This chapter contains the following sections that describe the Software Depot (SD) creation techniques. These techniques are necessary for an HP-UX Driver Developer to package a driver for distribution. The overall process is:

- Creating a Package
  - Software Depot Overview
  - Designing a Software Depot Structure
  - Selecting Product Install Directory on the Host
  - Writing a PSF
  - Writing Control Scripts (if required)
  - Packaging the Software Depot Components
  - Registering the Software Depot (Package)
- Installing the Software Depot
- Managing the Software Depot

Some Terms and Attributes used in this chapter are defined in the Glossary at the end of this Guide.
Software Depot Overview

This section focuses on the basic Software Depot (SD) packaging techniques that a HP-UX Driver Developer needs to know, to package a driver for distribution.

After a software application is developed, files are taken from the programmer’s environment and “Integrated” for distribution. To ensure ease of installation, maintenance and de-installation, software developed on HP-UX is often distributed as Software Depots. A Software Depot (SD) is a directory location on the local or remote host that is used as a “gathering place” for software products. It is a customizable source of software used for direct installations by the host on the network. The SDs are created using the Software Distributor. The C prepares these software files by organizing them into specific products, subproducts and fileset structures. It also uses special information files (see Creating a Product Specification File (PSF)) that are used to help other commands identify, distribute and manage the application. After it is organized, the software is then “mastered” or copied onto CD-ROMs or tapes for further distribution to Users or Customers. The resulting package can also be made “network accessible” to Users.

There are two types of Depots:

1. Directory Depot
   Software in a Directory Depot is stored under a normal directory on the file system (usually /var/spool/sw). This software is in a hierarchy of subdirectories and filesets organized according to a specific media format. A Directory Depot can be “writable” or read-only. When using the Software Distributor software management commands (see "jkg"), refer to a directory depot via its top-most directory. In a CD-ROM depot, this directory would be the CD-ROMs “mount point”.

2. Tape Depot
   Software in a Tape Depot is formatted as a tar archive. Tape depots such as cartridge tapes, DAT and 9-track tape are referred to by the file system path to the tape drive's device file. Software in a tape depot must first be transferred to a directory depot before it can be “pulled” by other hosts on the network. A tape depot can be accessed by only one command at a time.

A depot usually exists as a directory location (that is, a directory depot). Therefore, a host may contain several depots. For example, a designated SD server on the network might contain a depot of word processing software, a depot of CAD software, and a spreadsheet software depot, all on the same server.

The SD provides a powerful set of tools for centralized HP-UX software management. Software Depot has a set of commands which are included with the HP-UX operation system. These commands have the following Tasks distributed across them:

- Packaging the Software as a Depot
- Installing Depots
- Managing Depots
- Removing Depots

When working with SD the methods and objects are largely abstract, making it sometimes difficult to visualize objects, processes, and what is actually happening in the system.

Depots contain software objects such as bundles, products and filesets. A depot is similar to an operating system’s root area. Software objects exist only inside the depot or “root”. Objects can only be viewed in the depot with SD commands.

SD commands work on a hierarchy of software objects — bundles, products, subproducts and filesets, that make up the applications or operating systems that are to be managed.
**Bundles**

A collection of filesets, possibly from several different products, that are “encapsulated” for a specific purpose. Bundles can be stored in software depots and copied, installed, removed, listed, configured and verified as single entities. All HP-UX OS software is packaged in bundles. Bundles, since they are groups of filesets, are not necessarily supersets of products. Customer creation of bundles are not supported.

**Products**

A collection of subproducts (optional) and filesets. The SD commands maintain a product focus but still allow specifying of subproducts and filesets. Different versions of software can be defined for different platforms and operating systems, as well as different versions (releases) of the product itself. Several different versions could be included on one distribution media or depot.

**Subproducts**

Subproducts are used to group logically related filesets within a product if the product contains several filesets.

**Filesets**

Filesets include all of the files and control scripts that make up a product. They are the smallest manageable (selectable) SD software object. Filesets can only be part of a single product but they could be included in several different HP-UX bundles.

Bundles are designed to provide customers with a single installation unit to install when they purchased software products, such as the ANSI/C compiler. Bundles can be used to provide logical groupings for functionality, such as the “web server”.

Bundling the products makes it easy to treat several filesets as a single entity using the SD commands (discussed later). By specifying a bundle, all filesets under that bundle are automatically included in the operation. In general, performing a single operation on a bundle is the same as performing it individually on all the filesets listed in the bundle.

It is important to note that bundles do not eliminate the ability to “pick and choose” the products and filesets to install.

**NOTE**  
A depot is often confused with software objects. Depots can not be installed or moved like a software object. A depot can not exist without at least one software object residing in it.
SD Structure Capabilities

The structure of a Software Depot is shown in Figure 19-1, “SD Structure.”

Figure 19-1   SD Structure

SD can be viewed as 4-tier/3-tier software structure as shown. The bottom tier are “Filesets”, the set of files to be packaged in the depot. A fileset is the atomic element of the SD. Each fileset must appear in at least one, but only one SD-Product. SD-Filesets can only be installed or removed by the user as a whole entity. It is not possible (via SD) to install or remove only a select set of files from a given SD-Fileset. Files which are used for the same basic purpose should be grouped together in a single SD-Fileset. Likewise, files which are not used for the same basic purpose should be kept apart. The top tier are “Products” — a collection of filesets or (optional) subproducts and control scripts. SD-Product is the fundamental component in SD:

- Packaging is done at the SD-Product level.
- Many attributes are specified at the SD-Product level.
- Filesets only exist within an SD-Product.

SD-Filesets which are for the same functionality should be in the same SD-Product.

The middle tier are Subproducts; if an SD-Product contains several filesets, subproducts are used to group logically related filesets.

Bundles are another tier in the software structure in Figure 19-1, “SD Structure.” They are collections of filesets, possibly from several different SD-Products that are encapsulated for a specific purpose. Bundles provide an alternate way for users to view or select software. The SD structure can be viewed as:

Bundle ---> Products ---> Subproducts ---> Filesets

An SD structure can be any of these combinations:

Bundle1--->Product1---->SP1---->Fileset1 and Fileset2
Bundle2--->Product2---->SP4---->Fileset1 and Fileset2
Bundle1--->Product1---->SP3---->Fileset2 and Fileset1
and similarly for SP2 and SP5.

Product1.Fileset1 != Product2.Fileset1

**Software Objects Nomenclature**

**SD-Bundles**  
SD-Bundle names must be a maximum 16 characters with no underscores or white spaces. The following characters are not allowed in any of the above software objects names:

#, ?, / [], *, &

**SD-Products**  
SD-Product names must be sixteen or fewer characters. They should specify the type of functionality that they contain. Underscores are not permitted in SD-Product names. SD-Product names should be written in mixed case, with significant letters capitalized, and contain no white space. Acronyms may be entirely capitalized. Like SD-Fileset names, SD-Product names should be descriptive and unique. No two SD-Products may have the same name.

**SD-Subproducts**  
SD-Subproducts names must be sixteen characters or fewer in length and should accurately describe the SD-Subproduct's contents. The name should be descriptive enough that a user selecting software to install or remove can make an informed choice regarding the SD-Subproduct based solely on its name. No user should be forced to look at the underlying SD-Filesets when making this decision. The recommended format for SD-Subproduct names is the same as that for SD-Product.

**SD-Filesets**  
SD-Fileset's names must be entirely capitalized. Name must be unique. The maximum length of SD-Fileset names is 14 characters.
Software Depot Package Components

The SD consists of four major components; Packager, Controller, Daemon and Agent. All except the Agent are installed in the directory `/usr/sbin/.

1. Packager (swpackage) — The swpackage is a standalone utility that takes a Product Specification File (PSF), a set of Control Scripts, and the files the user wants to deliver and packages them into an SD depot. A depot can be either a directory tree or a tar archive. This depot can then be distributed to customers, who then install it on their systems using the swinstall command. While the depot is simply a directory structure, it may not install using `cp` or `tar`. This is run as a command line user interface only.

2. Controller — Sometimes called the Integrated Controller, better known to SD customers as those commands used to manage packaged software; `swaci`, `swconfig`, `swcopy`, `swdepot`, `swinstall`, `swjob`, `swlist`, `swreg`, `swaremove` and `swverify`. These commands are delivered as hard links pointing to a single binary. Controller commands are invoked by users to initiate SD actions. A Graphical User Interface (GUI) is available for `swcopy`, `swinstall`, `swlist -i`, and `swaremove`.

3. Daemon — The (/usr/bin/swagent) coordinates Controllers and Agents, acting as an intermediary. When a controller command, such as `swinstall`, is executed, the swagentd is contacted via a Remote Procedure Call (RPC). The swagentd then forks, and the child process issues an expects call to run the Agent `swagent`.

4. Agent — The (/usr/bin/swagent) performs the majority of the work done by SD. The program installs and removes the software. Agents perform “source” and “target” activities. A source agent is a `swagent` process which is reading a software source, such as a depot, while a target agent is operating on a target. A target can be a root filesystem or a depot.

Product Specification File

The master file is where the bundle configuration (attributes) information exists. It specifies the revision of the product, architecture, dependencies, installation path, and other attributes. The details of these and other attributes are explained in detail in the following sections. All the characteristics of a given product are described in a PSF file.

The PSF contains attribute information for all the software objects and it has a structure accordingly:

```plaintext
#Vendor information
  vendor
    Vendor Attributes
  #end vendor
#Category information
  Category
    Category Attributes
  #end category
#Bundle information
  Bundle
    Bundle Attributes
  #Bundle
#Product information
  Product
    Product Attributes
  #Subproduct information
  Subproduct
    Subproduct Attributes
  #Subproduct
    Subproduct2
    Subproduct2 Attributes
```
The Category information is “optional” and need not be used. An SD can be packaged without Category information. It is used only for selection mechanism.

The package information is shown in Figure 19-2, “SD Package.”

<table>
<thead>
<tr>
<th>VENDOR</th>
<th>CATEGORY</th>
<th>SD-BUNDLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Attributes</td>
<td>Attributes</td>
</tr>
<tr>
<td>– Object Definition</td>
<td>– Object Definition</td>
<td>– Object Definition</td>
</tr>
<tr>
<td>– Information</td>
<td>– Information</td>
<td>– Information</td>
</tr>
<tr>
<td>– Control</td>
<td>– Control</td>
<td>– Control</td>
</tr>
</tbody>
</table>

**SD-PRODUCT**

- Attributes
  - Object Definition
  - Information
  - Control
- Control Scripts
- SD-Subproduct
  - Attributes
  - Object Definition
  - Information
  - Control
  - Control Scripts
- SD-Fileset
  - Attributes
  - Object Definition
  - Information
  - Control
  - Control Scripts
  - Files

**Figure 19-2 SD Package**
**SD Objects Attributes Classification and Flow**

**Vendor Attributes:**
- Object definition — tag, title, and so forth.
- Information attributes — such as Description.

**SD-Category Attributes:**
- Object definition — tag, title, and so forth.
- Information attributes — such as Description.

**SD-Bundle Attributes:**
- Object definition — attributes like tag, title, and so forth.
- Information attributes — such as Description.
- Control attributes — OS, OS version, release, architecture, contents, and so forth.

**SD-Product Attributes:**
- Object definition — attributes like tag, title, and so forth.
- Information attributes — like Description.
- Control attributes — such as OS, OS version, release, and architecture.
- Control scripts — preinstall, postinstall, and so forth.
- Subproducts and Filesets.

**SD-Subproduct Attributes:**
- Object definition — like tag, title, and so forth.
- Information attributes — like Description.
- Control attributes — like contents.

**SD-Fileset Attributes:**
- Object definition — Definition attributes like tag, title, and so forth.
- Information attributes — Informative ones like Description.
- Control attributes — OS, OS version, release, and architecture.
- Control scripts — preinstall, postinstall, and so forth.
- Files — sources, binaries and object files to package.

**Policies of the PSF Attributes**

A PSF is a master file which contains configuration information (attributes) set accordingly. The SD commands operate based on the values set against these attributes.

The attributes are classified into:

- **Optional attributes**
  - Optional and need not be used, although they still can be included by the user.

- **Required attributes**
  - These are not optional.
HP software suppliers who need to structure and package software for distribution across SD require some policies for SD structuring and SD attributes. These policies ensure that customers will be more effective in using the software distribution tools if the software they are dealing with is consistently named and structured.

**Vendor Attributes**

The following attributes are to be included in the PSF file and their policies:

- **Tag**
  - Status: Required
  - Description: The short name for the vendor (distributor) of the software.

- **Title**
  - Status: Required
  - Description: The longer name for the vendor of the software.

- **Description**
  - Status: Required
  - Description: A file or text string describing the vendor (displayed only when the user asks to see a description of the software and then selects the vendor description for view).

**NOTE**

The vendor attributes will be that of the Distributor of the product. For example, if company A develops the product and HP distributes it, the vendor attributes will be HP's.

**Category Attributes**

The following are the category attributes and their policies:

- **Tag**
  - Status: Required
  - Description: The identifier for the category object. In general, only the choices listed should be used. There is a one-to-one relationship between category.tag and category.title attributes as explained under the category.title attribute.

  **Policy:**
  - **OrderedApps**
    - Specify a product that a customer can obtain from HP.
  - **TrialUseApps**
    - Trial version of an available product.
  - **Patch**
    - Software automatically set if the is_patch attribute is set to “true”; it is reserved for patches.

The following category tags are reserved for patches:

- defect_repair
- hardware_enablement
- enhancement
- general_release
- special_release
- trial_patch
- beta_release
- manual_dependencies
- critical
Creating a Software Depot
Software Depot Package Components

— panic
— halts_system
— corruption
— memory_leak

Title
Status: Required

Description: The one line, detailed name for the category. Notice that there is a one-to-one relationship between category.tag and category.title attributes. Use the category.title values exactly as defined:

Ordered Software Applications When category.tag is OrderedApps.
Trial Use Software Applications When category.tag is TrialUseApps.
Ordered HP-UX Bundles When category.tag is OrderedHP-UX.
Additional HP-UX Functionality When category.tag is HPUXAdditions.

These category.titles are for patches and are controlled by the patch creation tools:
— Use “” when category.tag is beta_release.
— Fix corruption when category.tag is corruption.
— Fix a critical defect when category.tag is critical.
— Provide defect repair when category.tag is defect repair.
— Provide enhancement when category.tag is enhancement.
— Fix a hang or abort when category.tag is halts_system.
— Provide new hardware support when category.tag is hardware_enablement.
— General release patch when category.tag is general_release.
— Fix a memory leak when category.tag is memory_leak.
— Requires manual review of dependencies when category.tag is manual_dependencies.
— Fix a system panic when category.tag is panic.
— Use “” when category.tag is special_release.
— Use “” when category.tag is trial_patch.

Description
Status: Optional

Description: A file or text string providing a more detailed description of the category. Use only the text strings listed in the Policy since the category.description is “global”, that is, a particular category.description applies to all objects in the depot/on the media with the corresponding category.tag.

