File Sharing

and Other Helpful Facts

for

HP-UX 10.0 Software Developers

Version 1.0

HEWLETT PACKARD

Last Modification: September 29, 1994

1. Introduction

This document is required reading for HP-UX 10.0 software developers. The information contained herein must be applied to all applications for HP-UX 10.0. Failure to do so may render a software product useless in an HP-UX 10.0 environment.

1.1 Abstract

HP-UX 10.0 is introducing many changes that affect the fundamental structure of the operating system. The major changes include a new file system layout (10.0 Filesystem Layout), a new diskless paradigm (NFS Diskless), a new software delivery mechanism (Software Distributor) and a converged kernel I/O system. Individually, each change is outlined and discussed in related documents. Collectively, these new technologies play together to create a development and operating environment that is vastly different from previous releases of HP-UX. When developing applications, software developers must be aware of not only the individual technology changes, but also how the changes ‘play together’ and affect product structure, installation and execution.

1.2 Purpose

The purpose of the document is to link the major HP-UX 10.0 technologies together (10FSL, NFSD, SD, Kernel) and provide a comprehensive, inter-related picture for software developers that enables them to correctly construct applications for HP-UX 10.0.

1.3 Audience

This document is mainly intended for software developers constructing applications for HP-UX 10.0. However, the material contained herein may also be useful to system adminstrators wishing to understand finer details of HP-UX 10.0.

1.4 Prerequisite and Related Documentation

Each of the major technologies is covered in detail in other related documentation. These documents should be read thoroughly **BEFORE** attempting to read this document:

[1] *HP-UX 10.0 File System Layout,* Hewlett-Packard Co., 1994

[2] *HP-UX 10.0 NFS Diskless Concepts and Administration*, Hewlett-Packard Co., 1994

[3] *Managing HP-UX Software*, Hewlett-Packard Co., 1994

Also included in this list are documents related to SD-UX and the Kernel I/O changes for HP-UX 10.0. Their references were not available at authoring time.

2. Terminology

2.1 File Sharing

HP-UX 10.0 is based upon a file sharing model that is much different than in past releases. The operating system and diskless environment are based upon NFS. This protocol, along with some implementation specifics, is used to share files between clients and servers. The terminology associated with file sharing is explained below.

2.1.1 Sharing Point

A sharing point is typically a product-specific directory containing files that may be shared among multiple systems. Because sharing points may be used by more than one system, they contain only ***static files***, such as man pages, libraries, default configurations, and commands. Files below a sharing point do not change during system runtime execution. Sharing points do *not* contain dynamic files such as user files, temporary files, log files, or system specific configuration data. Files below a sharing point must be static because they are exported to other systems via ***read-only*** NFS mounts.

Examples of sharing points include **/usr**, **/sbin** and **/opt/<appl>**.

2.1.2 Shared Root

A shared root is a directory containing one or more sharing points. A shared root typically contains sharing points that compliment one another. For example, one shared root (OS shared root) may contain both **/usr** and **/sbin** sharing points because both sharing points contain all of the static operating system files; these two directories are always co-located in the same shared root.

Shared roots come in two flavors: ***OS*** (operating system) and ***Application*** (non-OS). OS shared roots contain sharing points associated with the operating system (**/usr** and **/sbin**). Directories below the OS shared root include **/usr** and **/sbin**, and may also include **/opt/<appl>** directories. Application shared roots contain only non-OS products that reside below **/opt/<appl>**.

Shared roots are located on an NFS server. Some examples include:

- **/export/shared_roots/OS_700** : an OS shared root. On a S700, this is a symbolic link to / (using the **/sbin** and **/usr** sharing points there), but on a S800 this directory physically contains the operating system software.
- **/export/shared_roots/APPL_700** : an application shared root which contains software residing below /opt.

2.1.3 Private Root

A private root contains all the dynamic files associated with diskless clients. On an NFSD server, the private root is found below **/export/private_roots/<client>**. Files below a client’s private root are *not* shared among systems because they are system specific and may change during the system runtime execution. Directories found below a client’s private root include **/dev, /etc, /home, /mnt, /net, /stand, /tmp and /var**.

On a diskless client, **/opt** is a collection of mount points and may be configured differently for each client. Because its contents may vary among systems, **/opt** is part of the private root.

2.2 Install Root and Alternate Root

The install root is the location at which software is loaded during extraction from the product media. For a typical software installation, the install root is ‘**/**’. However, when configuring NFSD servers in a heterogeneous environment, the install root is usually located below the **/export/shared_roots** directory. An installation to a directory other than ‘**/**’ is called an alternate root installation.

For example, assume **/sbin/rc** is contained on the install media. If the customer chooses to load this file in the default install root of ‘**/**’, the file is loaded to **/sbin/rc**. However, if the customer chooses to change the install root, perhaps to **/foo/bar**, the file is loaded to **/foo/bar/sbin/rc**.

The SW_ROOT_DIRECTORY parameter holds the value of the install root. If its value is anything other than ‘**/**’, the value denotes an alternate root.

2.3 Product Directory

A product directory defines the location of a product below an install root. Product directories must be defined in the PSF (product specification file) and affect both product relocation and link-installation (see “*Managing HP-UX Software*”).

If a product is specified as relocatable, the product directory value in the PSF can be overridden by the customer with a new value. This has the effect of re-mapping any pathnames which contained a leading component equal to the PSF product directory to the new location. Files in the product which do not have the PSF product directory as a leading path component are unaffected.

