multiple repositories using multiple `pkg.depotd` daemons running on different ports.
- "Multiple Repositories Under One Domain" on page 84 describes how to run multiple repositories under one domain name with different prefixes.

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 43 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 74.

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 43.

1. **Create a new repository 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. **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=/var/share/pkgrepos/solarisstudio
   ```

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

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

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

   See "Configuring a Simple Prefixed Proxy" on page 83 for an example of setting the `pkg/proxy_base`.

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

   Make sure the repository name and description are set as shown in "Modifying Repository Property Values" on page 73.

6. **Start the repository service.**

   Restart the package depot server service.

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

7. **Check that the new instance is online.**

   ```
   $ 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
```

If the `pkg/server:studio` instance is not `online`, use the `svcs -xvL pkg/server:studio` command to get more information to diagnose and fix the problem.

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 /var/share/pkgrepos/solarisstudio/ solarisstudio
     ```

   - The `pkg/port` location.

     ```
     $ pkg set-publisher -g http://localhost:82/ solarisstudio
     ```

See Also See "Multiple Repositories Under One Domain" on page 84 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

This procedure describes using a `svc:/application/pkg/depot` service. Running a `pkg/depot` service is more scalable than running the `pkg.depotd` process under a `pkg/server` service.

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

Before You Begin To get the `svc:/application/pkg/depot` SMF service and the `pkg.depot-config(1M)` man page, make sure the `pkg:/package/pkg/depot` package is installed.

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

   The `pkg/server` instance provides the file system location of the repository for the `pkg/depot` 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`.

   ```
   $ svccfg -s pkg/server:studio listprop pkg/readonly
   pkg/readonly boolean     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 `pkg/depot:default` service is 80. To change the port number, set the `config/port` property of `pkg/depot:default`. The `pkg/port` values for `pkg/server` instances that are managed by the `pkg/depot:default` service are not used.

6. **(Optional) Set the system name for the `pkg/depot` instance.**

   Set the value of the `config/host` property of the `pkg/depot:default` service to the IP address or host name of the system that hosts the repository file system.

   ```
   $ svccfg -s pkg/depot:default setprop config/host=pkg.example.com
   ```

If you need to access this repository URI from other systems, specify the host name to use as the argument to the Apache `ServerName` directive. For more information, see the `pkg.depot-config`(1M) man page.

7. **Start the `pkg/depot` instance.**

   - **If the `pkg/depot` instance is `disabled`, refresh and enable the instance:**

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

     If the `pkg/depot` instance is not `online`, use the `svcs -xvL pkg/depot` command to get more information to diagnose and fix the problem.

   - **If the `pkg/depot` instance is `online`, refresh and restart the instance:**

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

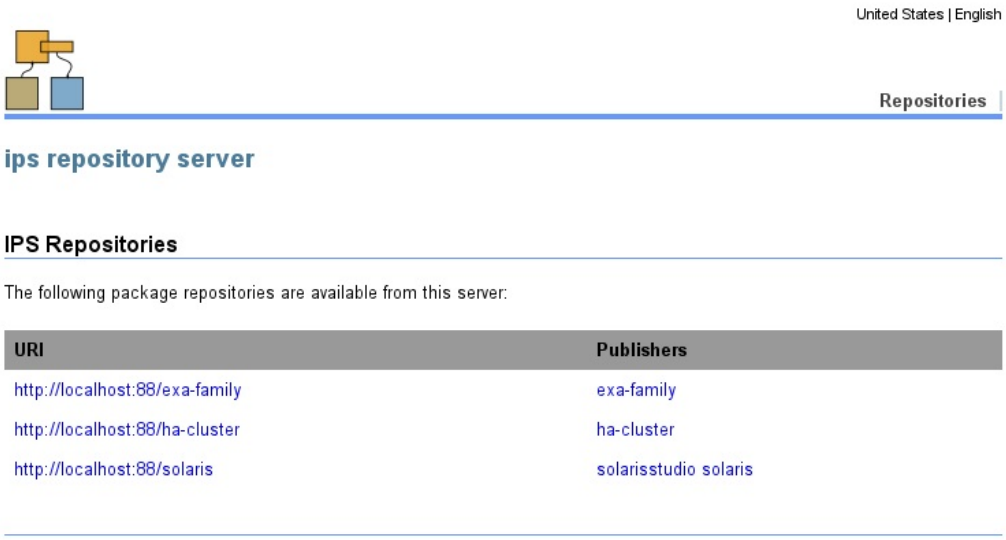
8. **Test that the repository server is working.**

   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.

   In the following example, `pkg/server` instances `solaris`, `ha-cluster`, and `exa-family` all have `pkg/standalone` set to `false` and `pkg/readonly` set to `true`. The `pkg/inst_root` location of the `pkg/server:solaris` instance provides packages from two publishers: `solaris` and

`solarisstudio`. The `config/port` value of this `pkg/depot` instance is set to 88. The following figure shows the display at `localhost:88`.