Policy:
Ordered Software Applications When category.tag is OrderedApps.
Trial Use Software Applications When category.tag is TrialUseApps.
SD-Bundle Attributes

Tag
Status: Required
Description: The identifier for the SD-Bundle (appears in the Software Selection window as the basic name for the SD-Bundle) — 16 characters maximum. The tag should be unique and descriptive.
Policy: A bundle tag requires one or more characters from “A–Z”, “a–z”, “0–9”, including the first character. No white space characters are allowed. The directory path character “/” is not allowed. The following SDU and shell metacharacters are not allowed:

Title
Status: Required
Description: The one-line, more detailed name for the SD-Bundle (appears on the Software Selection window and in the swlist one-liner) — 80 characters maximum. Use this chance to expand upon the sixteen character name of the SD-Bundle.
Policy: This attribute will be seen rather often, so it should be carefully chosen to clearly and concisely tell the user what the SD-Bundle is.

Description
Status: Required
Description: A file or text string (limited to 8 Kb) describing the SD-Bundle (displayed only when the user asks to see a description of the software and then selects this description for view). The free-form ASCII style of this attribute allows for detailed description of the software.
Policy: For maximum readability, each line should be limited to seventy-two characters or less. Tabs should not be used in this file/text string.

Revision
Status: Required
Description: The revision (release) number of the SD-Bundle.
Policy: The revision number of a SD-Bundle is of the form major:minor:release(path).

Architecture
Status: Required
Description: The target systems on which this SD-Bundle is to be installed and will run.
Policy: There are three parts to the architecture attribute; os name, os release, and os bits. The attribute is structured like:

<os name>_<os release>_<os bits>

The “os name” portion of the architecture attribute is always HP-UX.
The “os_release” portion of the string must match the targeted OS for which the SD-bundle is ignited.
The “os_bits” portion specifies the os bit configuration the SD-bundle will run on.

- 32 runs only on an OS in 32-bit mode
- 64 runs only on an OS in 64-bit mode
- 32/64 runs on either

For example:
- HP-UX_B.11.11_32 earliest OS is 11.11, 32-bit only
- HP-UX B.11.11_64 earliest OS is 11.11, 64-bit only
• HP-UX_B11.11_32/64 earliest OS is 11.11, 32 or 64 bit

Whenever architecture is defined for a software object, the 3 related attributes “machine_type”, “os_name” and “os_release” must be defined as well. These compatibility attributes are the responsibility of the packager.

\[\text{os_name}\]  
Status: Required
Description: OS’s on which the SD-Bundle will run:
\[\$(\text{uname} -s):$(\text{getconf CS\_KERNEL\_BITS})\]
The “getconf CS\_KERNEL\_BITS” indicates the bit configuration of the running kernel. Possible results are “32” (if the running kernel is 32 bit) and “64” (if the running kernel is 64 bit, and therefore supports execution of 32-bit applications also)

Policy:
• HP-UX — SD-Bundle runs on both 32-bit and 64-bit operating systems. Most 32-bit SD-bundles would use this specification.
• HP-UX:32* — SD-Bundle runs on 32-bit operating systems only.
• HP-UX:*64 — SD-Bundle runs on 64-bit operating system only.

\[\text{os_release}\]  
Status: Required
Description: OS releases on which the SD-Bundle is to be installed and will run: \[\$(\text{uname} -r)\].
Policy:
• B.11.11 means compatible with 11.11 only.
• ?.11.11 means compatible with 11.11 only

\[\text{os_version}\]  
Status: Optional
Description: OS versions on which the SD-Bundle will run: \[\$(\text{uname} -v)\]. This attribute is not used. If it is set, it must be set to “*”.
Policy: * — allow all versions.

\[\text{Category\_tag}\]  
Status: Required (If Category object was defined in the PSF).
Description: Visible only from swlist, and so forth, when users specify a software specification to select a class of SD-Bundles from the command line. In general, only the following choices should be used. There is a one-to-one relationship between bundle.category_tag and bundle.category_title attributes as explained under the bundle.category_title attribute.
Policy:
- OrderedApps Specify a product that a customer can obtain from HP.
- TrialUseApps Trial version of an available product.
- HPUXAdditions Additional HP-UX functionality SD-Bundle.

\[\text{vendor\_tag}\]  
Status: Required
Description: The short name for the vendor (distributor) of the software. This should match the vendor.tag attribute.
Category_title Status:

Obsolete if a category object is defined. Use category object instead of this attribute.

Description: The one-line, more detailed name of the SD-bundle category tag. Note that there is one-to-one relation between bundle.category_tag and bundle.category_title.

Policy:

- “Ordered Software Applications” when bundle.category_tag is “Ordered Apps”
- “Trial Use Software Applications” when bundle.category_tag is “TrialUseApps”
- “Additional HP-UX Functionality” when bundle.category_tag is “HPUXAdditions”

Contents Status: Required

Description: Describes the software contained in the SD-Bundle.

Policy: Should be a list of SD-Filesets with their software specifications, but can be a list of SD-Products with their software specifications. Each software specification should be fully qualified. This means each software specification should explicitly list the revision, architecture and vendor. The revision listed in any software specification is always the product revision. The software specification should not include the SD-Fileset level architecture (fa=) unless the SD-Bundle contains a subset of multi-stream SD-Fileset pairs, if applicable. For example, a typical bundle.contents would be:

Prod1.FILESET1,r=B.11.11,a=HP-UX_B.11.11_32,v=HP, fa=HP-UX_B.11.11_64

SD-Product Attributes

Tag Status: Required

Description: The identifier or name for the SD-Product (appears on the Software Selection window as the basic name for the SD-Product) — sixteen characters maximum.

Policy: SD-Product names must be sixteen or fewer characters. They should succinctly specify the type of functionality that they contain. It is recommended that SD-Product names be written in mixed case, with significant letters capitalized, and contain no white space. Acronyms can be entirely capitalized. Underscores are not permitted in SD-Product names.

Title Status: Required

Description: The one-line, more detailed name for the SD-Product (appears on the Software Selection window and in the swlist one-liner) — 80 characters maximum. Use this chance to expand upon the 16 character tag of the SD-Product.

Policy: This attribute will be seen rather often, so it should be carefully chosen to clearly and concisely tell the user what the SD-Product is.

Description Status: Required

Description: A file or text string (limited to 8Kb) describing the SD-Product (displayed only when the user asks to see a description of the software and then selects this description for view). The free-form ASCII style of this attribute allows for detailed description of the software.

Policy: For maximum readability, each line should be limited to 72 characters or less. Do not use tabs in this file/text string.
Revision Status: Required

Description: The revision (release) number of the SD-Product.

Policy: Because each ISU/ISD product has its own revision cycle, there is not a definitive policy for specifying product revision. ISU partners are free to manage their revision numbers, within the given constraints, in a way that works best for each product.

The suggested revision number of a SD-Product is of the form major:minor:release\(\text{path}\) each product.

---

**NOTE**

If the product is part of a bundle then, the “revision” attribute has the same value as that of the Bundle.

---

architecture Status: Required

Description: The target systems on which this SD-Product is to be installed and will run. Summarizes the supported hardware and operating systems.

\(<\text{os name}>_<\text{os release}>_<\text{os bits}>\)

The os name portion of the architecture attribute is always HP-UX.

The os release portion should be the minimum OS revision necessary to run the software.

---

**NOTE**

If the SD-Product can run only on 11.11 and later, the os\_release portion must be B.11.11.

---

Policy: There are three parts to the architecture attribute: “os name”, “os release”, and “os bits”.

---

**NOTE**

If the product is part of a bundle then, the architecture attribute has the same value as that of the Bundle.

---

The attribute is structured like:

The “os\_bits” portion specifies the OS bit configuration the SD-product will run on:

- 32 — runs only on an OS in 32-bit mode
- 64 — runs only on an OS in 64-bit mode
- 32/64 — runs on either

For example:

- HP-UX\_B.11.11\_32 — earliest OS is 11.11, 32-bit only
- HP-UX\_B.11.11\_64 — earliest OS is 11.11, 32-bit or 64-bit
- HP-UX\_B.11.11\_32/64 — lowest common denominator for multi release compatible software on 11.x (32 or 64-bit OS)

Whenever architecture is defined for a software object, the 3 related attributes machine\_type, os\_name and os\_release must be defined as well.
os_name  Status: Required
Description: OS releases on which the SD-Product will run:
$(uname -s):$(getconf CS_KERNEL_BITS)

The “getconf CS_KERNEL_BITS” indicates the bit configuration of the running kernel which are “32” (if running kernel is 32-bit) and “64” (if the running kernel is 64-bit, and therefore supports 32-bit also.)

Policy:
- HP-UX — SD-product runs on both 32-bit and 64-bit. Most 32-bit product would use this.
- HP-UX:32* — SD-product runs on 32-bit OS only.
- HP-UX:*64 — SD-product runs on 64-bit OS only

NOTE If the product is part of a bundle then, the os_name attribute has the same value as that of the Bundle.

os_release  Status: Required
Description: OS releases on which the SD-Product is to be installed and will run:
$(uname -r).

Policy:
- ?.11.* — Any 11.0 release
- ?.11.[01][01] — 11.00 through 11.11
- ?.10.*|?.11.* — Any 10.x and 11.x release
- ?.11.00 — 11.00 only

NOTE If the product is part of a bundle then, the os_release attribute has the same value as that of the Bundle.

os_version  Status: Optional
Description: OS versions on which the SD-Product will run: $(uname -v). This attribute is not used. If it is set, it must be set to “*”.

Policy: * — Allow all versions

directory  Status: Required
Description: The default, absolute pathname of the directory which is the root of the file system tree under which the SD-Product will be installed. Can be re-mapped by the user if is_locatable attribute is true.

Policy: Use a standard type of location (like /opt/myprod); do not include a trailing slash like (/opt/mywrongslash/).

is_locatable  Status: Required
Description: If set to true, users can install the SD-Product into an alternate product directory.
Policy:

- FALSE — SD-Product must be installed into the path specified by product.directory.
- TRUE — SD-Product may be installed to an alternate location.

NOTE: This attribute must be explicitly set to either true or false. The default is true.

category_tag  Status: Required
Description: The short name for the category of the software. This attribute associates the SD-Product with the standard category object.
Policy: This should match the proper category.tag attribute.

description_tag  Status: Required
Policy: This should match the proper description.tag attribute.

vendor_tag  Status: Required
Description: The short name for the vendor of the software.
Policy: HP (or whatever company is providing ISV software). This should match the vendor.tag attribute.

machine_type  Status: Required
Description: Machine types on which SD-product will run:

$(uname -m):$(getconf CS_HW_CPU_SUPP_BITS)

The “getconf CS_HW_CPU_SUPP_BITS” indicates the OS bit configuration that the hardware supports. Possible results are “32” (only 32-bit HP-UX is supported), “32/64” (both 32-bit and 64-bit HP-UX is supported) and “64” (only 64-bit HP-UX is supported)
Policy:

- * — if not concerned with machine type
- *.32* — if 32-bit capable hardware is required
- *.64* — if 64-bit capable hardware is required

preinstall  Status: Optional
Description: Pathname for the install pre-load SD-Product control script. The script is run by swinstall during the execution phase before loading the software files.

unpreinstall  Status: Optional
Description: Pathname for the undoing install preload SD-Product control script. The script is run during the swinstall load phase if it is initiated. An unpreinstall script should undo any operation taken by the preinstall script.
Policy: unpreinstall scripts need to be carefully crafted and extensively tested.

postinstall  Status: Optional
Description: Pathname for the install post-load SD-Product control script. The script is run by swinstall during the Load phase after loading the software files.

unpostinstall  Status: Optional
Description: Pathname for the undoing install post load SD-Product control script. The script is run during the swinstall load phase if recovery is initiated. An unpreinstall script should undo any operation taken by the postinstall script.
configure Status: Optional
Description: Pathname for the configure for use SD-Product control script. The script is run
by swinstall or swconfig to configure the host for the software, or to configure the
software for host-specific information. Configure scripts are run by swinstall for all
SD-Products after all SD-Products have completed the Load phase. The swconfig command
can also be used to rerun configure scripts that failed during a normal install.

verify Status: Optional
Description: Pathname for the verify integrity SD-Product control script. The purpose of the
verify script is to check for the correctness of the product or fileset installation and
configuration. It is run by the swverify command.

unconfigure Status: Optional
Description: Pathname for the configure undo SD-Product control script. The script is run
by swconfig or swremove to undo host or software configuration originally performed by a
configure control script.

preremove Status: Optional
Description: Pathname for the remove pre-remove SD-Product control script. The script is
run by swremove during the Remove phase just before removing files.

postremove Status: Optional
Description: Pathname for the remove post-remove SD-Product control script. The script is
run by swremove during the Remove phase just after the files have been removed.

SD-Subproduct Attributes

Tag Status: Optional
Description: The identifier for the SD-Subproduct (appears in the Software Selection
window as the basic name for the SD-Subproduct) — sixteen characters maximum.

There is a one-to-one relationship between the “subproduct tag” and “subproduct title”
attributes as explained under the “subproduct title” attribute. If a non-standard
SD-Subproduct name was created for the SD-Product, the “subproduct tag” attribute should
be that name.

SD-Subproducts and SD-Filesets within the same SD-Product can not have the same tag.

Policy: Standard subproduct tag names:

- ManualsByLang Localized manual pages and product documentation.
- Help On-line help.
- HelpByLang Localized on-line help.
- MessagesByLang Localized messages.
- Demonstration Demonstration of the product.
- Development Software development SD-Filesets
- ReleaseNotes Separate release notes SD-Fileset(s)
Title 
Status: Optional

Description: The one-line, more detailed name for the SD-Subproduct (appears in the Software Selection window) — 80 characters maximum. Use this chance to expand upon the standard name of the SD-Subproduct.

There is a one-to-one relationship between the “subproduct tag” and “subproduct title” attributes. If creating a non-standard SD-Subproduct name for the SD-Product, also develop a corresponding non-standard entry for the subproduct.title attribute.

Policy: Standard subproduct titles:

- Manual Pages and Documentation
  When subproduct.tag is “Manuals”.
- Manual Pages and Documentation in Multiple Languages
  When subproduct.tag is “ManualsByLang”.
- On-line Help
  When subproduct.tag is “Help”.
- On-line Help in Multiple Languages
  When subproduct.tag is “HelpByLang”.
- Messages in Multiple Languages
  When subproduct.tag is “MessagesByLang”.
- Demonstration of the Product
  When subproduct.tag is “Demonstration”.
- Software Development
  When subproduct.tag is “Development”.
- Product Release Notes
  When subproduct.tag is “ReleaseNotes”.

Description 
Status: Optional

Description: A file or text string (limited to 8Kb) describing the SD-Subproduct (displayed only when the user asks to see a description of the software and then selects this description for view). The free-form ASCII style of this attribute allows for detailed description of the software.