For example, consider a product called *Baseball* which contains two files: **/opt/baseball/bin/command** and **/sbin/init.d/baseball**. The product directory in the PSF is set to **/opt/baseball**. If the product is specified as relocatable, the product directory can be overridden by the user. If a user were to override the product directory to **/opt/football**, it would have the following results:

> /**opt/baseball/bin/command** would be re-mapped to **/opt/football/bin/command** because the initial path component of the file matches the PSF product directory specification.
>
> /**sbin/init.d/baseball** is loaded as-is and is not re-mapped because the initial path component does not match the PSF product directory specification.

Most products will have a PSF product directory specification of **/opt/<product>**.

The product directory attribute also affects link-install and is discussed in the next section.

2.4 Share Link

A share link defines a sharable product directory that allows a product to be link-installed to other systems. When a product is link-installed to a client, the share link attribute is used to construct the server’s export path of the product, such that clients may import through the link-install process. The exported directory is a concatenation of the install root, product directory and the share link. Assume Example product had the following attributes:

> install root = /export/shared_roots/OS_700
>
> product directory = /opt/Example
>
> share link = .

The resulting exported path of Example product to a client would be **/export/shared_roots/OS_700/opt/Example**. Share links must be defined in the product PSF. Most products will have a share link attribute value of ‘**.**’.

3. SD Control Script Summary

SD provides many different control scripts that are executed during various stages of product installation and deinstallation. This section provides a brief summary of control scripts, along with specific requirements that relate to file sharing.

3.1 Control Script Review

When operating in a standalone environment, all control scripts are executed on the target system. However, when loading and configuring software in a diskless environment, not all scripts are executed on both the client and server. Software developers must account for these differences when designing control scripts. The table below summarizes script execution in a diskless environment:

Control Script	Run on Server	Run on Client
checkinstall	Yes	No
preinstall	Yes	No
postinstall	Yes	No
configure	Yes	***Yes***
verify	Yes	***Yes***
checkremove	Yes	No
unconfigure	Yes	***Yes***
preremove	Yes	No
postremove	Yes	No

Control scripts that ***execute only on the server*** and not on the client:

- Must restrict their actions to sharing points within a single shared root.
- Must not perform any system specific configuration actions. (Exception: postinstall may affect the kernel if SW_DEFERRED_KERNBLD is unset.)
- Must perform all functions relative to the SW_ROOT_DIRECTORY.

Control scripts that ***execute both on the server and client***:

- Must restrict their actions to the private root of the system.
- Must perform system specific actions only.
- Must not modify the contents of the sharing point(s).

3.1.1 Control Scripts That Execute On The Server

Software can be installed for use by the system it is installed on, or in the case of an alternate root installation, for use by diskless clients or other systems. Software which is loaded into an alternate root is “installed”, but not “configured”. In other words, because it was loaded into an alternate root, it is not configured for use by the system on which it was loaded. But, certain aspects of software configuration, which are specific to the software, but not the system must still be executed.

Because of these kinds of issues, control scripts are split into two categories: those which operate in the context of the software that was loaded, and those which operate in the context of the system on which the software is to be used.

Control scripts falling into the first category include the *checkinstall, preinstall, postinstall, checkremove, preremove,* and *postremove* scripts. These have the following restrictions:

3.1.1.1 Execute relative to a sharing point

Within a shared root, there can be many sharing points (e.g. **/usr, /sbin, /opt/baseball, /opt/football**). Software configuration must constrain itself to only operating within the sharing points it has defined.

For example, if the *Baseball* product chooses to venture out of its sharing point of **/opt/baseball** and touch the file **/etc/opt/ball.config** within an installation control script, this configuration step will be lost because the **/opt/baseball** directory will be exported to another system (e.g. a diskless client), but the **/etc/opt/ball.config** file will not be.

3.1.1.2 Don't perform system specific configuration actions

Files outside of the sharing point for the software in question must not be affected. For example, a product called *Baseball* may normally install to **/opt/baseball**, but if it is being installed to an alternate root, may actually end up in **/export/shared_roots/OS_700/opt/baseball.** It is very important that the installation related control scripts do not attempt to affect configuration of the system on which the software was loaded.

Perhaps the *Baseball* product has a configuration file called **/etc/opt/ball.config**. This file should only be present on a system which is actually using the software in question. But, software which is installed to an alternate root is never configured for use by the system on which it was physically loaded.

Other reasons to not do system specific configuration in an installation script include:

- Software will not be used on the system it is installed on.
- Software may not match the architecture of the system on which it was installed.

3.1.1.3 Perform all functions relative to the SW_ROOT_DIRECTORY

Software is always installed relative to a root directory. In most cases, the root directory is **/**, but in the case of an alternate root install, it can be anything. Installation control scripts must understand the concept of alternate root installs and be capable of doing software specific (as opposed to system specific) configuration relative to the root in which the software was installed.

The SW_ROOT_DIRECTORY variable is set up by SD and can be used as a prefix to paths that are to be affected during execution of a control script. For example, if software installation consists of making a hard link from **/usr/bin/foo** to **/usr/bin/bar**, the code in the control script should actually execute the following command:

```
ln $SW_ROOT_DIRECTORY/usr/bin/bar $SW_ROOT_DIRECTORY/usr/bin/foo
```

The ROOT variable is also available for applications which load under **/opt**. $ROOT is a concatenation of the root directory of the alternate root and the root directory of the application. For example, if the alternate root is **/export/shared_roots/APPL_700** and the application has specified a software directory of

/opt/baseball, then ROOT will be **/export/shared_roots/APPL_700/opt/baseball**. This variable can be used to do software configuration relative to the install directory (as opposed to the install root).

3.1.2 Control Scripts That Execute On All Systems

The second type of control script is a configuration script. These scripts are responsible for configuring software to run on a given system, checking that configuration, and also unconfiguring software. These scripts include the configure, verify, and unconfigure control scripts.

