Tru64 UNIX

Guide to Preparing Product Kits

Part Number: AA-RH9SD-TE

June 2001

Product Version: Tru64 UNIX Version 5.1A

This manual describes how to create, deliver, and install software product kits to use on Compaq Tru64 UNIX operating systems.

Compaq Computer Corporation
Houston, Texas

Contents

About This Manual

A product kit is the standard mechanism by which software products are delivered to and maintained on a Compaq Tru64™ UNIX operating system. This manual describes the procedures for creating, installing, and managing the collections of files and directories that make up a product kit to be installed on a customer's system. Kits can be distributed on CD-ROM, diskette, or magnetic tape.

Audience

This manual is intended for software developers who are responsible for creating product kits. They are expected to have experience with UNIX based operating systems, shell script programming, and system administration.

New and Changed Features

The following list describes the major changes made to this manual:

- Added information about restrictions on subset names (*Section 3.1*).
- Divided the *Creating Subsets* chapter into three chapters: *Preparing Subsets* (*Chapter 3*), *Creating Subset Control Programs* (*Chapter 4*), and *Producing and Testing Subsets* (*Chapter 5*).
- Updated the SCP library routines in *Table 4–1* and the global variables in *Table 4–4*.
- Added a new method to determine subset installation status (*Section 4.7*).

Note

This information replaces former methods of determining subset installation status and may require changes to the way you code subset control programs.

- Added restrictions for kernel product kits used to support third party hardware (*Section 7.1*).
- Removed instructions for building a consolidated firmware CD-ROM from the appendices. This information will be updated and relocated to *Compaq Tru64 UNIX Best Practices*.

Previous versions of this manual are available on the World Wide Web at the following URL: **`http://www.tru64unix.compaq.com/docs/`**. Technical updates to this manual are also available from the same location.

Organization

This manual is organized as follows:

Chapter 1	Introduces the kit-building process
Chapter 2	Describes how to create and populate kit directories
Chapter 3	Describes how to organize product files into subsets and create kit production files
Chapter 4	Describes how to create subset control programs
Chapter 5	Describes how to produce and test subsets
Chapter 6	Describes how to create, test, and deliver user product kits
Chapter 7	Describes how to create, test, and deliver kernel product kits
Glossary	Defines terms used in this manual

Related Documentation

You may find the following documents helpful when preparing product kits:

- *Sharing Software on a Local Area Network* describes Remote Installation Services (RIS) and Dataless Management Services (DMS). RIS is used to install software across a network instead of using locally mounted media. DMS allows a server system to maintain the root, `/usr`, and `/var` file systems for client systems. Each client system has its own root file system on the server, but shares the `/usr` and `/var` file systems.

  This manual may be helpful if you are preparing a product kit that will be installed in a RIS environment.

- *Writing Device Drivers* provides information for systems engineers who write device drivers for hardware that runs the operating system. Systems engineers can find information on driver concepts, device driver interfaces, kernel interfaces used by device drivers, kernel data structures, configuration of device drivers, and header files related to device drivers.

  This manual may be helpful if you are preparing product kits for a device driver.

- The *Installation Guide* describes the procedures to perform an Update Installation or a Full Installation of the operating system on all supported processors and single-board computers. It explains how to prepare your system for installation, boot the processor, and perform the installation procedure.
- The *Installation Guide — Advanced Topics* describes the advanced installation procedures such as Installation Cloning, Configuration Cloning, and how to customize the installation process with user supplied files.
- *System Administration* describes how to configure, use, and maintain the operating system. It includes information on general day-to-day activities and tasks, changing your system configuration, and locating and eliminating sources of trouble. This manual is intended for the system administrators responsible for managing the operating system. It assumes a knowledge of operating system concepts, commands, and configurations.
- *Reference Pages Sections 8 and 1m* describe commands for system operation and maintenance and are intended for system administrators. In printed format, this is divided into two volumes.
- The *Release Notes* describe known problems you might encounter when working with the operating system and provides possible solutions for those problems. The printed format also contains information about new and changed features of the operating system, as well as plans to retire obsolete features of the operating system. Obsolete features are features that have been replaced by new technology or otherwise outdated and are no longer needed. These are intended for anyone installing the operating system or using the operating system after it is installed.