Policy: For maximum readability, each line should be limited to 72 characters or less. Do Not use tabs in this file/text string.

SD-Fileset Attributes

Tag 
Status: Required

Description: The identifier for the SD-Fileset (appears in the Software Selection window as the name for the SD-Fileset) — 14 characters maximum.

Policy: SD-Subproducts and SD-Filesets within the same SD-Product cannot have the same tag; the fileset.tag cannot be identical to the “SD-Product tag”. SD-Fileset tags must be unique, even across SD-Products. The Fileset tag should be in capital letters.
Creating a Software Depot
Software Depot Package Components

Chapter 19 509

Title Status: Required
Description: The one-line, more detailed name for the SD-Fileset (appears in the Software Selection window) — 80 characters maximum. Use this chance to expand upon the fourteen character tag of the SD-Fileset.
Policy: This attribute will be seen rather often, so it should be carefully chosen to clearly and concisely tell the user what the SD-Fileset is. Often, the first position includes an abbreviation of the SD-Product name as well.

Description Status: Required
Description: A file or text string (limited to 8Kb) describing the SD-Fileset (displayed only when the user asks to see a description of the software and then selects this description for view). The free-form ASCII style of this attribute allows for detailed description of the software.
Policy: For maximum readability, each line should be limited to seventy-two characters or less; tabs should not be used in this file/text string.

Revision Status: Required
Description: The revision/version number of the SD-Fileset.
Policy: Each SD-Fileset must have the same revision as the SD-Product containing it.
The attributes (architecture, machine_type, os_name, os_release) may be specified not only at SD-product level, but also at SD-Fileset level. SD-Filesets which do provide values for the attributes will use their own values, not those of their parent SD-product. This allows a single SD-product to contain, for example an SD-Fileset compatible with both 32-bit and 64-bit machines, another SD-Fileset specific to 32-bit systems and other with 64-bit systems. The syntax and semantics of SD-Fileset attributes are identical to those of SD-product level.

architecture Status: Required
Description: The target systems on which this SD-Fileset is to be installed and will run. Summarizes the supported hardware and operating systems.
Policy: There are three parts to the architecture attribute: os name, os release, and os bits. The attribute is structured like:

<os name>_<os release>_<os bits>
The os name portion of the architecture attribute is always HP-UX.
The os release portion should be the minimum OS revision necessary to run the software.
If the SD-Product can run only on 11.11 and later, the os_release portion must be “B.11.11”.
The os bits portion specifies the OS bit configuration on which the SD-Fileset will run on:

• 32 — runs only on an OS in 32-bit mode.
• 64 — runs only on an OS in 64-bit mode.
For example:

• HP-UX_B.11.11_32 — earliest OS is 11.11, 32-bit only
• HP-UX_B.11.11_64 — earliest OS is 11.11, 32-bit or 64-bit
• HP-UX_B.11.11_32/64 — earliest OS is 11.11, 32-bit or 64-bit
Creating a Software Depot

Software Depot Package Components

- HP-UX_B.11.11_32/64 — lowest common denominator for multi release compatible software on 11.x (32 or 64-bit OS)

**os_name**

Status: Required

Description: OS releases on which the SD-Fileset will run:

\$(uname -s):\$(getconf CS_KERNEL_BITS)

The “getconf CS_KERNEL_BITS” indicated the bit configuration of the running kernel. Possible results are “32” (if running kernel is 32-bit) and “64” (if running kernel is 64-bit and, therefore supports 32-bit also)

Policy:

- HP-UX — SD-Fileset runs on both 32-bit and 64-bit. Most 32-bit SD-Filesets would use this.
- HP-UX:32* — SD-Fileset runs on 32-bit OS only
- HP-UX:*64 — SD-Fileset runs on 64-bit OS only

**os_release**

Status: Required

Description: OS releases on which the SD-Fileset is to be installed: \$(uname -r).

Policy:

- ?.11.* — Any 11.0 release
- ?.11.[01][01] — 11.00 through 11.11
- ?.11.00 — 11.00 only

**os_version**

Status: Optional

Description: OS versions on which the SD-Fileset will run: \$(uname -v). This attribute is not used. If it is set, it must be set to “*”.

Policy: * — Allow all versions.

---

**NOTE** Since the fileset is part of a product or a bundle, the os_version attribute has the same value as that of the product or bundle.

**category_tag**

Status: Optional

Description: The short name for the category of the software. This attribute associates the SD-Fileset with the standard category object.

Policy: This should match the proper category.tag attribute.

**is_kernel**

Status: Optional

Description: Defined if the SD-Fileset contains OS kernel files; that is, it contains files which are used to build the kernel. If it is set to TRUE additional policy should be handled in the control scripts which will be explained later in this document.

Policy:

- FALSE — The SD-Fileset does not require a system reboot after installation.
- TRUE — The SD-Fileset requires a system reboot after installation.
is_reboot  Status: Optional
Description: Defined if the SD-Fileset requires a system reboot after installation.
Policy:
- FALSE — The SD-Fileset does not require a system reboot after installation.
- TRUE — The SD-Fileset requires a system reboot after installation.

Prerequisites, corequisites and exrequisites
Status: Optional
Description: Defined the SD-Fileset(s) upon which this SD-Fileset depends. Prerequisites define SD-Fileset(s) which must be installed (configured) before this SD-Fileset can be installed (configured). Note that all SD-Filesets are installed and then configured. Prerequisites specify an install-time dependency. If prerequisites are installed within the same session as the dependent software, the prerequisites will be installed before its dependent. Corequisites define SD-Fileset(s) must be configured before the dependent SD-Fileset can operate correctly (i.e. a run-time dependency). Exrequisites define SD-Fileset(s) which are required not to be installed when this SD-Fileset is installed.
Policy:
When dependencies exist, state them. Prerequisites and corequisites are declared in the same manner and must follow the same rules. A dependent is the SD-Fileset that is declaring the dependency. A requisite is the SD-Fileset that is depended upon. The requisite(s) can be contained in the dependent's product or in another product. Each requisite SD-Fileset must be sufficiently specified. A requisite is normally specified as product.fileset.

The revision specified is that of the requisite's SD-product, not that of SD-Fileset itself, you should use >= operator when specifying revision. For example, r>=B.11.11 will allow your dependency to be satisfied by the highest revision greater than or equal to the 11.11 pre-release software.

preinstall  Status: Optional
Description: Pathname for the install pre-load SD-Fileset control script. The script is run by swinstall during the execution phase before loading the software files.

unpreinstall  Status: Optional
Description: Pathname for the undoing install pre-load SD-Fileset control script. The script is run during the swinstall load phase if is initiated. An unpreinstall script should undo any operation taken by the preinstall script.
Policy: Unpreinstall scripts need to be carefully crafted and extensively tested.

postinstall  Status: Optional
Description: Pathname for the install post-load SD-Fileset control script. The script is run by swinstall during the Load phase after loading the software files.

unpostinstall  Status: Optional
Description: Pathname for undoing the install post-load SD-Fileset control script. The script is run during the swinstall load phase if recovery is initiated. An unpreinstall script should undo any operation taken by the postinstall script.
Policy: Unpostinstall scripts need to be carefully crafted and extensively tested.
Creating a Software Depot

Software Depot Package Components

configure  Status: Optional

Description: Pathname for the configure for use SD-Fileset control script. The script is run by `swinstall` or `swconfig` to configure the host for the software, or to configure the software for host-specific information. Configure scripts are run by `swinstall` for all SD-Filesets after all SD-Filesets have completed the Load phase. The `swconfig` command can also be used to rerun configure scripts that failed during a normal install.

verify  Status: Optional

Description: Pathname for the verify integrity SD-Fileset control script. The purpose of the verify script is to check for the correctness of the product or fileset installation and configuration. It is run by the `swverify` command.

unconfigure  Status: Optional

Description: Pathname for the configure undo SD-Fileset control script. The script is run by `swconfig` or `swremove` to undo host or software configuration originally performed by a configure control script.

preremove  Status: Optional

Description: Pathname for the remove preremove SD-Fileset control script. The script is run by `swremove` during the Remove phase just before removing files.

postremove  Status: Optional

Description: Pathname for the remove postremove SD-Fileset control script. The script is run by `swremove` during the Remove phase just after the files have been removed.

request  Status: Optional

Description: Pathname for the request interactive SD-Fileset control script. The script is run by the `swask` command or the `swinstall` or `swconfig` commands with the “ask” option. The script requests response from the user as part of software installation or configuration.

Policy: Since request scripts are user-interactive, they cannot be used in Cold Install or unattended installation situations. Because of this, their use is not recommended.

File Definitions

The files contained in each SD-Fileset must be specified within the PSF. A file can be in one-and-only-one SD-Fileset. A file cannot reside in multiple SD-Filesets.

---

**NOTE**

A symlink (4) command can cause confusion during installation. Suppose a link `/usr/bin/x` which is a symlink to `/usr/bin/y`. If the product wants to install `/usr/bin/x` as a file or directory, SD will install the file or directory at `/usr/bin/y` and leave `/usr/bin/x` as a symlink to `/usr/bin/y`. SD will follow the links down and will not replace them.

file permissions  Status: Optional

Description: Explicitly specifies default permissions for the files being packaged into the SD-Fileset. This keyword applies only to the SD-Fileset in which it is defined. Later definitions within a SD-Fileset replace previous definitions. In the default condition, destination files receive permissions from their respective source files.

Use:

```
file_permissions [-m mode (octal) | -u umask] [-o [owner[,][uid]]] [-g [group[,][gid]]]
```
Creating a Software Depot

Software Depot Package Components

Chapter 19 513

**directory**  
Status: Optional  
Description: Specifies a source directory in which subsequently listed filenames are located (for this SD-Fileset only). The source directory can be either an absolute or relative pathname, relative pathnames will be interpreted relative to the current working directory in which `swpackage` is invoked.  
Use:  
directory source_dir [destination_dir]

**file**  
Status: Required  
Description: Specifies the files to be packaged into a SD-Fileset. When the directory keyword is used, pathnames are relative to the directory specified; otherwise, pathnames must be absolute. The directory keyword must be used for recursive file specification.  
This keyword is also used with directories when explicit permissions are given. Permissions can be included on this line or previously with the `file_permissions` keyword. No explicit file entry is needed for a directory when the `file_permissions` and directory keywords are implemented in sequence.  
Use:  
For explicit naming:  
file [-m mode (octal)] [-o [owner[],][uid]] [-g [group[],][gid]] source [destination]

For implicit (recursive) naming:  
file *

For explicit naming of directories:  
file [-m mode (octal)] [-o [owner[],][uid]] [-g [group[],][gid]] source_dir [destination_dir]

Combining the file definitions:  
File definition keywords can be combined for many purposes. Some possibilities are shown here. To include all files from `/src/develop/ddk/driver/qlispdrv` directory on the build system in the `/opt/ddk/sampldrvs/qlisp/` on the target system:  
directory/src/develop/ddk/driver/qlispdrv=/opt/ddk/sampldrvs/qlisp/file  
file *

To implement the previous example giving the directory `qlisp` and all of the files in `*` the permission set 755, owner root, and group users:  
file_permissions -m 755 -o root -g users  
directory/src/develop/ddk/driver/qlispdrv=/opt/ddk/sampldrvs/qlisp/file  
file *

To explicitly provide the permission set 555, owner root, and group root to the SD-Product directory `/opt/ddk/`:  
file_permissions -m 555 -o root -g root file /src/develop/ddk /opt/ddk
Control Scripts

These are the shell scripts which act on the product files. Their functions are self explanatory but can be chosen by the developer:

- preinstall
- postinstall
- preremove
- postremove
- configure
- request
- verify
- checkinstall, and so forth.

A product might not have all of the control scripts. The needed scripts must be specified in the PSF. Each Fileset in a product can have control scripts of its own.

Control Scripts Overview

Control scripts can be used to manage the SD-Product more efficiently. The use of control scripts is optional. This section describes in detail all the control scripts one needs to use during SD package. There are around nine named control scripts that have specific function within the SD environment. Each script should fulfill the requirements of the SD-UX software distribution tools.commands and of the particular fileset.

The control scripts are separate shell scripts which are included as attributes in the PSF for specific purpose and are to be used with SD commands/tools.

The following control scripts are available to be used in SD-UX:

- checkinstall: Run from within a swinstall session.
- preinstall: Run from within a swinstall session.
- postinstall: Run from within a swinstall session.
- configure: Run from within a swinstall session and also from swconfig.
- verify: For use with swverify.
- checkremove: Run from within a swremove session and also from swconfig.
- unconfigure: Run from within a swremove session and also from swconfig; the reverse of configure.
- preremove: Run from within a swremove session.
- postremove: Run from within a swremove session.
- request: Run from swinstall and swremove session; user interaction script.
Control Script Levels

The control scripts can be used at two levels of SD-UX:

- Product level
- Fileset level

Product Level Control Scripts

Control scripts for use with SD-UX are associated both with filesets and with products. A product can have its own set of control scripts; each fileset within that product can also have its own control scripts. The product level control scripts will be run whenever any fileset within that product is selected for installation, removal, or verification. Therefore the activities in product level scripts must pertain to all software delivered in that product, but not to any fileset in particular.

Any actions which might be included in a particular control script for every fileset in a product should instead be in the appropriate product level control script.

Fileset Level Control Scripts

The scripts for a particular fileset must pertain only to the installation, verification, configuration, or removal of that fileset, and not to any other fileset or to the product that the fileset is a part of.

Special Types of Filesets

There are several types of filesets based on the special actions that need to be taken on the system.

Reboot Fileset

A reboot fileset is one which requires that the target system be rebooted as part of software installation and configuration. The fileset. is_reboot attribute in the PSF containing the fileset must have a value of TRUE for a reboot fileset. If one or more selected filesets has a reboot flag, the system will be rebooted after all selected filesets have been installed or removed.

Kernel Fileset

A kernel fileset contains components for inclusion in the system kernel. Selection of one or more kernel filesets requires that the system kernel be rebuilt as part of the installation process. The fileset, is_kernel attribute in the PSF containing the fileset must have a value of TRUE for a kernel fileset. For HP-UX, all kernel filesets are also reboot filesets.

Prerequisite Fileset

A prerequisite fileset is one whose correct installation is required for proper install-time operation. A fileset should specify a prerequisite fileset only if it needs the prerequisite fileset configured before the fileset configure script is run.

Corequisite Fileset

A corequisite fileset is one whose correct installation is required for proper run-time operation.
Control Script Format

The file should be a shell script (as opposed to a binary) and written to be interpreted by the Posix.2 shell /sbin/sh. Korn shell (formerly /bin/ksh) syntax is acceptable to the Posix.2 shell, however the script must be run in the Posix shell (that is, the first line of any script must be "#!/sbin/sh" - not "#!/bin/ksh" or "#!/usr/bin/ksh"). A script written in "csh" will not be supported.

The file should have a simple header similar to the following example. The first line must indicate the interpreter shell by using the "#!" convention. Included in the header should also be comment lines which state the Product and Fileset to which the script belongs, the name of the script, the revision string as required by the what(1) command, and a simple copyright statement.