3.1.2.1 Restrict actions to the private root

There are many reasons why a configuration control script must restrict its activities to outside of sharing points, but the most obvious is that for support of NFS diskless, the sharing points are mounted read-only. This is done to protect software which may be used by many systems from receiving system specific configuration that is only appropriate for a single system. Any attempt to operate on files within a sharing point will result in error.

3.1.2.2 Only perform system specific actions

Software installation, and its associated scripts are responsible for making sure that the files in the sharing points are correctly configured. In contrast, software configuration scripts are responsible for making sure that files outside the sharing points are correctly configured so that the software can be executed.

4. Application Classification

When considering the number of possible system configurations and end user environments, developers may become overwhelmed with the number of details for correct product placement and installation in a 10.0 environment. In the following section, we attempt to classify applications into three distinct categories, along with a set of factors that helps determine product category. After completion, the reader should be able to identify their product type and understand the implications for correct product installation and location.

4.1 Product Types

4.1.1 OS Application Product

OS application products are natural extensions to the operating system. Such products are usually hidden from an end user and are only known to a system administrator. Products that fit into this generalized category are networking products, new file system types, and distributed computing services. Customers generally feel that these are not applications, but rather part of the operating system they are using.

A product falling into this category:

- Loads its entire product contents into the operating system sharing points, **/usr** and **/sbin.**
- Uses **/etc** for configuration files and **/var** for runtime or variable files.
- Is small in size, because the **/usr** and **/sbin** directories may be limited in size.
- Enhances the core operating system functionality.

4.1.2 Pure Application Product

Pure application products naturally ‘look & feel’ like applications. These typically have large user interfaces and are mostly intended for end-users.

A product falling into this category:

- Loads its entire product contents into its own sharing point, **/opt/<appl>**.
- Uses **/etc/opt/<appl>** for configuration files and **/var/opt/<appl>** for runtime and variable files.
- Is large in size, and can make use of mount points below **/opt.**

4.1.3 Hybrid Application Product

Some products may fit into the pure application type, but require some interaction with the operating system sharing points. These types of products are hybrids. They may exhibit some of the ‘look & feel’ characteristics of the previous two types, but are considered a hybrid, for reasons that are discussed in the next section.

A product falling into this category:

- Loads its entire product contents into its own sharing point, **/opt/<appl>**, with the exception of single-user mode commands.
- Utilizes the postinstall script to:
    - *copy* startup/shutdown scripts from the product sharing point to **/sbin/init.d**

- •*create* startup/shutdown links below **/sbin/rc*.d**
- •*link* any kernel libraries from **/usr/conf/lib** to the product sharing point
- •*link* any master file components from **/usr/conf/master** to the product sharing point

- Must ensure that it loads into an operating system shared root. This is required because some files will be placed below **/usr** and **/sbin** and the correct location for these directories is below the operating system shared root.

4.2 Factors Determining Product Type

When determining the proper classification for a product, there are a number of different factors that should be considered. Each of the factors are applied later in the section to determine product type. The reader should evaluate each factor against their product.

4.2.1 System Boot Interactions with the Operating System

Some products require commands or files to be available before all file systems have been mounted or during single user mode. With HP-UX 10.0, only **/sbin** is guaranteed to be available during system boot, requiring the applicable product components to be located below **/sbin**.

Some products require that tasks, such as starting daemons, occur during system boot. This is accomplished through the startup/shutdown model for HP-UX 10.0. The startup scripts used during system startup physically reside in **/sbin**.

If a product has single-user or recovery-mode commands, or a startup/shutdown script, it is considered to have system boot interactions. Some examples include:

- Taskbroker and NetLS have a daemon to start and use a startup script to do so.
- A third party file system may have fsck-like code that must be located in **/sbin** so that other file systems can be checked before being mounted.

4.2.2 Kernel Functionality

Some products deliver new kernel functionality through kernel libraries and master file components below **/usr/conf**. If a product delivers below either one, or both, it is considered to deliver kernel functionality. Some examples include:

- A virtual reality environment design tool may deliver a driver to control an EISA card connected to a VR harness (helmet, glove, etc...).
- A nuclear reactor administration system may deliver device drivers to control interfaces for the control rods and cooling systems.
- X.25 delivers drivers for the X.25 network protocol.

NOTE: If a product only modifies a kernel tunable parameter, the product is *not* considered to modify kernel functionality.

4.2.3 User Interactions

Some products are targeted for end-users or system admins, while others are not so obvious. When evaluating a product’s user interaction category, determine which interactions dominate the product: user or system administrator. Products that do not have any interactions (they are just resident) are considered to have system administrator interactions. Some examples include:

- FrameMaker and C++ are typically used by the end-user.
- OpenView and UPS (uninterruptable power supply monitor) are typically used by a system administrator.
- A third-party file system is just there. Everyone may be using it, but it is transparent to the end-user.

4.2.4 User Interface

Some products have an extensive user interface, including GUIs (graphical user interface) or TUIs (textual user interface), while others have either no interface or only accept command line arguments. Some examples include:

- OpenView, SD, FrameMaker, and Glance+ have significant user interfaces.
- UPS and C++ have little or no user interfaces.