The `pkg/depot:default` service has associated `httpd.worker` processes. The `pkg/server` instances do not:

```
$ svcs -p pkg/depot pkg/server
STATE          STIME    FMRI
online         16:44:31 svc:/application/pkg/server:solaris
online         16:44:32 svc:/application/pkg/server:ha-cluster
online         16:44:32 svc:/application/pkg/server:exa-family
online         16:45:03 svc:/application/pkg/depot:default
               16:45:02     6834 httpd.worker
               16:48:04     6948 httpd.worker
               16:48:04     6949 httpd.worker
               16:48:04     6950 httpd.worker
               16:48:05     6953 httpd.worker
               16:53:11     7043 httpd.worker
```

9. Set the publisher origin.

Set the publisher origin to one of the following values:

- The unique `pkg/inst_root` location.

```
$ pkg set-publisher -g /var/share/pkgrepos/solarisstudio/ solarisstudio
```

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

```
$ pkg set-publisher -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 55 and "Customizing Your Local Repository" on page 74, 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.

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 /var/share/pkgrepos/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 /var/share/pkgrepos/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 publish` command (see the `pkgsend`(1) man page) 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 11.3* 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 11.3* 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/`. Except for the leading `/` character, 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 11.3* and in the `pkg`(1) man page. The value of the image property is relative to the image, so that default value is `etc/certs/CA`.

If you create your own SSL Certificate Authority certificates, put those certificates in the directory named by `repository/trust-anchor-directory` and refresh the `ca-certificates` service as described in "Creating a Self-Signed Server Certificate Authority" on page 92 and "How to Use a Custom Certificate Authority Certificate"

in *Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.3*. Put the CA certificates directly in the directory named by `repository/trust-anchor-directory`; do not put the certificates in another subdirectory.

`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 73. 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 /var/share/pkgrepos/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 systems 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 72 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 /var/share/pkgrepos/solaris \
repository/description="Local copy of the Oracle Solaris 11 repository." \
repository/name="Oracle Solaris 11"
$ pkgrepo get -p solaris -s /var/share/pkgrepos/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 11.3* 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 11.3*. 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 76.

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
```

A value in the `O` column indicates whether the package is obsolete (`o`) or renamed (`r`).

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 /var/share/pkgrepos/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 /var/share/pkgrepos/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 /var/share/pkgrepos/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 /var/share/pkgrepos/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 74, 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 11.3* 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 56.

♦ ♦ ♦ CHAPTER 5

Running the Package Depot Server Behind a Web Server

If you only want to scale to serve more clients concurrently, use the `pkg/depot` service as described in "How to Enable Users to Retrieve Packages Using an HTTP Interface" on page 43. Use the instructions in this chapter if you want to configure additional capabilities such as caching, load balancing, multiple repositories under one domain, and secure access.

Running the package depot server behind an Apache web server instance provides a web server configuration that employs the best practices described in this chapter, including the following benefits:

- Provides depot scalability without the need for manual web server configuration or reverse proxies.
- Improves performance and availability. Increases transfer rates and decreases CPU load. Running the depot server behind a web proxy can improve the performance and availability of the repository by enabling load balancing over multiple depots and enabling content caching.
- 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.
- Enables providing a secure depot 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 11.3 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 82) and repository search results (see "Cache Considerations for Search" on page 82), 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, including using the htcacheclean command to control the size of the cache. See *Developing System Services in Oracle Solaris 11.3* for information about how to create a periodic or scheduled SMF service to run `htcacheclean`.

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 any single file to be cached. The Apache default of 1 MB might be too small for most repositories. The following directive sets the maximum size of a cached file 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 64 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
```

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 83 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 system, 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 must be exactly the same to avoid problems when the load balancer switches clients from one node to another. To ensure that the catalogs are exactly the same, clone the repositories that participate in load balancing by using the `pkgrecv --clone` command as described in “Maintaining Multiple Identical Local Repositories” on page 60.

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 83.

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 83 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 system 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, as described in “Creating a Certificate Authority for Client Certificates” on page 88.
2. Issue certificates from this CA to the clients that are allowed to access the repository, as described in “Creating Client Certificates Used for Accessing the Repository” on page 89.

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:

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 11.3*.

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 “Adding SSL Configuration to the Apache Configuration File” on page 91. 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 90.

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 88.
- 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 89. 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 clients that need to access the SSL-protected repository.

Enabling Systems to Access the Protected Repository

To access the SSL-protected repository, clients 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. 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.

Adding 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 79. 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

        # enable all available TLSv1 flavors, but not SSLv2 or SSLv3
        SSLProtocol all -SSLv2 -SSLv3

        # 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 88.

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 `subjectAltName` directive as described in Subject Alternative Name in the OpenSSL documentation.

```
$ pktool gencsr label=apache subject="CN=pkg-sec.internal.example.com" \
altname="IP=192.0.2.0,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 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:

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

Refresh 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 89 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 92.

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 87.

▼ 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

        # enable all available TLSv1 flavors, but not SSLv2 or SSLv3
        SSLProtocol all -SSLv2 -SSLv3

        # 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

R