Example:

```bash
#!/sbin/sh
#########
# Product:
# Fileset:
# configure
(#) $Revision: 11.11 $
#########
# (c) Copyright Hewlett-Packard Company, 2002
#

```

Scripts without a specific shell identifier (that is, #!/sbin/sh) will be run under /sbin/sh by default. This is not recommended, because it may result unambiguous coding and testing approaches.

Execution Environment

This section contains details both of the file system location where the control scripts are delivered, and the environment variables to be called in those control scripts.

Location of the File System

When installing from magnetic tape or from a depot across the network, the checkinstall, preinstall, postinstall for a particular Fileset will be downloaded to a temporary directory from which they will be invoked: /var/tmp/<CATALOG_DIR>/catalog/<PRODUCT>/<FILESET>/control_script.

The form of the <CATALOG_DIR> is: AAAa<pid>, where <pid> is the swinstall process ID number. These files are delivered to that location from the depot immediately after product selection has completed, at the beginning of the Analysis phase and before any system checks have begun. The temporary directory is removed automatically upon exiting swinstall. After successful fileset installation, these control scripts and all other control scripts will be located in the Installed Product Database (IPD). They will be delivered to that location from the depot as part of the installation of the fileset’s other files:

```
/var/adm/sw/products/<PRODUCT>/<FILESET>/control_script
```

The location of the IPD is relative to the root directory under which the software installation is done. If the installation is to an alternate root, /mnt/disk2 for example, then the IPD for that software will be under:

```
/mnt/disk2/var/adm/sw/products/<PRODUCT>/<FILESET>
```

All necessary directories under /var/adm/sw will be created by the SD process. All files under those directories will be filled by SD initiated processes.
Environment Variables

The control scripts are invoked as the super user, that is, with an effective uid of 0. All of the control scripts are invoked without arguments. Information is learned by executing commands from within the scripts, by calling other utilities, and by environment variable values. The following environment variables will be passed to each control script. They can be tested for a particular value or used to construct another environment variable but should never be set or altered within a control script.

SW_PATH
A path to commands which can be called from within control scripts. The path for HP-UX 11.* is “/usr/lbin/sw/bin:/var/adm/sw/sbin:/sbin:/usr/bin:/usr/ccs/bin”.

PATH
The path to commands which can be called from within control scripts, this includes SW_PATH and is set when control_utils is sourced. The path for HP-UX 11.* is “${SW_PATH%:/}:/usr/sbin:/usr/lbin/sw”.

SW_ROOT_DIRECTORY
The path to the root directory under which the software is being installed. This will normally be “/”, but will be different in the case of an alternate root install. It will always be “/” for configure, swverify, and unconfigure. This value must be prefixed to the path of any installed file but not to commands to be executed. When prefixing this variable to a path, do not use a “/” as the leading character in the path, since that will expand to “//”. Most of the time this is harmless, but some uses of the resulting path involve string comparisons, and the “//” could cause a failure. The value is set on the command line as a suffix to the target system name, which is distinguished by the “@” symbol.

swinstall \* @ hpmysys:/mnt/disk30...

Be aware that the installation to an alternate root requires that the “-r” option be provided when the SD command is invoked, both in interactive and noninteractive mode. Also, changing product location (described below) has no effect on this variable.

SW_LOCATION
The directory where the product is located. This will normally be “/” but will be different in the case of a relocated product. When prefixing this variable to a path, do not use a “/” as the leading character in the path, since that will expand to “//”.

The value is derived from a path which can be appended to the product name of each locatable product on the command line or in the software selection file.

swinstall... Accounting:/opt/alternate1/acct...

In the GUI, a menu button labeled “Change Product Location” (accessible from the Actions pull-down menu) permits the user to redirect the installation of a product to a nondefault directory. This action is denied if the product is not locatable. If no path is specified or the product is not locatable, $SW_LOCATION is the default path defined on the media.

SW_SOFTWARE_SPEC
A string containing the full, unambiguous specification of the current software. The format is PRODUCT [.FILESET], l=$SW_LOCATION, r= revision, a= architecture, v= vendor.

SW_CONTROL_DIRECTORY
The directory path where the script currently being executed is located. See the “Location of the File System” section for locations of control scripts. This value is necessary if the control script invokes a subscript which is shipped along with the fileset.
SW_SESSION_IS_KERNEL

This variable is set to TRUE only if there are one or more kernel filesets which will be installed. It is unset at all other times. Its value should be tested by control scripts to learn whether a kernel rebuild and subsequent system reboot will be done.

SW_SESSION_IS_REBOOT

This variable is set to TRUE only if there are one or more reboot filesets which will be installed. It is unset at all other times. This variable will tell that reboot filesets have been selected, not that they have been installed or that the system has been rebooted because of them.

Relocating the Product

Many non-OS SD-Products can be installed to a directory other than the default directory specified by the developer. This action is known as relocating the SD-Product. In the GUI, the menu button labeled “Change Product Location” (accessible from the Actions pull-down menu) can be used to implement this redirection. In the command line mode, the product directory can be specified by appending that directory to the name of the product to be installed:

```
swinstall -s /opt/ddk/sampldrvs/qlisp/qlispdoc QLISPDOC:/opt/docs.
```

This will relocate the product QLISPDOC to new location /opt/docs. Here SW_LOCATION is now /opt/docs. It is possible to relocate products while installing to an alternate root:

```
swinstall -s /opt/ddk/sampldrvs/qlisp QLISP:/opt/ddk/sampldrvs/qlisp@newhost: /opt/ddk
```

Here both SW_LOCATION and SW_ROOT_DIRECTORY are changed /opt/ddk/ and “newhost”.

System Commands

Script developers should use only commands and syntax which comply to the IEEE Standard 1003.2 (Posix.2) standard. Compliance will not guarantee that all operating systems on which the scripts might run will conform to Posix.2.

Since control scripts are run under the POSIX shell, any built in command that works for the shell should work in a control script. A list of commands are shown in Table 19-1, “System Commands.”

<table>
<thead>
<tr>
<th>Table 19-1</th>
<th>System Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>export</td>
</tr>
<tr>
<td>bg</td>
<td>fc</td>
</tr>
<tr>
<td>case</td>
<td>fg</td>
</tr>
<tr>
<td>cd</td>
<td>for</td>
</tr>
<tr>
<td>command</td>
<td>getopt</td>
</tr>
<tr>
<td>continue</td>
<td>hash</td>
</tr>
<tr>
<td>echo</td>
<td>if</td>
</tr>
<tr>
<td>exec</td>
<td>jobs</td>
</tr>
<tr>
<td>exit</td>
<td>kill</td>
</tr>
</tbody>
</table>
There is a very brief list of nonbuilt-in commands which can be used safely by `checkinstall`, `preinstall`, and `postinstall` scripts. The commands in this list and the syntax used in control scripts must be available on all supported architectures and operating systems, and they should comply with the Posix.2 standard. This document will refer to the set of non-built-in commands as Posix.2 commands, refer to Table 19-2, “Posix.2 Commands for All Control Scripts,” even though some of them are HP specific and are not Posix compliant.

**Table 19-2 Posix.2 Commands for All Control Scripts**

<table>
<thead>
<tr>
<th>awk</th>
<th>cmp</th>
<th>In</th>
<th>pa</th>
<th>tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch-rc</td>
<td>cp</td>
<td>ls</td>
<td>rep</td>
<td>uname</td>
</tr>
<tr>
<td>chgrp</td>
<td>ioscan</td>
<td>mkboot</td>
<td>rm</td>
<td>wc</td>
</tr>
<tr>
<td>chmod</td>
<td>lifcp</td>
<td>mkdir</td>
<td>rmdir</td>
<td></td>
</tr>
<tr>
<td>chown</td>
<td>lifls</td>
<td>mv</td>
<td>sed</td>
<td></td>
</tr>
</tbody>
</table>

When an OS update is done, a situation which could place incompatible versions of essential commands on the system, the user must first fetch the correct version of SD-UX. Part of that process includes getting usable versions of these commands to a location on the system where control scripts will access them.

These commands will be preserved for the duration of the installation process for use by the `checkinstall`, `preinstall`, and `postinstall` scripts. Use of `$SW_PATH` in all control scripts to set PATH, as shown in the sample control scripts, ensures that the preserved version of the command is accessed rather than the newly installed, possibly incompatible versions.

In addition to the commands previously listed, the commands shown in Table 19-3, “Additional Preserved Commands,” will be preserved during an OS update. These commands are listed separately because they are not appropriate for use in control scripts.

**Table 19-3 Additional Preserved Commands**

<table>
<thead>
<tr>
<th>ar</th>
<th>ccom</th>
<th>cpp</th>
<th>make</th>
<th>reboot</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc</td>
<td>config</td>
<td>ld</td>
<td>mount</td>
<td>umount</td>
</tr>
</tbody>
</table>

**SW DIST Commands**

In addition to the SD executables there are tools that are used by SD-initiated processes and are delivered in the SW-DIST product. These include:

- `/usr/lbin/sw/control_utils`  General purpose utility functions for all control scripts
- `/usr/sbin/update-ux` Utility used to update to a newer OS.

The installation of system software, especially during development cycles, must be preceded by importing the most recent version of the SW-DIST product by `swinstall`-ing it from the OS-Core depot. `update-ux` installs the latest SW-DIST automatically.
General Control Script Actions

This section covers the details of the control script actions such as inputs from the user, what the control script output is, and any exit values returned from the control scripts.

Input

A control script must never require interaction from a user. All control scripts must be designed to run to completion, whether successful or not, without intervention (an exception being “request” script). There are commands for which no input is required under all normal circumstances, yet in some unique configuration require user confirmation. These situations must be anticipated by the developer.

Output

Standard output and standard error from control scripts are directed to the swagent log file:

```
$SW_ROOT_DIRECTORY/var/adm/sw/swagent.log
```

The log file is located relative to the root directory under which the software is installed. The output consists of messages initiated by the swagent process and messages resulting from the control script actions. If there are any intermediate events of which the user should be informed, the resulting messages are appended to the log file. Output from the control script, including output from commands the control script invokes, is appended to the log file. Generating those messages is the responsibility of each control script. Notice there are no further messages from swagent if the script succeeds, that is, if it exits with a “zero” value. The script must not write directly to the console or attempt any other method of writing to the display. Commands called by control scripts must not write to the console. Instead, the output from the script will be handled by the calling process and appended to the log file. Output should be simple echo statements to stdout; they will be captured by the parent process and appended to the log file.

For example, the output to the log file if a warning is returned from the checkinstall script might look like:

```
* Running the 'checkinstall' script for 'FUEL.UDMH'

WARNING: The product 'OXIDIZER.N2O4' is already present on the system. Proceeding with the installation of FUEL.UDMH can result in a volatile configuration.
If used with the Instant Ignition product, results could include locally elevated levels of temperature and noise, reduced visibility, loss of volatile memory, burned monitor phosphor, and severe disk fragmentation. Your mileage may vary.

WARNING: The 'checkinstall' script for 'FUEL.UDMH' had a warning. (exit code "2").
The script location was: '/var/tmp/AAAa01234/catalog/FUEL/UDMH/checkinstall'

* This script had warnings but the execution of this product will still proceed.
Check the above output from the script for further details.
```

At the end of the checkinstall execution in the Analysis Phase, the log will have a message:

```
WARNING: The Analysis Phase had warnings. See the above output for details.
```

Exit Values

The calling process acts upon the value returned by each control script. The action triggered by the return value differs depending on the type of script being run. The prescribed return values are:

0 — SUCCESS  No impediment to installation or removal was found. The process can proceed to the next task in the sequence.

1 — FAILURE  The action differs greatly according to script type. Refer to the individual script discussions for details.
2 — WARNING
A condition exists or an event was detected which does not impede the installation or removal process but should be presented to the user. This message type is also to be used if the possibility of a failure exists, yet there is not enough information to guarantee a failure.

3 — EXCLUDE
The action differs according to script type. This exit value is only valid in the checkinstall, and checkremove. Refer to these individual script discussions for details.

In the checkinstall and checkremove script scenarios, the EXCLUDE exit value has far reaching ramifications when used with filesets which are dependencies for other filesets. It is highly recommended that this return value be implemented only for isolated filesets, no other fileset should depend (prerequisite or corequisite) on a fileset which might be excluded.

11 — GLOBAL_ERROR
A serious condition exists which demands that the install process not continue past the Analysis Phase. This will not cause an immediate exit of the process, but will prevent any selected software from being installed. The remaining check scripts will be executed and they may emit messages to the log file. The system will be checked for available disk space. The session will exit after cleaning up.

Required Actions
The SD requirements do not insist that any one control script be present for correct operation. An absent control script file is functionally equivalent to an existing script which returns SUCCESS. If a fileset's requirements do not include any operations in a particular control script, then that script must be omitted from the fileset's delivery. It is the packager's responsibility to verify that a control script is needed or not. A control script must be re-executable. The developer must include safeguards against a failure for an invocation of a script that has already been executed. A script which makes a change to the system the first time it is executed must have protection against error for subsequent executions.

Permitted Actions
The set of actions that a particular type of script is allowed to take is discussed in the details for each script type in the "Guidelines for Control Scripts" section.

Prohibited Actions
There are some actions that must not be initiated from any control script:

- Do not shut down or reboot the system.
- Do not change the system's INIT state.
- Do not initiate a kernel build.
- Do not remove any control scripts after execution.
- Do not leave debugging enabled. The -u and -x shell options must be unset.
- Do not remove any of the fileset's files in the Installed Product Database.
- Do not alter files owned by a different fileset.

Do not modify startup/shutdown scripts either directly or indirectly. Each fileset that has a startup/shutdown script should have a configuration file under /etc/rc.config.d which contains system configuration parameters unique to that fileset. The startup/shutdown script should be completely data free and should
require no modifications. Modifications to the configuration data file should be made with extreme caution. Changing system configuration parameters that may have been established deliberately by the user is not something to be done casually or silently.

Do not change any links between startup/shutdown scripts. The links should exist on the media, and should remain as shipped after configuration.

When an application installed under /opt requires a startup script, moving files and creating links are valid components of the install process. This scenario presents an exception to the above prohibition.

**control_utils**

The function library `/usr/lbin/sw/control_utils` is delivered with the SW-DIST product. It is a library of Posix.2 shell compatible functions which perform commonly used tasks. The `control_utils` file should be sourced by each control script. If the control script does not require anything from the `control_utils` file, omit this. The establishment of this function library allows a standardized set of functions to be available to all filesets.

**Installed State**

The result of a software installation attempt will leave the product or fileset in a particular state on the system. The state of the installed software can be displayed by executing the following command:

```
swlist -l fileset -a state <PRODUCT>.<FILESET>
```

The output will look like:

```
# Target: hpmysys:/
#<PRODUCT>.<FILESET> <installed_state>
```

The function `get_install_state` in `control_utils` can also be used to retrieve a fileset's installed state.