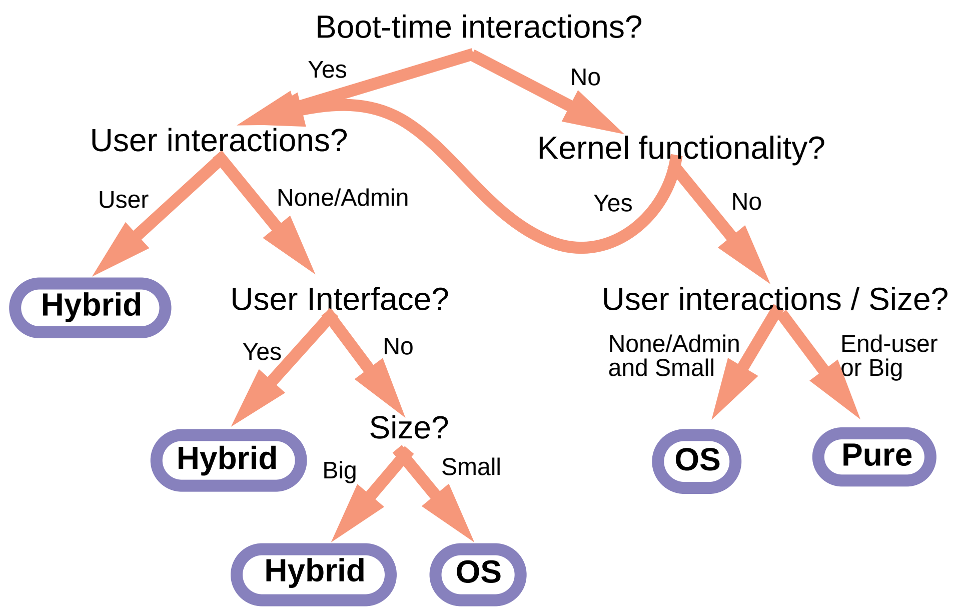
4.2.5 Size

One important product attribute is required disk space. This is the most subjective attribute to judge. When evaluating your product, consider whether or not a customer may want to mount additional disk space to install the application. If so, it is probably very large. Large products will tend *not* to be co-located with the operating system. Some examples include:

- OpenView, SD, and FrameMaker are large.
- UPS and xtetris are small.

4.3 Product Type Decision Tree

Once a product has been evaluated based upon the factors listed in the previous section, it may be classified using the following decision tree:

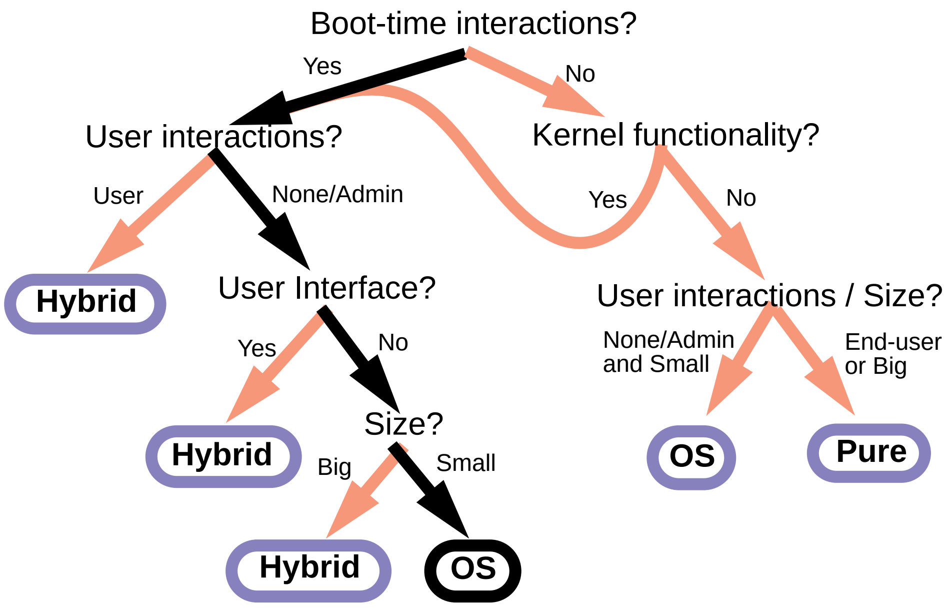


4.4 Examples

The examples contained herein are not meant to be accurate depictions of the actual products. In some cases, we have modified the actual product attributes for the purposes of having a good example.

4.4.1 OS Application - FDDI

FDDI is a networking product that is an extension of the operating system. It is transparent to end-users, but requires system administration. The product contains components that must start during system boot, a kernel library, header files, EISA configuration data, a command-line interface, and is small in size. Applying the decision tree, FDDI is an OS application product:



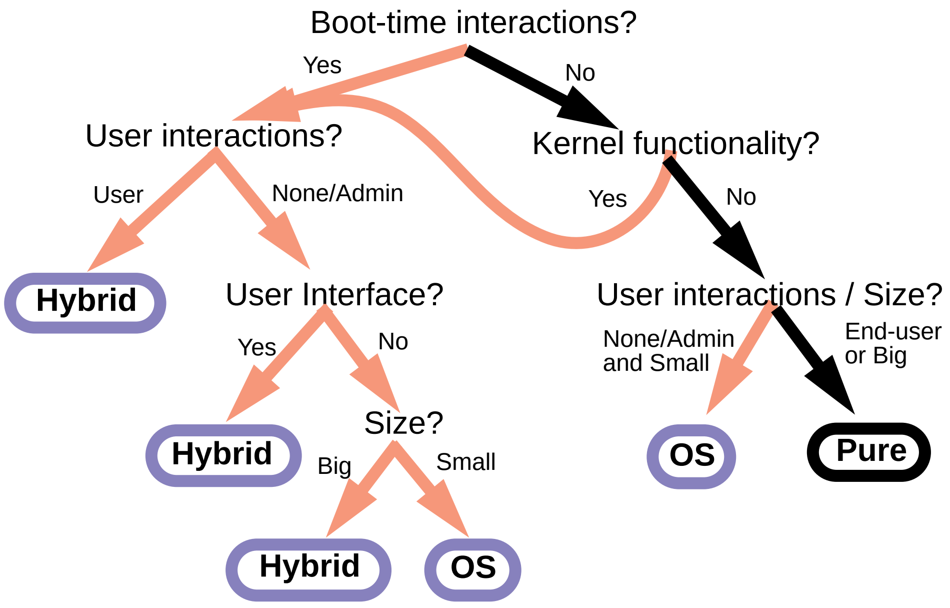
Following the guidelines for OS products located at the beginning of the section, the product files are loaded directly from the media into the OS sharing points during the installation process:

• **/usr/conf**	kernel libraries and headers
• **/usr/sbin and /usr/bin**	commands
• **/usr/newconfig/{path}**	default configuration files
• **/usr/lib/nls**	NLS message catalogs
• **/sbin/lib/EISA**	EISA configuration files
• **/usr/lib**	firmware download files and shared libraries

Once the product has been installed onto the system, product configuration occurs, first on the server and then on each subsequent client. During configuration, the files below **/usr/newconfig/{path}** are compared and copied into their respective locations. In the case of FDDI, there is only one file. FDDI has no runtime files that it creates below **/etc** or **/var**.

4.4.2 Pure Application - Glance+

Glance+ is a system performance monitoring tool. It is mainly used by the system administrator and has a text-based user interface. The kernel hooks are provided by HP-UX, so no kernel libraries or headers are delivered with the product. The product has no single-user mode commands, no system boot activities and is considered large in size. Applying the decision tree, Glance+ is a Pure application product:



Following the Pure application guidelines, the product files are loaded directly off the media into the product's own sharing point (this may be in either an OS or Application shared root):

• **/opt/Glance/bin**	user commands
• **/opt/Glance/lbin**	back-end commands
• **/opt/Glance/help**	help files
• **/opt/Glance/lib**	libraries
• **/opt/Glance/share/man**	man pages
• **/opt/Glance/newconfig**	default configuration files
• **/opt/Glance/lib/nls**	message catalogs

Once Glance+ has been installed below its sharing point, first the server and the subsequent clients are configured. During the swconfig process, the following locations are created to handle configuration, log and database files:

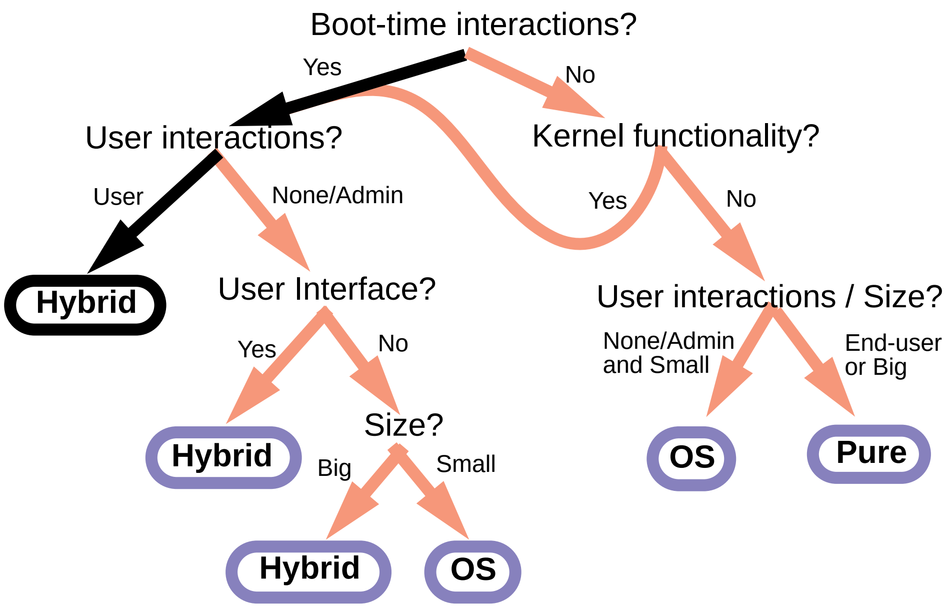
- **/etc/opt/Glance**
- **/var/opt/Glance**

Now Glance+ has been installed and configured on a system. During runtime execution, Glance+ creates the following files:

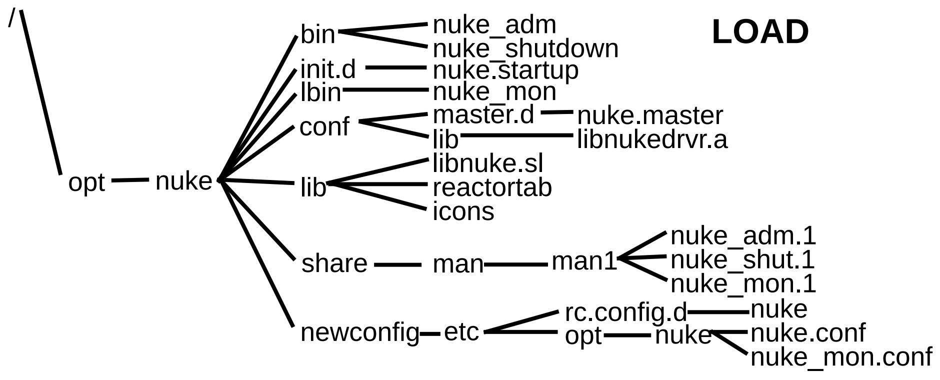
- **/var/opt/Glance/databases**
- **/var/opt/Glance/log**