The Tru64 UNIX documentation is available on the World Wide Web at the following URL:

```
http://www.tru64unix.compaq.com/docs/
```

Icons on Tru64 UNIX Printed Manuals

The printed version of the Tru64 UNIX documentation uses letter icons on the spines of the manuals to help specific audiences quickly find the manuals that meet their needs. (You can order the printed documentation from Compaq.) The following list describes this convention:

G	Manuals for general users
S	Manuals for system and network administrators

P	Manuals for programmers
R	Manuals for reference page users

Some manuals in the documentation help meet the needs of several audiences. For example, the information in some system manuals is also used by programmers. Keep this in mind when searching for information on specific topics.

The *Documentation Overview* provides information on all of the manuals in the Tru64 UNIX documentation set.

Reader's Comments

Compaq welcomes any comments and suggestions you have on this and other Tru64 UNIX manuals.

You can send your comments in the following ways:

- Fax: 603-884-0120 Attn: UBPG Publications, ZKO3-3/Y32
- Internet electronic mail: `readers_comment@zk3.dec.com`

  A Reader's Comment form is located on your system in the following location:

  `/usr/doc/readers_comment.txt`

Please include the following information along with your comments:

- The full title of the manual and the order number. (The order number appears on the title page of printed and PDF versions of a manual.)
- The section numbers and page numbers of the information on which you are commenting.
- The version of Tru64 UNIX that you are using.
- If known, the type of processor that is running the Tru64 UNIX software.

The Tru64 UNIX Publications group cannot respond to system problems or technical support inquiries. Please address technical questions to your local system vendor or to the appropriate Compaq technical support office. Information provided with the software media explains how to send problem reports to Compaq.

Conventions

The following conventions are used in this manual:

`%` `$`	A percent sign represents the C shell system prompt. A dollar sign represents the system prompt for the Bourne, Korn, and POSIX shells.
`#`	A number sign represents the superuser prompt.
`%` **`cat`**	Boldface type in interactive examples indicates typed user input.
`file`	Italic (slanted) type indicates variable values, placeholders, and function argument names.
[|] { | }	In syntax definitions, brackets indicate items that are optional and braces indicate items that are required. Vertical bars separating items inside brackets or braces indicate that you choose one item from among those listed.
. . .	In syntax definitions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.
`cat`(1)	A cross-reference to a reference page includes the appropriate section number in parentheses. For example, `cat`(1) indicates that you can find information on the `cat` command in Section 1 of the reference pages.
$\boxed{\text{Return}}$	In an example, a key name enclosed in a box indicates that you press that key.
Ctrl/*x*	This symbol indicates that you hold down the first named key while pressing the key or mouse button that follows the slash. In examples, this key combination is enclosed in a box (for example, $\boxed{\text{Ctrl/C}}$).

1 Introducing Product Kits

This manual provides product and kit developers with the proper method to create a product kit. The following topics are discussed in this chapter:

- An overview of product kits (Section 1.1)
- Defining the different product types and describing the sample product used in this guide (Section 1.2)
- Describing the available formats for layered product kits (Section 1.3)
- Illustrating the kit-building process (Section 1.4)

1.1 Overview

A product kit is the collection of files and directories that represent new or upgraded software to be installed onto a customer's system. The kit contains not only the actual files and directories that compose the product, but also includes the supporting files that are required to install the product on the system. The product kit is the standard mechanism by which most products are delivered and maintained on a customer's system. Kits for user and kernel products can be distributed on a CD–ROM, diskette, or tape for installation onto the customer's system.

Before building a kit, consider the kind of product for which you are building a kit:

- Does it run in user space or kernel space?
- Is it used during the initial installation and bootstrap of the operating system?

The answers to these questions determine the type of format you choose, the type of medium you use to distribute the kit, and the installation procedures that your users run when they install the kit on their systems.