Some of the several possible installed states:

- Configured — the software has been installed, and the configure script has been run successfully
- Installed — the software has been installed, and:
  - the configured script has not yet run
  - the configured script has run and failed
  - the unconfigure script has been run
- Transient — the software in the fileset's product is in the process of being installed or removed. This state is established for each of the product's filesets that is marked for installation. The state of all marked filesets in a product is changed to transient when the first fileset in that product enters Load (or Remove) phase. The state changes to installed when the last of the product's marked filesets has been successfully installed.
- Corrupt — the software currently on the system is an incomplete state. This might be the result of a software installation that had an abnormal termination while the fileset was in transient state.
Recommendations

There are number of conditions that a developer should not assume when developing a control script. Many of them are mentioned throughout this document, but they bear repeating:

- The directory under which software is being installed might not be “/”. The script should prefix \${ROOT} from control_utils to ensure that paths are accurately defined.
- One of more volumes on the system might not be configured for long file names. Limit all file names to 14 or fewer characters including all extensions.
- The system might not be quiescent. A binary might be busy (e.g., open for execution) while its fileset is being updated. In this event, the file's basename will be renamed to #basename and the incoming file will be placed on the system.
- The current working directory ($CWD) must not be assumed or considered in a control script.
- Non-standard features in the super-user’s home environment will not be available.
- A command might not execute correctly. Checking the return value for success is prudent.
- A command’s functionality might have changed from a previous release. When in doubt, refer to the manpages for the new OS release.
- Other filesets might (or might not) be already present on the system. Test for the installed state of a fileset before acting on it.
- One or more filesets might not have been installed correctly. Be sure to explicitly specify all needed requisite filesets.
- When using \$(ROOT), $SW_ROOT_DIRECTORY, $SW_LOCATION, or $SW_CONTROL_DIRECTORY as a prefix to a path, do not also use “/” as the first character in the rest of the path. Each of these variables ends in a “/”, which would produce a “//” in the expanded path. That causes problems if the path is used in string comparisons.
- Use $SW_SYSTEM_FILE_PATH instead of defining the path for the location of the kernel.
- Use the ch_rc command to modify parameters within /etc/rc/config.
- Use utilities provided in control_utils whenever possible. For example, use the newconfig_msgs function instead of writing your own and placing it in your control script.

Performance Considerations

- Use Posix.2 shell built-ins wherever possible. The Posix.2 shell has a rich set of capabilities which can be used in place of many common commands. Creative use of built-ins, Input/Output manipulations, pattern matching and conditional expressions make scripts more code efficient and quicker.
- Source control_utils only when using the functions it provides.
- The /etc, /dev and /var directories are guaranteed to exist for cold install, diskless install and all forms of update. You should never create these directories.

Backward Compatibility Considerations

Several new functions are added to the control_utils file for the 11.11 release. These functions will not be available on systems earlier than 11.11 that haven't been updated with the 11.11 version of SD. If the developer wants to use a function from the 11.11 control_utils; the easiest thing to do is to copy the functions wanted to a file and then deliver those functions as a control_file.
As an example, the `msg` function is being used for the 11.11 version of `control_utils`, the developer could copy the function to your packaging directory. Then the file containing the function (let's call it “foo” to keep with UNIX tradition) would be included as a `control_file` within the product or fileset that has control scripts needing this new functionality.

To include “foo” you would add this line to your PSF:

```
control_file foo
```

Then with your control scripts needing the new functionality you would source your `control_file` like this:

```
.
$SW_CONTROL_DIR/foo

UTILS= "/usr/lbin/sw/control_utils"

if $[[] ! -f $UTILS ]
then
  echo "ERROR: Cannot find $UTILS"
  exit 1

fi

.$UTILS

exitval=$SUCCESS
```

### Guidelines for Control Scripts

This section covers the guidelines for all the control scripts of SD. This involves a description of all aspects required to write and execute a control script in the SD environment. It describes in detail the purpose and order of script execution, and actions of all control scripts in SD.

#### Checkinstall

**Purpose of checkinstall script**

The purpose of a checkinstall script is to ensure that the target system has no product or fileset specific conditions which would cause either an installation failure or a runtime failure. This specifically excludes conditions that are tested elsewhere in the `swinstall` process such as inadequate disk space, unmounted volumes, unresolved fileset dependencies, inappropriate architecture, and so on.

**Order of script execution**

The checkinstall scripts are called while the installation is in the Analysis phase. Execution occurs after the check for mounted volumes and the check of currently installed software has completed, and before the check for product and fileset dependencies and the analysis of available disk space. The checkinstall script is run only when the user has invoked `swinstall`, and not during `swcopy` or `swremove`. The checkinstall script for a particular product or fileset is invoked in series with the checkinstall scripts of all other selected products and filesets. The checkinstall scripts for selected products and filesets are all executed before any installation is begun. If there is a product level checkinstall script, it is called prior to the calling of any checkinstall scripts belonging to that product’s filesets. Unlike the preinstall and postinstall scripts, checkinstall works with all filesets in one product before moving on to the next product.

The explanation of the phases mentioned can be found in “Step 5: Packaging the Components” and the “Installing a Depot” section.

**Checkinstall system commands**

The commands available to the checkinstall script under all conditions are relatively few. They include the Posix.2 commands listed in the “System Commands” section.
Commands that are part of the SW-DIST product. For example, the `get_sysfile` command can be used to extract the `/stand/system` file to check for a certain kernel configuration. The path to the command might not be established in the environment's `$PATH`. In this case the command must be specified with its full path.

Checkinstall actions

Output:

As mentioned in the “Output” section.

Exit values:

The calling process acts upon the value returned by each checkinstall script. These values correspond to environment variables set in the control_utils code. The prescribed return values are:

0 — SUCCESS

No impediment to installation found. Installation will proceed immediately if in non-interactive mode. If in the interactive mode, the process will wait for the user to select the start_install action item.

1 — FAILURE

In a product level checkinstall script, the entire product will be unselected, along with all other products which had any of this product's filesets as a requisite. In a fileset level checkinstall script, the fileset will be unselected along with all other filesets which had any of this fileset as a requisite. The error message should explain why. Any remaining filesets and products unaffected by the failure will remain selected for installation. Any remaining checkinstall scripts will be run in install order.

2 — WARNING

Possible conflict explained in warning message. The selection of the current fileset and its prerequisite and corequisite filesets will remain intact, barring other failures. The warning message should explain the condition.

3 — EXCLUDE

The current fileset will be unselected. A message to the log file should explain the unselection. Any remaining filesets unaffected by the unselection will still be selected for installation. Any remaining checkinstall scripts will be run in install order.

11 — GLOBAL_ERROR

This value will cause the install session to exit before any software installation occurs. The Analysis Phase will proceed through all the checks in order to accumulate all analysis information. This exists to allow a checkinstall script prevent installation if test results warrant such action.

Required Actions:

The actions of a checkinstall script must be extremely unobtrusive since there is no commitment to installing at the time the scripts are run. The checkinstall script, like all control scripts, must be re-executable. It is possible for a checkinstall script to be executed numerous times within a single install session.

Permitted Actions:

Typical actions valid from within a checkinstall script at any time are:

- Test for selected software.
- Test the system's hardware configuration.
- Test kernel configuration.
- Test the init state of the system.
- Test the system's I/O structure.
Creating a Software Depot
Software Depot Package Components

- Test for software already installed.
- Test the revision, or other attributes of installed software.

Prohibited Actions:
Mentioned in the “Required Actions” section. In addition the checkinstall script must not:
- Create new files, other than temporaries that must be removed before exiting.
- Copy, move, remove, or modify existing files.
- Kill processes.
- Spawn processes that will linger after the script exits.

Configuration Directory and Functions

The newconfig directories and functions are used in configure and preinstall scripts. The handling of files delivered to the newconfig directory is done by newconfig functions. These functions are required for the configure, pre- and post-install scripts. The newconfig directory for the OS products is /usr/newconfig.

For example, if /etc/disktab is fileset that is to be changed with each major release, then the “working” file is the file at the ultimate path. This file may or may not have been modified by the system administrator after being placed there by a previous installation.

Example: /etc/disktab

The “previous” file is the file delivered by the last software installation of the current fileset. This file should be unchanged from the way it was delivered.

Example: /usr/old/usr/newconfig/etc/disktab

The third file of this discussion is the one being installed by the current invocation of swinstall:

The “new” file is the file that is newly installed by swinstall.

Example: /usr/newconfig/etc/disktab

Preinstall

Purpose of Preinstall control script

The purpose of a fileset’s preinstall script is to prepare the system for installation of the fileset. By the time the preinstall script for a particular fileset is executed, the actions of that fileset’s checkinstall script have determined that there are no fileset-specific impediments to installing the fileset. Then the Analysis Phase has ensured that there are no known system-specific impediments to installing the current fileset. All that remains is to prepare the system and to begin installing files. The steps to prepare the system are done by the fileset’s preinstall script. A typical preinstall script is intended to kill processes that could interfere with installation, for example daemons that keep an executable file opened. Removal of obsolete software is done in a preinstall script.

Order of script execution

The preinstall scripts are called while the installation is in the Execution Phase. Each fileset’s preinstall script is executed just before that fileset’s files are installed onto the target system. A product level preinstall script is called prior to the calling of any preinstall scripts belonging to that product’s filesets.
Creating a Software Depot
Software Depot Package Components

Chapter 19

Preinstall system commands

The commands available to the preinstall script under all conditions are relatively few. They include the Posix.2 commands that are part of the SW-DIST product listed in the “System Commands” section.

Preinstall actions

Output:

AS mentioned in the “Output” section.

Exit Values:

The calling process acts upon the value returned by each preinstall script. The prescribed return values are:

0 — SUCCESS All processes were run successfully.

1 — FAILURE A FAILURE occurred, indicating that the installation and configuration is certain to fail in some way. The failure message should explain what the failure was. If a kernel fileset’s preinstall script return FAILURE, the install process will exit if in noninteractive mode and will suspend and wait for user input if in interactive mode. When the fileset is neither a kernel fileset nor a prerequisite for a kernel fileset, installation of the files will proceed even if a preinstall FAILURE occurs.

2 — WARNING A WARNING was sent to the log file. A condition was detected which might result in installation or configuration failure. The warning message should explain the condition. The installation of selected software will proceed.

Permitted Actions: Typical actions valid from within a preinstall script are:

Call the newconfig_prep function to set aside files that will be delivered under /usr/newconfig as follows:

newconfig_prep:

The fileset’s preinstall script should handle all files delivered under /usr/newconfig by that fileset. It should invoke the control_utils function newconfig_prep once per newconfig file, in a loop if more than one file is involved. The function requires one argument — the absolute path of the file’s ultimate location, not including a $SW_ROOT_DIRECTORY or $SW_LOCATION prefix.

Example: newconfig_prep /etc/disktab

The function will copy the previous file from its place under /usr/newconfig to a corresponding path under /usr/old/usr/newconfig (see the “Configuration Directory and Functions” section). The file under /usr/newconfig will remain. Results are unspecified if the working file is linked to the previous file, or if either is linked to any other file. The newconfig_prep function must be called for every file that is handled by newconfig_cp.

newconfig_cp:

The newconfig_cp function is usually called from a fileset’s configure script. It must be called for every file that is handled by newconfig_prep. Before the fileset’s postinstall or configure script is executed, the previously installed file will have been copied (by the newconfig_prep function) from under /usr/newconfig to /usr/old/usr/newconfig, the new file will have been installed under /usr/newconfig, and the working (that is, currently installed) file will be unaffected. The task of the newconfig_cp function is to decide whether the newly installed file should overwrite the working file, and to do so if necessary. The
newconfig_cp function should be invoked once per newconfig file. Like the newconfig_prep function, the newconfig_cp function requires one argument: the absolute path of the file's ultimate location, not including a $SW_ROOT_DIRECTORY or $SW_LOCATION prefix.

Example: newconfig_cp /etc/disktab

The function first checks for existence of the new file, and exits with an error value if the new file is not found. A likely cause of this error is if the file is delivered directly under /usr/newconfig rather than as a pseudo-root, or if the argument is passed as a relative path rather than an absolute path.

Prohibited Actions: The list of prohibited actions given in the “Required Actions” section, also applies here.

Postinstall

Purpose of Postinstall Script

Postinstall scripts are used to prepare for a kernel build when required by the install conditions. They also can drive events that must occur before a system reboot occurs.

Order of Postinstall script execution

Each fileset's postinstall script is run just after the fileset's files are loaded onto the target system during the Execution Phase of the installation process. A product level postinstall script is invoked after all of that product's filesets have been installed.

Postinstall System Commands

The commands available to the preinstall script under all conditions are relatively few. They include the Posix.2 commands listed in the “System Commands” section, and commands that are part of the SW-DIST product.

Postinstall Actions

Output:

As mentioned in the “Output” section.

Exit Values:

The calling process acts upon the value returned by each postinstall script. The prescribed return values are:

0 — SUCCESS

The script completed successfully.

1 — FAILURE

The fileset installation experienced a fatal error. Although the files were installed onto the target system successfully, a required post-installation action failed. This could jeopardize the correct functionality of the fileset.

If a kernel fileset's postinstall script returns FAILURE, the install process will exit if in non-interactive mode and will suspend and wait for user input if in interactive mode.

When the fileset is neither a kernel fileset nor a prerequisite for a kernel fileset, installation of all other files will proceed even if a postinstall FAILURE occurs.

2 — WARNING

Installation succeeded, but a condition exists of which the user should be notified.

Required Actions:

A postinstall script is an essential part of every kernel fileset and must be delivered with each kernel fileset because kernel configuration is done from within a postinstall script.
Permitted Actions: Typical actions valid from within a postinstall script at any time are:

- Copy, move, or remove files.
- Conditionally create links to another location.
- Use `newconfig_cp` to conditionally copy files delivered to `/usr/newconfig` to a working location.
- Use `mod_systemfile` to modify `/stand/system`.
- Perform other actions that affect the successful build of a kernel.

Prohibited Actions: The list of prohibited actions listed in the “Required Actions” section are also applicable.

Configure

Purpose of Configure Script

A configure script is used to perform product or fileset installation actions that cannot be accomplished by simple unconditional file extraction from the software source media. Configure scripts are typically used to alter system specific files. The configure script is invoked either as part of the `swinstall` process, or else as the result of the user invoking the `swconfig` command without the `-u` option. The configure script is not run when the user has invoked `swcopy` or `swremove`.

Configure Execution Environment

Since the configure scripts are run on the target system, `$SW_ROOT_DIRECTORY` will be “/” in all cases when configure is run.

Order of Configure Script Execution

If one or more reboot (including kernel) filesets has been successfully installed, then the execution of the configure scripts is postponed until after the system has rebooted. If all of the selected filesets are non-reboot filesets, then the configure scripts are invoked from within the `swinstall` session as the last step of the Execution Phase. The configure scripts are executed in product order, not install order.