4.4.3 Hybrid Application - Nuclear Reactor Controller

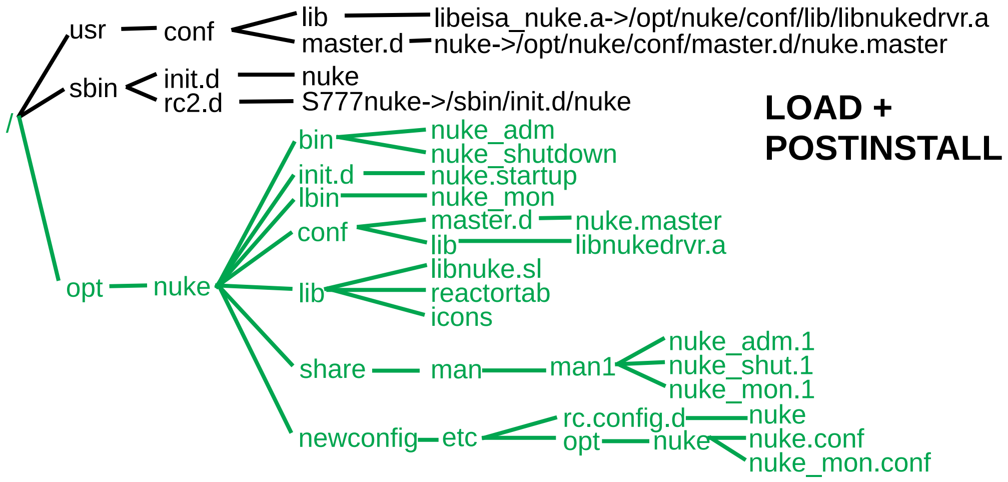
The nuclear reactor controller is a Motif-based application that monitors and controls nuclear power plants. It contains a kernel driver for an interface card to the reactor complex, starts a monitor daemon during system boot, is used by reactor operators on a daily basis and is large in size. Applying the decision tree, the Nuclear Reactor Controller is a Hybrid application:



Following the Hybrid product guidelines, the product is loaded from the media directly into its own sharing point below **/opt/nuke**. This may be physically located under **'/'** or below an alternate root, such as **/export/shared_roots/OS_700**, but not below an application shared root (the product installer determines the install root location). In either case, all product actions are performed relative to the value of SW_ROOT_DIRECTORY. The directory structure is depicted below:



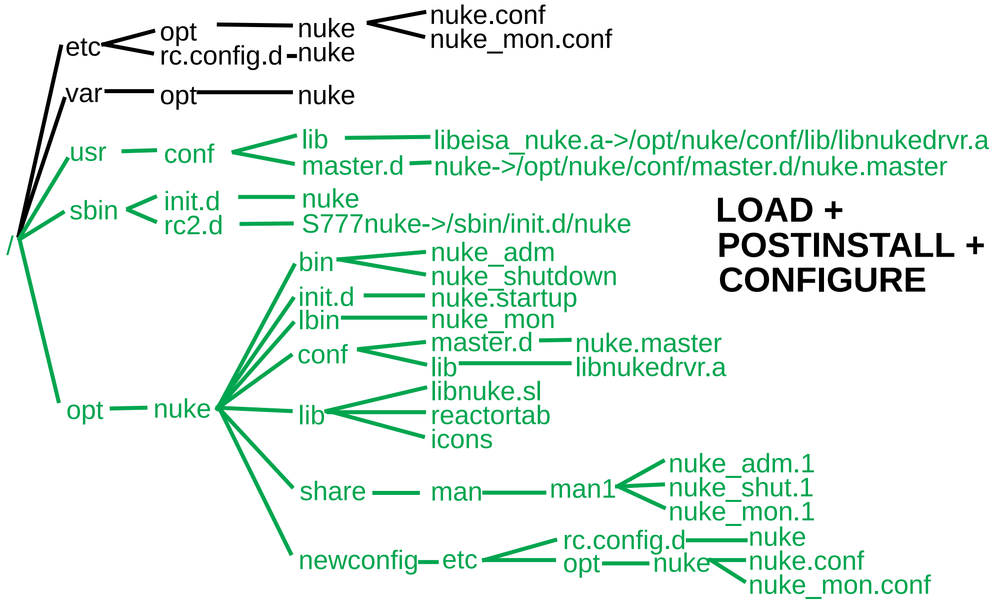
Once the product has been loaded into its sharing point, postinstall actions are used to populate the operating system sharing points, /usr and /sbin (the full contents of the OS sharing points is not shown, only the affected files are listed):



All postinstall actions are performed among sharing points within the same shared root indicated by the value of SW_ROOT_DIRECTORY. Because only servers have write access to sharing points, postinstall actions are performed only on servers and not diskless clients. The actions performed by the example postinstall include:

- Symlink kernel components from **/usr/conf** to the product's sharing point
- Transfer ownership of the master file component from the -KRN to the -RUN fileset
- If SW_DEFERRED_KERNBLD is unset, add kernel driver to system file
- Copy the startup/shutdown script from the product's sharing point to **/sbin/init.d**
- Create startup/shutdown links in **/sbin/rc*.d**