This chapter helps you answer these questions. It describes the product types supported by the kit-building process and the options for packaging and installing the kit on the customer's system. It leads you through the steps involved in building kits for the various kinds of products, and it describes the installation options that the operating system supports.

After you determine the type of product kit that you are creating, you can use the specific chapters in this manual as shown in Table 1–1:

Table 1–1: Using This Manual

ALL product kits	
1. Introducing Product Kits	
2. Creating Kit Directories	
3. Preparing Subsets	
4. Creating Subset Control Programs[a]	
5. Producing and Testing Subsets	
ONLY user product kits	**ONLY kernel product kits**
4. Producing User Product Kits	5. Producing Kernel Product Kits

[a] Optional for user product kits, required for kernel product kits.

This manual uses the fictitious Orpheus Document Builder (ODB) product to demonstrate how to build kits for each product type. `OAT` is the code assigned to *Orpheus Authoring Tools, Inc.*, the fictitious company developing the ODB product, and `100` is the product version number. The same product is used for each type of product kit, but Chapter 6 and Chapter 7 describe the files specific to user and kernel product kits.

1.2 Product Types

The process described in this book lets you deliver layered products for a customer's system. A layered product is any software product that is not part of the base operating system. There are two kinds of layered products:

- User product

  A user product runs in user space. Commands and utilities are in this category, as are applications such as text editors and database systems. Users interact directly with user products, for example, through commands or window interfaces.

- Kernel product

  A kernel product runs in kernel space. Users do not directly run kernel products, but the operating system and utilities access them to perform their work. For example, a device driver is one common type of kernel product. A user runs an application or utility, which generates system requests to perform operations such as opening a file or writing data to a disk. The system determines which device driver should service this request and then calls the required driver interface.

1.3 Kit Formats

Before being copied onto the distribution media (diskette, CD-ROM, or tape), the product files are gathered into subsets. A subset groups together related files and specifies whether they are required or optional for the product. You can copy the product files onto the distribution media in one of the following formats:

- Compressed `tar` format

  In compressed `tar` format, the product files belonging to the same subset are written to the distribution media as a single file. Kits for user and kernel products should be produced in compressed `tar` format. During installation, the `setld` utility uncompresses the files and moves them onto the customer's system, preserving the files' original directory structure. The `gentapes` and `gendisk` utilities can create kits in compressed `tar` format.

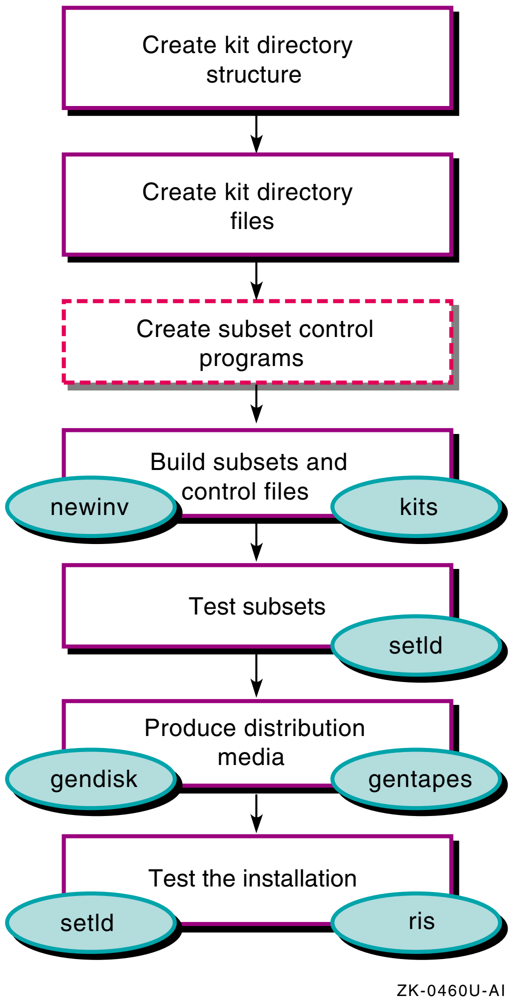
- Direct CD–ROM (DCD) format

  In direct CD-ROM (DCD) format, the files are written to any disk media (CD–ROM, hard disk, or diskette) as a UNIX file system (UFS). Subsets distributed in DCD format cannot be compressed. The `gendisk` utility can create kits in DCD format.