The order is only affected by prerequisites and kernel filesets. Within any one software product, the configure script is executed first for kernel filesets and their prerequisites and then for other non-kernel filesets. Configure scripts for a fileset’s prerequisites are run before the fileset’s own configure script to ensure all features of a prerequisite fileset are available during the configuration of the current fileset. If a software product has a configure script, that product’s configure script is called prior to the calling of any configure scripts belonging to that product’s filesets.

Configure System Commands

The configure script, in comparison to the preinstall and postinstall scripts, has a greater selection of commands available to it under all conditions. They include:

- The Posix.2 commands listed in the “System Commands” section.
- Commands that are part of the SW-DIST product.
- Commands that are included in any prerequisite filesets.
- Commands included in the script’s own fileset that do not need to be configured before using.
Configure Actions

Output:
As mentioned in the “Output” section.

Exit values:
The calling process acts upon the value returned by each configure script. The prescribed return values are:

0 — SUCCESS  Fileset configuration succeeded. The state of the fileset is changed to be CONFIGURED.
1 — FAILURE  The fileset configuration experienced a fatal error. Although the files were extracted onto the system successfully, a required configuration action failed, and one or more features of the fileset will not work. The state of the fileset remains INSTALLED.
2 — WARNING  Configuration succeeded, but a condition exists of which the user should be notified. The state of the fileset is CONFIGURED.
3 — EXCLUDE  Fileset configuration must be run once again to complete this configuration. The fileset state remains INSTALLED. The next invocation of the configure script (by swconfig at system reboot) will complete the configuration process.

Permitted Actions:  Typical actions valid from within a configure script at any time are:

- Create special device files.
- Append to existing files such as the fileset's rc.config.d file.
- Conditionally establish symbolic links.
- Change file attributes.
- Conditionally copy, move, or link files which have been delivered to a location under a private directory (that is, one only known to the packager).

Prohibited Actions:  The list of prohibited actions listed in the “Required Actions” section, also applies here.

Unconfigure

Purpose of Unconfigure Script  The purpose of the unconfigure script is to undo most configuration changes that were made to the system by the corresponding configure script. A typical action in an unconfigure script would be the removal of device special files or the changing of a system configuration parameter in a file under /etc/rc.config.d.

Order of Unconfigure Execution  The unconfigure script is invoked as part of the swremove process, or as the result of the user giving the -u option to the swconfig command. Execution of each fileset's unconfigure script occurs just after the user has elected to proceed with removing the selected products and filesets, and just before the selected products and filesets are actually removed. It is the first step in the commitment to remove functionality. If a software product has an unconfigure script, that product's unconfigure script is called after the calling of any unconfigure scripts belonging to that product's filesets. The unconfigure scripts for any swremove session are invoked in reverse prerequisite order. The unconfigure script for all
prerequisite filesets that have been selected for removal will be run after
the unconfigure script for the current fileset. The presence of kernel
and/or reboot filesets has no effect on the order of execution.

**Unconfigure System Commands**

The commands available to the preremove script under all conditions include:

- The Posix.2 commands listed in the “System Commands” section.
- The commands that are part of the SW-DIST product.
- Commands that are included in any prerequisite filesets.
- Commands that are included in the script’s own fileset.
- Core system command.

**Unconfigure Actions**

**Output:**

As mentioned in the “Output” section.

**Exit Values:**

The calling process acts upon the value returned by each unconfigure script. The prescribed return values are:

0 — SUCCESS

The script completed successfully. The install state is now INSTALLED.

1 — FAILURE

The script experienced a fatal error. If called as part of an **swremove**
process, the attempt to remove the files in the product or fileset will
proceed regardless. The install state is now INSTALLED.

2 — WARNING

A nonfatal condition was detected and should be reported to the user. If
called as part of a **swremove** process, the attempt to remove the files in the
product or fileset will proceed regardless. The install state is now
INSTALLED.

3 — EXCLUDE

The fileset was unselected. Unconfiguration did not take place. The install
state is unchanged.

**Permitted Actions:**

Typical actions valid from within a unconfigure script at any time are:

- Kill processes, including daemons, owned or spawned by files in the
current fileset.
- Move or remove files and directories that were created by the
corresponding configure script.
- Alter a value in the system’s configuration files.
- Remove client specific files such as log files.
- Use **IPD_addfile()** and **IPD_delfile()** (**control_utils**) when
copying, moving, or removing files.

**Prohibited Actions:**

The list of prohibited actions listed in the “Required Actions” section, also
applies here.
Preremove

Purpose of Preremove Script

The purpose of the preremove script is to perform any necessary actions not done in the unconfigure script in preparation for the removal of the fileset's files. It should undo any actions taken in a postinstall script. For kernel filesets, it should modify the /stand/system file to delete entries such as driver names and configurable parameters whose functionality is part of the fileset.

Order of Preremove Execution

The preremove scripts are called during the Execution Phase. Execution of each fileset's preremove script occurs just before that fileset's files are actually removed. If a software product has a preremove script, that product's preremove script is called prior to the calling of any of that product's fileset level preremove scripts. The preremove scripts for any swremove session are invoked in reverse prerequisite order. The preremove script for all prerequisite filesets that have been selected for removal will be run after the preremove script for the current fileset. The presence of kernel and/or reboot filesets has no effect on the order of execution.

Preremove System Commands

The commands available to the preremove script under all conditions include:

- The Posix.2 commands listed in the “System Commands” section.
- Commands that are part of the SW-DIST product. Commands that are included in any prerequisite fileset.
- Commands that are part of the script's own fileset, provided the unconfiguration has not rendered them unusable.
- Core system commands.

Preremove Actions

Output:

AS mentioned in the “Output” section.

Exit Values:

The calling process acts upon the value returned by each preremove script. The prescribed return values are:

0 — SUCCESS The script completed successfully.

1 — FAILURE The script experienced a fatal error. The attempt to remove the files in the product or fileset will proceed regardless.

2 — WARNING A nonfatal condition was detected and should be reported to the user. The attempt to remove the files in the product or fileset will proceed regardless.

Permitted Actions:

Typical actions valid from within a preremove script are:

- Move or remove files and directories.
- Modify the /stand/system file.

Prohibited Actions:

The list of prohibited actions listed in the “Required Actions” section, also applies here.
Postremove

Purpose of Postremove Script

The purpose of the postremove script is to perform any necessary cleanup actions after the fileset's files have been removed. Any files which might have been created by the fileset, and which might not have been added to the IPD should be removed by the postremove script. A typical action in a postremove script would be the removal of newly emptied directories when those directories are the exclusive property of the fileset.

Order of Postremove Script Execution

The postremove script for a fileset is called in the Execution Phase immediately after that fileset's files have been removed from the system. If a software product has a postremove script, that product's postremove script is called after the calling of any postremove scripts belonging to that product's filesets. The postremove scripts for any swremove session are invoked in reverse prerequisite order. The postremove script for all prerequisite filesets that have been selected for removal will be run after the postremove script for the current fileset. The presence of kernel and/or reboot filesets has no effect on the order of execution.

Postremove System Commands

The commands available to the preremove script under all conditions include:

- The Posix.2 commands listed in the “System Commands” section.
- Commands that are part of the SW-DIST product.
- Commands that are included in any prerequisite fileset.
- Core system commands which have not been affected by the current removal.

Postremove Actions

Output:

As mentioned in the “Output” section.

Exit Values:

The calling process acts upon the value returned by each postremove script. The prescribed return values are:

0 — SUCCESS  The script completed successfully.
1 — FAILURE   The script experienced a fatal error. The error message is logged and swremove continues.
2 — WARNING    A nonfatal condition was detected and should be reported to the user.

Permitted Actions:

Typical actions valid from within a postremove script at any time are:

- Remove newly emptied directories that are owned and used exclusively by the fileset.
- Replace any files that were set aside by the installation of the fileset.
- Any other actions that will bring the system closer to the condition it was in prior to the fileset’s installation.

Prohibited Actions:

The list of prohibited actions listed in the “Required Actions” section, also applies here.
Files to Package

The files needed to create a product are the Object/Source files or document files on which the PSF and control scripts act to install/uninstall/verify a product. These files can be drivers, libraries, tools or a document. The pathnames of these files, both in the package and their final destinations when a product gets installed on a host, have to be specified in the PSF.
Creating a Software Depot

There are various steps that are involved in creating an SD package:

- “Step 1: Design an SD Structure”
- “Step 2: Selecting the Product Directory Structure”
- “Step 3: Writing a PSF”
- “Step 4: Writing Control Scripts”
- “Step 5: Packaging the Components”
- “Step 6: Registering the Depot”

Step 1: Design an SD Structure

As explained in the “Software Depot Package Components” section, packaging can be done at SD-Product level or at Bundle level. It is up to the user to select what level best suits his/her package. If the user intends to have more than one product as part of a package then he/she can choose Bundle level packaging. If not SD-Bundle can be omitted and packaging can also be done at SD-Product level. Depending upon the usage (Product contains several filesets, subproducts can be used to group logically related filesets.) of the SD depot SD-Subproduct objects should be used. The user also has to decide whether the packaging calls out for any specific control scripts.

Step 2: Selecting the Product Directory Structure

Once the SD package structure is decided, product directory structure (location) has to be selected. This is the directory structure where all the source/object/bin/docs files reside once installed. This has to be decided before writing the PSF as directory information is required as one of the attributes in SD-Product and SD-Fileset category of the PSF.

Step 3: Writing a PSF

After the SD structure and product directory structures are decided, it is time to write a PSF. As mentioned earlier a PSF is a master file which has the configuration attributes of the package. These attributes are to be filled in, they take effect accordingly once the product is installed.

An overview and all the attribute policies are explained in detail under the “Product Specification File” section, and the “SD Objects Attributes Classification and Flow” section.

Based on the selection of structure refer to (step 1) a PSF has to be written. To omit the bundle and consider product level packaging the bundle attribute for such a package must not be included. When considering Bundle level packaging both bundle and product attributes need to be included. In both cases Fileset attributes are needed. The Subproducts' attributes consideration is optional.

The attributes mentioned in the “SD Objects Attributes Classification and Flow” section are as required (check the status) and should be filled in, as mentioned in the policies.

NOTE Policies are the recommendations suggested. If not followed accordingly the PSF will generate errors during package creation.
Writing a PSF Vendor Object

In the PSF, the vendor information of the package needs to be filled in first. The information filled in should be unique. Information such as name of the products' company.

Under this object fill in all attributes:

tag The vendor's trademark.
title Vendor's trademark expansion or company name.
description Description of the vendor.

Based on the vendor attribute policies discussed under the "Vendor Attributes" section, a sample template of vendor object of a PSF:

```
vendor
tag <.....>
title <.....>
description <.....>
end
```

Example:

```
#vendor information (considered an comment as in case "/** for "C " code)
vendor
tag HP
title Hewlett-Packard
description A Technology company
end #vendor
```

Writing a PSF Category Object

The category attributes are defined outside the SD-Bundle, SD-Product or SD-Fileset. The category_tag attribute within the SD-Bundle, SD-Product and/or SD-Fileset definition is used to refer to the associated category object. Category class attribute definitions are “global”, a particular category tag and description applies to all referring objects in the depot/on the media.

The list of attributes and the policies of filling in these attributes are mentioned under the “Category Attributes” section:

tag Identifier of the category object.
title Detailed name for the category.
description A text string describing the category.

A Category object template:

```
category
tag <.....>
title <.....>
description <.....>
end
```

Example:

```
#category information
Category
tag TrialUseApps
title Trial Use Software Applications
description "Trial Use Software Applications"
end #category
```
Writing a PSF Bundle Object

The package Bundle information follows the vendor information. All products and filesets that are considered under the bundle are mentioned in this object.

The list of attributes and the policies of filling in these attributes are also mentioned under the “SD-Bundle Attributes” section:

- **tag** Identifier of the bundle.
- **title** Detailed name of the bundle.
- **Description** A text string describing the bundle.
- **Revision** Revision number of the bundle.
- **Architecture** Target systems on which the bundle will be installed.
- **os_name** The OS’s on which the bundle will run.
- **os_release** OS releases on which the bundle will run.
- **os_version** (optional) OS versions on which the bundle will run.
- **vendor_tag** Short name of the vendor (refer to the “Writing a PSF Vendor Object” section).
- **contents** Software contained in the bundle.

A Bundle object template:

```
bundle
  tag <.....>
  title <.....>
  description <.....>
  revision <.....>
  vendor_tag <.....>
  architecture <.....>
  os_name <.....>
  os_release <.....>
  os_version <.....>
  contents <.....>
end #bundle
```

Example:

```bash
#bundle information

bundle
  tag B11_11QLISP
  title "11.11 QLISP sample driver"
  description "11.11 QLISP sample driver version 1.0"
  revision 11_11.1.0
  vendor_tag HP
  architecture HP-UX_B.11.11_32/64
  os_name HP-UX
  os_release ?.11.11
  os_version *
  contents QLISP11_11_1_0.qlisp1_0src,r=11_11.1.0,
  a=HPUX_B.11.11_32/64,v=HP
end #bundle
```
Writing a PSF Product Object

The Product information should follow the bundle information. If we consider product level packaging then bundle information should be omitted and product object should follow the vendor one. All filesets of the products should be included under this object.

**NOTE**  This document considers a fileset as a separate object.

The following is a list of all the product object attributes:

- **tag**: Identifier or name of the product.
- **title**: Detailed name of the product.
- **description**: A text string describing the product.
- **revision**: Revision (release) number of the product.
- **Architecture**: target systems on which product is installed.
- **os_name**: OS's on which Product will be installed.
- **os_release**: OS release on which product is installed.
- **os_version**: (optional), OS versions on which product will be handled.
- **directory**: The default, absolute pathname of the directory under which product will be installed.
- **is_locatable**: Product alternate directory.
- **vendor_tag**: Short name of the vendor (refer to the “Writing a PSF Vendor Object” section).
- **preinstall**: (optional) Path name of install pre-load product control script (holds same for all other control scripts).

Refer to the policies mentioned in the “SD-Product Attributes” section.

**NOTE**  The attributes revision, architecture, os_name, os_release, os_version are the same as in the bundle object if we consider bundle level packaging.

A product object template:

```
product
tag  <......>
title   <......>
revision <......>
vendor_tag <......>
directory <......>
preinstall <......>
is_locatable <......>
architecture <......>
os_name <......>
os_release <......>
os_version <......>
```

Example:

```
#product information
product
tag  QLISP11_11_1_0
title  "QLISP Sample drivers 1.0.0 for 11.11 DDK 1.0"
revision  11_11.1.0
vendor_tag  HP
```
NOTE The “end” of the product should be written after filesets.

Writing a PSF Subproduct Object

Subproduct object is optional. Subproducts information should be filled in here, following product information.

The following is a list of attributes under subproduct object:

- tag: The identifier for the Subproduct
- title: More detailed name of the Subproduct
- description: Text string describing the Subproduct

The policies are described in the “SD-Subproduct Attributes” section.