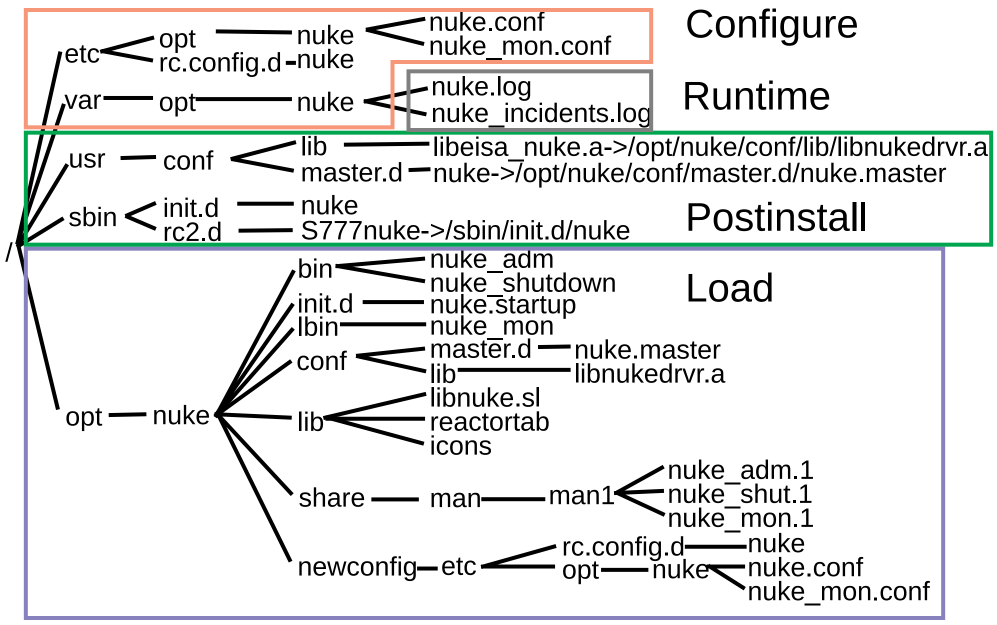
Once postinstall has been completed, system configuration occurs, populating the system's private file system:



The configuration process is run on both clients and servers. Therefore, configuration actions must not write to sharing points; write actions must be constrained to the system's private root. All of the actions are performed relative to **'/'** and do not require the SW_ROOT_DIRECTORY variable. Actions performed by the configure script include:

- configure default configuration files using newconfig_cp
- modify any system-specific configuration parameters (e.g. entries in **/etc/services**)
- create any private directories that may be needed during runtime.
- If SW_DEFERRED_KERNBLD is set, add kernel drivers to the system file.

After software configuration has occurred, the runtime execution of the system will create any necessary files below **/etc/opt** and **/var/opt**. The diagram below summarizes the file placement/creation actions as a function of the installation and runtime processes:



5. Product Removal Considerations

Developers are faced with a number of choices when designing the product deinstallation process. This section enumerates some points to assist with this process. These considerations are only meant to be a brief discussion. The reader is encouraged to consult "*Managing HP-UX Software*" for full explanations.

Software products are installed and deinstalled using SD. There are four control scripts a developer may utilize to assist with deinstalling the product: *checkremove, unconfigure, preremove* and *postremove*.

The *checkremove* control script is used to verify a fileset may be removed. It runs only on the server, where the software is installed. This type of script is used mainly by system critical components that should not ever be removed. Since SD resolves dependencies during removal as well, most filesets will not need one of these scripts.

The *unconfigure* control script executes both on the client and server. It must perform only system specific actions such as killing daemons, removing files not listed in the IPD, or un-doing configuration that was done by the configure script (e.g. removing an entry from **/etc/services**). This script reverses most actions performed by the *configure* control script. However, note that these scripts should not lower kernel tunables, cause a kernel rebuild, or change configuration settings associated with other products.

The *preremove* control script is executed just prior to SD removing the registered files from the system. Actions include removing any shared files placed by the *postinstall* control script or removing shared files not listed in the IPD. Some examples include removing the startup/shutdown script from **/sbin/init.d**, any startup/shutdown links in **/sbin/rc*.d** and any kernel library or master file links in **/usr/conf**.

Once SD has removed the files registered in the IPD, the *postremove* control script is executed. Most software products should be completely removed from a system at this point and should not require a *postremove* control script.

6. MiddleWare

A number of products are ‘building blocks’ that are layered between the operating system and other applications. We commonly call these middleware products. They typically have APIs that enable other applications to ‘register’ and take advantage of the middleware’s services. Examples include X11, HP-VUE and SoftBench.

Traditionally, applications have interfaced with middleware in a number of ways. For example, X products would locate their app-defaults, fonts, and bitmaps in a central directory where X would look to find these files. However, with HP-UX 10.0, these traditional directories below **/usr** are no longer available; applications must not write to the sharing point of other products. This constraint requires middleware application developers to provide the necessary and correct hooks that enable applications to register with the middleware product. A few options are discussed below; there may be others.

6.1 Registration Directories

With HP-UX 10.0, applications are not allowed to locate files in another application’s sharing point. One solution is to locate the registration directory outside the sharing point in the private root. For example, prior to 10.0, HP-VUE utilized the **/usr/vue/config** directory and allowed other applications to register their product by writing into this directory. With 10.0, HP-VUE has located the registration directory below **/etc/vue/config**, in the private root of the filesystem. Application’s wishing to register their products with HP-VUE will utilize the **/etc/vue/config** directory (among other things).

6.2 Programmatic Calls and Environment Variables

Other middleware applications may wish to take advantage of programmatic registration calls. Some middleware applications look in pre-determined directory locations to find registered product files. In addition, middleware applications may provide user and application-definable locations through environment variables and programmatic calls. For example, prior to 10.0, applications would locate their app-defaults below **/usr/lib/X11/app-defaults**. With 10.0, applications must locate their app-defaults within their product sharing points below **/opt/<appl>/lib/X11/app-defaults** and use a specific X11 call that tells X11 where to find an application’s app-defaults. (This option existed in prior releases; the only new thing is that everyone must use it. Specific usage may be found in the X11 change documents for 10.0.) Another example is NLS. Message catalogs were formerly located in a central location below **/usr/lib/nls**. With 10.0, each application will keep their message catalogs below their respective sharing points. To register with NLS, applications must utilize the NLS_PATH environment variable to tell NLS where to find the specific application message catalogs.