Since Subproduct object is optional, so are all its attributes.

A subproduct object template:

```
subproduct
tag <......>
title <......>
description <......>
contents
end
```

Example:

```
#Subproduct information
subproduct
tag QLISP11_11_1_0
title "QLISP Sample drivers 1.0.0 for 11.11 DDK 1.0"
description "This subproduct contains a collection of all sources of QLISP 1.0"
contents QLISP11_11_1_0.qlispsrc1.0,r=11.11.1.0,a=HPUX_B.11.11_32/64,v=HP
end #subproduct
```

Writing a PSF Fileset Object

Filesets object should follow product object if we do not consider subproducts. Since filesets are the atomic units, all the information of files that are to be packaged should be filled in here.

A list of all the attributes under Fileset object:

- tag: The identifier for the Fileset
- title: Detailed name of the Fileset
- Description: A text string describing the Fileset
- revision: Revision number of the Fileset
- Architecture: Target systems on which Fileset is installed.
Creating a Software Depot

Chapter 19

os_name

OS's on which Fileset will be installed

os_release

OS releases on which the Fileset is to be installed.

os_version

(optional) OS versions on which Fileset will be installed

ls_kernel

(optional) Defined if the Fileset contains OS kernel files

ls_reboot

(optional) Defined if the Fileset requires a system reboot after installation

preinstall

(optional) path name of install pre-load Fileset control script (holds same

file_permissions

Explicitly specifies default permissions for the files being packaged into
the Fileset

directory

Specifies a source directory in which subsequently listed filenames are
located (for this Fileset only)

file

Specifies the files to be packaged into a Fileset

The policies are mentioned under the "SD-Fileset Attributes" section.

NOTE

The revision, architecture, os_name, os_release, and os_version attributes are the same as in product object.

A Fileset object template:

```
fileset
tag <......>
description <......>
title <......>
revision <......>
archnitecture <......>
os_name <......>
os_release <......>
os_version <......>
file_permissions <......>
directory <......>
file <......>
end #fileset
end #product (if this is the last fileset under product)
```

Example:

```
fileset
tag qlispsrc1_0
description "DDK11.11.1.0 MS HBA sample drivers"
title QLISP 1.0 for DDK11_11
revision 11_11.1.0
architecture HP-UX_B.11.11_32/64
os_name HP-UX
os_release ?.11.11
os_version *
file_permissions -m 0555
directory ../.../ddkpack/sampldrvs/ms/scsihba/src=/opt/ddk/11.11/
sampldrvs/massstorage/qlisp/11.11.1.0/src/
file *
end #fileset
end #product (if this is the last fileset under product)
```
Step 4: Writing Control Scripts

This is (optional) and this section presents in detail the usage, templates and samples of all the SD control scripts.

Writing a Checkinstall Script

The purpose of a checkinstall script is to ensure that the target system has no product or fileset specific conditions which would cause either an installation failure or a runtime failure. This specifically excludes conditions that are tested elsewhere in the swinstall process such as inadequate disk space, unmounted volumes, unresolved fileset dependencies, inappropriate architecture, and so forth.

The details such as order of script execution, commands available for checkinstall script, its actions, etc. are explained in detail in the “Checkinstall” section.

The actions of a checkinstall script must be extremely unobtrusive since there is no commitment to installing at the time the scripts are run. The checkinstall script, like all control scripts, must be re-executable. It is possible for a checkinstall script to be executed numerous times within a single install session.

The checkinstall script should be used to test:

- Selected software.
- The system’s hardware configuration.
- Kernel configuration.
- The init state of the system.
- The system’s I/O structure.
- Software already installed.
- Revision, or other attributes of installed software.

Checkinstall Script template:

```bash
#!/sbin/sh
#########
# Script information and what it is used for
#########
Setup the SD environment by calling control utils

############################
Include code here to address things that SD may not know about.
For example code to check if the correct hardware is present or if an
incompatible product is present or check for if greater version of the
product is already installed
############################

Sample Checkinstall Script:

1.

#!/sbin/sh
######
# Product: DEMO
# Fileset: LATEST
# checkinstall
# @(#) $Revision: 11.11 $
######
# (c) Copyright 2003, Hewlett-Packard Company
```
```bash
###
UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]]
then
    msg ERROR "Cannot find $UTILS"
exit 1
fi
. $UTILS
exitval=$SUCCESS

PROD_VER=1 # Current version of the product
PROD_LIB=/usr/conf/lib/libenet.a
WHAT_BIN=/usr/bin/what

# Check if we are trying to install over a more recent version. If so, skip this
# installation. B.11.11 is hard-coded as the release
if [[ -x $WHAT_BIN && -e $PROD_LIB ]]; then
    what_str=
    $WHAT_BIN $PROD_LIB
    print $what_str | /usr/bin/grep -q 'B.11.11'
    if [[ $? = 0 ]]; then
        INST_VER=
        print $what_str | /usr/bin/awk '{print substr($0,index($0,"B.11.11."),2);}'
        if [[ $INST_VER -gt $PROD_VER ]]; then
            echo "ERROR: A newer version of the PCI FDDI has already been installed"
            echo " on this system (perhaps from a patch)."
            echo " The version in the kernel is B.11.11.$INST_VER."
            echo " The version attempted is B.11.11.$PROD_VER."
            echo " To force the installation, you must first swremove this"
            echo " product from the kernel, then retry the installation."
            exit $exitval
    fi
fi

2.
```

```bash
#!/sbin/sh
###
# Product: DEMO
# Fileset: LATEST
# checkinstall
# @(#) $Revision: 11.11 $
###
# (c) Copyright 2003, Hewlett-Packard Company
#
###
UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]]
then
    msg ERROR "Cannot find $UTILS"
exit 1
fi
. $UTILS
exitval=$SUCCESS #Anticipate success

#Make sure that product is not installed on an alternate root.
#Exlude the product from installation if it is alternate root.
if [[! $SW_ROOT_DIRECTORY != "/"]]
then
    echo "ERROR product cannot be installed on alternate root"
    exitval = $ERROR
    exitval $exitval
```
Writing a Preinstall Control Script

The purpose of a fileset’s preinstall script is to prepare the system for installation of the fileset. By the time the preinstall script for a particular fileset is executed, the actions of that fileset’s checkinstall script have determined that there are no fileset-specific impediments to installing the fileset. Then the Analysis Phase has ensured that there are no known system-specific impediments to installing the current fileset. All that remains is to prepare the system and to begin installing files. The steps to prepare the system are done by the fileset’s preinstall script.

A typical preinstall script is intended to kill processes that could interfere with installation, for example daemons that keep an executable file opened. Removal of obsolete software is done in a preinstall script.

The details such as order of script execution, commands available for checkinstall script, its actions, and so forth are explained in detail in the “Preinstall” section.

Preinstall Script template:

```bash
#!/sbin/sh

#########
# Product: DEMO
# Fileset: LATEST
# preinstall
# @(#) $Revision: 11.11 $

UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]]; then
    echo "ERROR: Cannot find $UTILS"
    exit 1
fi

. $UTILS
exitval=$SUCCESS

##############
# FUNCTIONS
##############

# Do the newconfig preparation step for configuration files.
for file in \
    /etc/rc.config.d/demo
    do
```
newconfig_prep $file
done
exit $exitval

Writing a Postinstall Control Script

Postinstall scripts are used to prepare for a kernel build when required by the install conditions. They also can drive events that must occur before a system reboot occurs.

The details such as order of script execution, commands available for checkinstall script, its actions, etc. are explained in detail under the “Postinstall” section. A postinstall script is an essential part of every kernel fileset and must be delivered with each kernel fileset because kernel configuration is done from within a postinstall script.

A postinstall script can be used to:

- Copy, move, or remove files.
- Conditionally create links to another location.
- Use newconfig_cp to conditionally copy files delivered to /usr/newconfig to a working location.
- Use mod_systemfile to modify /stand/system.
- Perform other actions that affect the successful build of a kernel.
Creating a Software Depot

Template Postinstall script:

```sh
#!/sbin/sh

#########
# Product: DEMO
# Fileset: LATEST
# postinstall
# @(#) $Revision: 11.11 $
#
#########
# (c) Copyright 2003, Hewlett-Packard Company
#
UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]] then
    echo "ERROR: Cannot find $UTILS"
    exit 1
fi
. $UTILS
exitval=$SUCCESS

# here is an example that adds the static driver "mydriver"
# to the target system file.

mod_systemfile ${SW_SYSTEM_FILE_PATH} -a mydriver
retval=$?
if [[ $retval -ne $SUCCESS ]] then
    msg ERROR "Could not enter 'mydriver' in the
        $(SW_SYSTEM_FILE_PATH)"
    [[ $exitval -ne $FAILURE ]] && exitval=$retval
fi
exit $exitval
```

Writing a Configure Control Script

A configure script is used to perform product or fileset installation actions that cannot be accomplished by simple unconditional file extraction from the software source media. Configure scripts are typically used to alter system specific files. The configure script is invoked either as part of the `swinstall` process, or as the result of the user invoking the `swconfig` command without the -u option. The configure script is not run when the user has invoked `swcopy` or `swremove`.

A configure control script can be used to:

- Create special device files.
- Append to existing files such as the fileset’s `rc.config.d` file.
- Conditionally establish symbolic links.
- Change file attributes.
Conditionally copy, move, or link files which have been delivered to a location under a private directory (that is, one only known to the packager).

Configure script template:

```sh
#!/sbin/sh
#########
Script information and what it is used for
#########
Setup the SD environment by calling control utils

Configure script action, for example, configure nettl

Sample Configure script:

```sh
#!/sbin/sh
#########
# Product: DEMO
# Fileset: LATEST
# configure
# @(#) $Revision: 11.11 $
#########

# (c) Copyright 2002, Hewlett-Packard Company
#
#
UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]]; then
    echo "ERROR: Cannot find $UTILS"
    exit 1
fi

$UTILS
exitval=$SUCCESS #Anticipate Success

#Configure nettl ( configure network tracing and logging command subsystem
database)

nettlconf -S -id 179 -name enet -class 12 -kernel ..............

exit $exitval
```

Writing an Unconfigure Control Script

The purpose of the unconfigure script is to undo most configuration changes that were made to the system by the corresponding configure script. A typical action in an unconfigure script would be the removal of device special files or the changing of a system configuration parameter in a file under /etc/rc.config.d.

An unconfigure script can be used to:

- Kill processes, including daemons, owned or spawned by files in the current fileset.
- Move or remove files and directories that were created by the corresponding configure script.
- Alter a value in the system’s configuration files.
- Remove client specific files such as log files.
Unconfigure script template:

```
#!/sbin/sh

#########
Script information and what it is used for

#########
Setup the SD environment by calling control utils


Configure script action, for example, remove nettl entry

Sample Unconfigure script:

```
#!/sbin/sh

#########
# Product: DEMO
# Fileset: LATEST
# configure
# @@(#) $Revision: 11.11 $

#
#
# (c) Copyright 2003, Hewlett-Packard Company
#

UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]]; then
echo "ERROR: Cannot find $UTILS"
exit 1
fi

$UTILS
exitval=$SUCCESS # Anticipate success

# Remove nettl entry.
ettlconf -delete 179

exit $exitval
```

Writing a Preremove Control Script

The purpose of the preremove script is to perform any necessary actions not done in the unconfigure script in preparation for the removal of the fileset's files. It should undo any actions taken in a postinstall script. For kernel filesets, it should modify the /stand/system file to delete entries such as driver names and configurable parameters whose functionality is part of the fileset.

A preremove script can be used to:

- Move or remove files and directories.
- Modify the /stand/system file.

Preremove script template:

```
#!/sbin/sh

#########
Script information and what it is used for

#########
Setup the SD environment by calling control utils


preremove script action, for example, remove system files for drivers
```
Sample Preremove Script:

```bash
#!/sbin/sh

########
# Product: DEMO
# Fileset: LATEST
# preremove
# @(#) $Revision: 11.11 $
########
#
# (c) Copyright Hewlett-Packard Company 2003
#

UTILS=/usr/lbin/sw/control_utils
if [ -f $UTILS ]
then
  . $UTILS
else
  echo "ERROR: Cannot find $UTILS"
  exit 1
fi
exitval=$SUCCESS

#############################################################

###########
# FUNCTIONS
########

DeleteDriverEntry()
{
  typeset retval

  for driver_name in \
    driver_1 \n    driver_2 \n    last_driver
  do
    mod_systemfile ${SW_SYSTEM_FILE_PATH} -d $driver_name
    retval=$?
    if [[ $retval -ne $SUCCESS ]]
      then
        [[ $retval -ne $SUCCESS && $exitval -ne $FAILURE ]] &&
        exitval=$retval
      fi
  done
}

exit $exitval
```
Writing a Postremove Control Script

The purpose of the postremove script is to perform any necessary cleanup actions after the fileset’s files have been removed. Any files which might have been created by the fileset, and which might not have been added to the IPD should be removed by the postremove script. A typical action in a postremove script would be the removal of newly emptied directories when those directories are the exclusive property of the fileset.

A postremove script can be used to:

- Remove newly emptied directories that are owned and used exclusively by the fileset. Replace any files that were set aside by the installation of the fileset.
- Any other actions that will bring the system closer to the condition it was in prior to the fileset’s installation.

Postremove script template:

```
#!/sbin/sh
#########
# Product: DEMO
# Fileset: LATEST
# postremove
# @(#) $Revision: 11.11 $
#########
# (c) Copyright Hewlett-Packard Company 2003
#
UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]]; then
echo "ERROR: Cannot find $UTILS"
exit 1
fi
. $UTILS exitval=$SUCCESS
# Remove formatted manual pages
for file in
  ninstall 
  installd
do
  rm -f /usr/share/man/cat1m.Z/$file.1m
done
# Remove files that might not have been removed
for file in 
  /opt/demo/app/log* 
  /dev/demo
do
  rm -f $file >/dev/null 2>&1
```

Sample Postremove Script:
done

############
Remove empty directories that this fileset owned exclusively.
for dir in \\
    /opt/demo/app \\
    /opt/demo/config \\
    /opt/demo/data \\
    /opt/demo
do
    rm -rf $dir >/dev/null 2>&1
done

exit $exitval

Step 5: Packaging the Components

Once PSF and control scripts are written, its time to package the components. SD provides the swpackage command to package these components. This section focuses on some of the basic usage of swpackage command. Details of which can be found as part of man page on swpackage.

NOTE
All SD commands are provided as manpages on the host as part of OS.

SD Components Packaging

As explained in the “Creating a Software Depot” section, a package consists of; a PSF file, the control scripts and the source/object files of product that needs to be packaged for distribution.