7. Test Configurations

In order to get maximum exposure for product functionality and installation/deinstallation, the following testing configurations are recommended:

- ☞ **S700 and S800 standalone -** Does not guarantee diskless-safe operation
- ☞ **S700 server with S700 client(s) -** Tests most diskless scenarios
- ☞ **S800 server with S700 client(s) -** Optimal testing configuration (alternate root install)

Developers should try to test software products in all the above configurations. However, if only one is possible, test the S800 server with S700 clients. This is the most optimal and will cover most functionality and installation/deinstallation scenarios.

The major difference between a S800 server and a S700 server is that the S700 server's operating system shared root (**/export/shared_roots/OS_700**) is a symbolic link to **'/'** because the S700 clients are able to share the OS with the server.

In the case of a S800 server, the **/export/shared_roots/OS_700** shared root must be populated by doing an alternate root install of a S700 operating system to that install root. Because S700 software on a S800 is alternate root installed, the control scripts which run as part of software installation are more fully exercised. In other words, errors which may not appear during an install to **'/'** may appear during an alternate root install because the use of the SW_ROOT_DIRECTORY variable was overlooked when designing the control scripts.

To simulate a S800/S700 environment with only S700 systems, do an alternate root install of S700 software to another shared root of your choice (e.g. **/export/shared_roots/TEST**) and add diskless clients using this shared root as the source for their OS software.

NOTE:	All software must be capable of correctly installing into an alternate root. The user of SD has complete control over the install root for software. Also, alternate root installs are useful apart from diskless uses.

8. Multiple Revisions & Relocatable Products

With HP-UX 10.0, the SD product replaces the DUI product (distributed update and install). SD has features which, coupled with proper software design, enable a product to support multiple revisions. However, developers must account for many details and test their designs prior to releasing a product that supports multiple revisions. Some important considerations are discussed below.

8.1 Installed vs. Configured

When supporting multiple revisions, developers must understand *installation* and *configuration*. Supporting multiple revisions of a product *installed* does not necessarily imply that more than one version is *configured* to run on the system. Installation refers to the software being loaded onto a system; configuration refers to the necessary (and version specific) configuration/log files being in place for product execution.

For example, the Nuclear Controller product may wish to support more than one version being *installed* on a system at once. This requires that each version of the product have its own sharing point; each defined by a different version number. However, because Nuclear Controller is an extension of the operating system and must be co-resident with one, only one version of Nuclear Controller may be *configured* per system. This is true because the same log and configuration files are accessed by the multiple versions.

Unless the versions can distinguish among different configurations, the product probably cannot support multiple versions being *configured* on a system. However, the product may support more than one version being *installed*. This may be advantageous when flipping between versions during testing. However, there are still other considerations that must be resolved before such a goal can be achieved.

8.2 Shared Libraries

Products or commands using using shared libraries contain hard-coded paths embedded in the executables which reference the libraries. These hard-coded paths are determined when the executable is linked with the shared libraries. If a product/command depends on a hard coded path to a library, and the customer has relocated the product which contains that library (it may be your own product), the executable will fail unless there is a mechanism to resolve the path to the relocated shared library.

8.3 Finding Relocated Commands

Some products contain commands and other files that the product itself, or a dependee product, may expect to find in a certain location. Part of multiple revision support involves product relocation. If a product is relocated during installation, how will the product (or dependee products) find the new location of commands and other files? Any design for multiple revision support must take this into account.

8.4 Testing

Once a multiple revision support design has been completed, there are many different testing scenarios that must be exercised before releasing the product. The complete matrix is beyond the scope of this brief overview. However, a product is not ready to claim multi-revision support or relocatablility until it has been thoroughly tested to ensure the new flexibility does not break an installed system.

9. Q&A

Question: **What are the differences among /etc, /var & /tmp directories?**

Answer: The *HP-UX 10.0 File System Layout Whitepaper* describes each of these directories in detail, and additional information may be found there. **/etc** contains configuration files that are ‘fixed’ in length and content. Files in **/etc** are usually editable by users or system admins to affect the behavior of the system. **/var** contains variable length files, either temporary or configuration in nature. These files are usually not edited by customers. **/tmp** is mainly used by the operating system and does not containing any pesistent temporary files associated with users or other software products.

Question: **May applications continue to use system calls, such as mktmp(), that create arbitrary temporary files?**

Answer: Yes. In most cases, these library calls still create temporary files in their previous locations and do not accept arguments that enable temporary files to be created below application specific temporary directories, such as **/var/opt/<appl>/tmpfile**. In most cases, these temporary files are created below **/tmp**. Consult the appropriate documentation for specific library calls.

Question: **How do I get startup/shutdown script sequencing numbers?**

Please see the *HP-UX 10.0 File System Layout Whitepaper* for details.

Question: **What if I do not care about sequencing?**

If you have a software component that you would like to start during system boot, but do not necessarily care how/when it gets started, the best solution is to incorporate it in with the rest of the startup model. This is the most reliable and recommended solution. Consult the *HP-UX 10.0 File System Layout Whitepaper* for details on choosing sequencing numbers for the “don’t care” situation.

Question: **What if my product does not support diskless?**

Answer: If a product is a pure application and loads in **/opt/<product>**, the way to not support diskless is to not specify a share_link attribute in the PSF.

If a product has components in one of the OS sharing points (**/usr** or **/sbin**), these components will be seen by diskless clients that use this OS. The software itself will have to be diskless-smart, both in the execution of its system startup script (if it has one), and in the execution of the product itself.

In general, all products should function correctly in both diskless and standalone environments. Products that do not support diskless configurations will be rare.

Question: **How can I guarantee that my Hybrid or OS application product is co-resident with an operating system in a shared root?**

Answer: To guarantee that a product is loaded into a shared root where an operating system is present, the product can define a corequisite on the OS-Core product. See “*Managing HP-UX Software*” for further details about corequisites.