The command used to create a package into a target depot:

Syntax: swpackage -s *.psf
Inputs PSF file, depot directory
Options [-s] — PSF file
         [-d] — depot directory
         [-p] — Previews the package without creating a depot
Usage1 swpackage -s “PSF file path”
Usage2 swpackage -s “PSF file path” -d “depot directory path”

Syntax: swpackage -s *.psf
Usage2: swpackage -s “PSF file path” -d “depot directory path”

For example, if the PSF file of the package is in directory /home/qlisp/ and depot directory is /home/qlisp/driversource then:
#swpackage -s /home/qlisp/qlispdrv.psf
- or-
#swpackage -s /home/qlisp/qlispdrv.psf -d /home/qlisp/driversource

By default the depot directory is /var/spool/sw. If the user does not specify the depot directory as in usage1, then swpackage puts the depot into /var/spool/sw -or if the user specifies as in usage2, then swpackage puts the depot into user specified directory.
swpackage Process

Phase 1 Selection

When running swpackage, specify a PSF and any other options to include. The swpackage command begins the session by telling what source, target, software selections, and options used:

- Determine the product, subproduct, and fileset required for the structure.
- Determine which files are contained in each fileset.
- Determine the attributes associated with each objects.
- Check PSF syntax and terminates the session if any are encountered.

Phase 2 Analysis

The swpackage performs these checks during this phase:

1. Check for unresolved dependencies

   For every fileset in each selected product, swpackage checks to see if a requisite of the fileset is not also selected or not already present in the target depot. Unresolved dependencies within the product generate errors. Unresolved dependencies across products produce notes.

2. Check for software being repackaged

   For each selected product, swpackage checks to see if the product already exists in the target depot.
   - If it does exist, swpackage checks to see which filesets are being added (new filesets) or modified.
   - If it exists and all filesets are selected, swpackage checks to see if any existing filesets have been obsoleted by the new product.

3. Performing Disk Space Analysis (DSA)

   The swpackage verifies that the target depot has enough free disk space to package the selected products.
   - If adequate disk space is available for the packaging operation to proceed, swpackage writes a note to the log file to note the impact on disk space.
   - An error results if the package will encroach into the disk’s minfree space.
   - An error results if the package phase requires more disk space than is available.

4. Build

   When packaging a product, if the target depot does not exist, swpackage creates it. If it does exist, swpackage will merge new product(s) into it. Before a new depot directory is created, swpackage checks to see if this product version has the same identifying attributes as an existing product version. If all the identifying attributes match, you are re-packaging (modifying) an existing version. Otherwise, swpackage creates a new version in the target distribution.

   Each product is packaged in its entirety and when all specified products have been packaged successfully, the distribution’s global INDEX file is built/rebuilt.
   - It checks if the product is new or already exists. If it is new, create the product’s storage directory.
• For each fileset in the product, copy the fileset's files into their storage location (within the product's storage directory), and create the fileset's catalog (database information) files.
• After the individual filesets, create the product's informational files (meta-files).

NOTE
The swpackage does not register the depot created. It asks you to register. This is notified at the end of the swpackage process.

Step 6: Registering the Depot

After the depot is created, we need to register the depot to install and manage the depot further. The command used to register the depot is swreg.

Command usage: swreg -l depot

options: [-l] specifies the object to register or unregister. Exactly one level must be specified. For registering or unregistering a depot, use “-1 depot”, and for a root use “-1 root”.

For example: swreg -l /home/qlisp/qlispdrv

In this case qlispdrv is the target depot directory.
Installing a Depot

Once a SD depot is created and registered, it is ready for distribution. If a user wants to use the distribution, the depot should be installed first on a host. The command used to install a depot is `swinstall`.

Features of `swinstall`:

- Optional GUI.
- Compatibility filtering to ensure the software will run on the installed system.
- Ability to perform kernel rebuilding or rebooting.
- Automatic use of dependencies to automatically select software on which to operate (in addition to any software specified directly).
- Ability to run control scripts as part of the installation like all install and request scripts.

Command Usage

For GUI based `swinstall`, startup the GUI:

```
#/usr/bin/swinstall to generate TUI based
- or -
#/usr/bin/swinstall -i to generate a GUI based
```

Select Source

Specify the source depot that contains the software to install. The Specify Source dialog automatically lists the local host and default depot path.

Optionally, to specify another host system, type a source host name, or:

1. Click on the Source Host Name button. The system displays a dialog that lists all host system names contained in the `defaults.hosts` file (`$HOME/.sw/defaults.hosts` or `/var/adm/sw/defaults.hosts`).
2. Choose a host name from the list.
3. Click OK. The host name appears in the appropriate box in the Specify Source dialog.

Optionally, to specify the path to the depot, type a new path, or:

1. Click on the Source Depot Path button to display a list of registered depots on the source host.
2. Highlight one of the depots.
3. Click OK to make it appear in the Specify Source dialog.

Select Software

Use the Software Selection window to select the software to be installed.
Analysis
SD-UX analyzes the software that has been selected. The Analysis window displays status information about
the analysis process. When the analysis is complete and the host status shows Ready, click **OK** to start the
actual installation.

Installation
In this step, SD-UX proceeds with the actual installation.

swinstall usage, Command Line based:

**Syntax**
```bash
swinstall -s "depot source" "depot"
```

**Options**
```bash
[-s source]
[-p preview]
[-x command_option=value]
[@target_selections]
```

Basic Operation
In the command:

```bash
swinstall -s /home/qlispqlispdepot
```

The "/home/qlisp" is the depot source, where the depot resides or is downloaded. This command installs
qlispdepot under the directory specified in the PSF file.

Other useful operations are, for example:
```bash
swinstall -p -s /home/qlispqlispdepot
```
will only preview the installation, not install it.

To change the target install directory:
```bash
swinstall -s /home/qlispqlispdepot:/home/elsewhere
```
will change the install directory specified in the PSF file to /home/elsewhere.

---

**NOTE** When creating a depot using `swpackage` the developer should have set "is_locatable TRUE". Only then can we change the target directory with the `swinstall` command.

Target Selection
For example, the command:
```bash
swinstall -s /home/qlisp qlispdepot:otherhost
```
will install the depot into some other host.
**command_options**

Change the behavior of the command by changing its options. These can be set with a value of “true” or “false” and some of them require respective settings as attributes in the PSF files also. Some of the useful ones are:

- **ask=true** Executes a request script, which asks for a user response. The ask option has two possible values:
  - true — executes the request script (if one exists for the selected software) and stores the user response in a file named response.
  - false — (default for `swinstall` and `swconfig`), does not execute request scripts.

- **autoreboot=false** Normally set to false, indicating that installation of software requiring a reboot is not allowed from the command line. If set to true, this option allows installation of the software and automatically reboots the local host.

- **reinstall=false** Prevents SD-UX from reinstalling (overwriting) an existing revision of a fileset. If set to true, the fileset are reinstalled.

- **allow_downdate=false** Normally set to false, so installing an older version of software than already exists is disallowed. This prevents installing older versions by mistake. Additionally, many software products do not support this “downdating”.
  - If set to true, a previous version can be installed but SD-UX issues a warning message.

- **allow_multiple_versions=false** Normally set to false, so installed or configured multiple versions (for example, the same product, but a different revision, installed into a different location) are disallowed. If set to true, install and manage multiple versions of the same software.
Managing the Depot Software

Once the depot is installed, we do need to manage the software of the depot to suit our needs. SD provides the following `swlist` and `swcopy` depot managing commands.

**swlist**

The `swlist` command can display lists of registered depots residing on a host.

Command Usage `swlist`

Options

- `-l` depot @ hostA — to list all depots on remote host A.
- `-l` product Bundle — to list all products in the specific Bundle.
- `-l` fileset Bundle — to list all filesets in a specific Bundle.
- `-l` fileset Product — to list all filesets in a specific product. For example:
  
  ```
  swlist -l depot
  swlist -l product QLISPBUNDLE
  swlist -l fileset QLISPBUNDLE
  swlist -l fileset QLISPDRVSRC
  ```

**swcopy**

The `swcopy` command copies or merges `software_selections` from a software source to one or more software depot `target_selections`. This is a GUI and command line based command. The GUI steps are same as for `swinstall` commands.

Command Usage `swcopy [-s] source [@[target selections]]`

Options

- `-s` — Source depot ([host][:][/directory) specifies the source depot from which software is installed or copied. The default source type is directory.
- `[@ target selections] - ([host][:][/directory)

Example:

```
swcopy -s /home/ddk/depots/QLISP/QLISPdrvsrc /opt/qlisp
```

Here, `/home/ddk/depots/QLISP/QLISP drvsrc` is a depot path of QLISP driver sources and `swcopy` will copy the QLISP `drvsrc` product to `/opt/qlisp` depot path on local host.

Example:

```
swcopy -s /home/ddk/depots/QLISP/QLISPdrvsrc @ hostA:/opt/qlisp
```

Will copy the QLISP `drvsrc` product to `/opt/qlisp` depot path on host A.
Removing the Installed Software

Any installed software can be removed by using the `swremove` command.

**Command usage:** `swremove Bundle/product/product`.

- `#swremove Bundle`
  Removes all software installed from the Bundle.

- `#swremove Bundle.product`
  Removes all software installed from the product contained in the bundle.

- `#swremove Bundle.product.fileset`
  Removes all software installed from the fileset contained in a product which part of a bundle.

- `#swremove product`
  Removes all software in the product.

- `swremove QLISPBUNDLE`
  Removes all driver sources, binaries, documents from the QLISP bundle.

- `swremove QLISPBUNDLE.QLISPDRVSRC`
  Removes only the driver sources but not binaries and documents.

- `swremove QLISPDRVSRC`
  If QLISPDRVSRC is not part of any bundle.

**NOTE** When `swremove` is done, it only removes the files (software). It does not remove the directory path where the files are installed.

For example, if qlisp driver sources are installed on a system at `/opt/ddk/src/qlisp/*.ch`, `#swremove QLISPDRVSRC` will remove `*.ch` sources, but can still see an empty `/opt/ddk/src/qlisp` directory there.

**Command Messages Logging**

The SD commands’ messages are logged in `/var/adm/sw/swagent.log`. 
Examples

Creating a Package

PSF

For example, qlisp.psf

```
vendor
tag HP
title Hewlett-pakard
description Hewlett
end
bundle
tag B11_1QLISP
title "11.11 QLISP sample driver Version 1.0"
description "11.11 QLISP sample driver Version 1.0 with only sources"
revision 11_11.1.0 ( suggested use - major:minor:release )
vendor_tag HP
architecture HP-UX_B.11.11_32/64 ( for 32 and 64 bit systems)
os_name HP-UX
os_release ?.11.11
os_version *
contents QLISP11_11_1_0.qlispsrcfiles1_0,r=11_11.1.0,a=HP
UX_B.11.11_32/64,v=HP
end
product
tag QLISP11_11_1_0
title "QLISP 1.0 Sample drivers sources "
revision 11_11.1.0
vendor_tag HP
directory /opt/ddk/11.11 ( where the QLISP sample sources product is
installed)
is_locatable TRUE ( while installing the user can relocate the
above install path)
preinstall scripts/preinstall (scripts is a directory)
postinstall scripts/postinstall
architecture HP-UX_B.11.11_32/64
os_name HP-UX
os_release ?.11.11
os_version *
fileset
tag qlispsrcfiles1_0
description "QLISP 1.0 sources only"
title QLISPI_0SOURCES
revision 11_11.1.0
architecture HP-UX_B.11.11_32/64
os_name HP-UX
os_release ?.11.11
os_version *
file_permissions -m 0555
directory /samldrvrs/ms/scsihba=/opt/ddk/11.11/sampldrvrs/
masstorage/qlisp/11.11.1.0
file *
end #fileset
```
end #product
Control Scripts Files

Preinstall
This script checks if there are any obsolete qlisp configuration files, removes them and continues the installation:

```bash
#!/sbin/sh
########
# Product: QLISP11_11_1_0
# Fileset: qlispsrcfile1_0
# preinstall
# @(#) $Revision: 11.11$
########
#
# (c) Copyright Hewlett-Packard Company 2003
#
########
# set up the SD environment
UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]]
then
    echo "ERROR: Cannot find $UTILS"
    exit 1
fi
. $UTILS
exitval=$SUCCESS          # Anticipate success

# Remove any obsolete qlisp files
rm -f ${SW_ROOT_DIRECTORY}/usr/conf/master.d/qlisp
exit $exitval
```

Postinstall
This script installs the driver into kernel by adding qlisp into the system file:

```bash
#!/sbin/sh
########
# Product: QLISP11_11_1_0
# Fileset: qlispsrcfile1_0
# postinstall
# @(#) $Revision: 11.11$
#
# (c) Copyright Hewlett-Packard Company 2003
#
########
# UTILS="/usr/lbin/sw/control_utils"
if [[ ! -f $UTILS ]]
then
    echo "ERROR: Cannot find $UTILS"
exit 1
fi
. $UTILS
exitval=$SUCCESS          # Anticipate success
```
Creating a Software Depot

Examples

```
#DRV_NAME=qlisp
if [[ $SW_ROOT_DIRECTORY = "/" && -z $SW_DEFERRED_KERNBLD ]]
 then
 mod_systemfile ${SW_SYSTEM_FILE_PATH} -a $DRV_NAME
 retval=$?
 fi
fi
```

Commands

Create a Package

```
#swpackage -s qlisp.psf -d /home/user/qlispdrvdepots
```

Creates a depot under /home/user/qlispdepots.

```
#ls /home/user/qlispdrvdepots
```

Will have:

```
B11_11QLISP
QLISP11_11_1_0
catalog
swagent.log
```

```
# ls /home/user/qlispdrvdepots/QLISP11_11_1_0
```

Has qlisp sources under directory specified in the "=target dir path" across the "directory" fileset of the PSF.

```
#ls /home/user/qlispdrvdepots/catalog
```

Has INDEX file with creation time and all information set in PSF.

```
#cd B11_11QLISP
```

Has the pfiles (product files) in turn will have INDEX and INFO files which are self explanatory.

```
#cd QLISP11_11_1_0
```

Has pfiles same as in for B11_11DDK.

Register the Depot

```
#swreg -l depot /home/user/qlispdrvdepots
```

Registers the qlisp depot.

Installing the Depot

```
#swinstall -s /home/user/qlispdrvdepots B11_11QLISP
```

Installs the qlisp sources under /opt/ddk/11.11/sampldrvs/massstorage/qlisp/11.11.1.0.
List the Product

```bash
#swlist
Lists B11_11QLISP with revision and its title.
#swlist -l product B11_11QLISP
Lists QLISP11_11_1_0 product.
#swlist -l fileset B11_11QLISP
Lists qlispsrcfiles1_0.
```

Remove the Sources from the Product

```bash
#swremove B11_11QLISP
Removes all files under /opt/ddk/11.11/sampldrvs/massstorage/qlisp/11.11.1.0/.
#swremove B11_11QLISP.QLISP11_11_1_0
```

---

**NOTE**

If we have more than one product in a bundle and wish to remove only one product then we can use the above command. If we have only one product in an bundle, removing bundle itself will remove all the products, therefore the above command is not useful in that case.

Removes the files from the same location.

```bash
#swlist
Will not show B11_11QLISP and product and filesets under it.
```