1.4 Kit-Building Process

Figure 1–1 shows the process of creating and packaging a kit. In the figure, boxes drawn with dashed lines represent optional steps; for example, you do not have to create subset control programs if your kit requires no special handling when it is installed. In Figure 1–1, the commands enclosed in ovals perform the associated steps of the kit-building process.

Figure 1–1: Steps in the Kit-Building Process


Create kit directory structure
Create kit directory files
Create subset control programs
Build subsets and control files
newinv
kits
Test subsets
setld
Produce distribution media
gendisk
gentapes
Test the installation
setld
ris
ZK-0460U-AI

The kit-building process consists of the following steps:

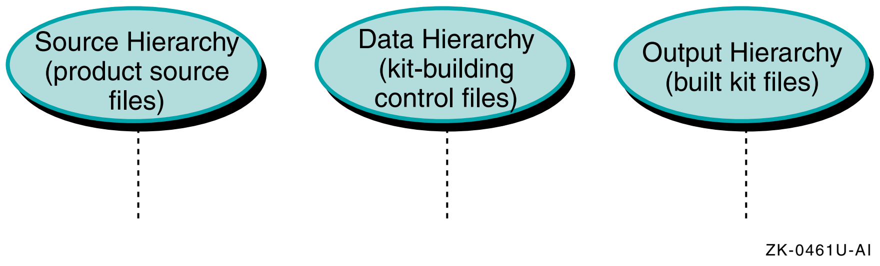
1. Create the kit directory structure that contains the source files.

   On the development system, create the following directory structure for the kit you want to build:

   - A source hierarchy, containing all the files that make up the product
   - A data hierarchy, containing files needed to build the kit
   - An output hierarchy, to contain the result of the kit-building process; one or more subsets that make up the product kit

Figure 1–2 shows these directory hierarchies.

Figure 1–2: Kit Directory Hierarchies


Source Hierarchy (product source files)
Data Hierarchy (kit-building control files)
Output Hierarchy (built kit files)
ZK-0461U-AI

This directory structure is the same for user product kits and kernel product kits. Only the contents of these directories differ among the product types. For example, a kernel product kit needs additional files that are unique to this specific kit type. See Chapter 6 and Chapter 7 for more information about the requirements for each product kit type.

2. Create kit production files.

   This includes a master inventory file containing information about each file in the subset, a key file to define product attributes such as the product name, product version, subset definitions, and additional files for kernel product kits.

3. Create subset control programs (if needed).

   The `setld` utility can call a subset control program (SCP) to perform installation steps specific to your kit. You supply an SCP for your kit only if the product requires special installation steps, such as those needed for kernel product kits. The SCP is optional for user products. Most layered products supply a subset control program, though the actions the programs perform differ for each product type. For example, the subset control program for a kernel product may call the `kreg` utility to maintain the system file that registers kernel layered products, while the subset control program for a user product would not.

4. Build subsets and control files.

   Before transferring your kit onto distribution media, organize the product files into subsets. Subsets group together related files. For example, one subset could contain optional product files, while another subset could contain the files required to run the product. The `kits` utility creates subsets according to the specifications you define in the master inventory and key files. Invoke the `kits` utility from the same directory where the master inventory file is located. See Chapter 3 for information about the master inventory and key files.

5. Test subsets.

   You must test your subsets to ensure that they can be loaded onto a running system, that the product runs on the system, and that the subsets can be deleted. Subset testing includes loading all subsets onto a running system and deleting all subsets from a running system. If your kit includes optional subsets, you also should load the mandatory subsets onto a running system to determine if the product works as expected. If not, you may have to reclassify some optional subsets as mandatory.

6. Produce distribution media.

   When you have created the subsets for the product, you are ready to package the kit with either the `gendisk` or `gentapes` utility. At this point, you must decide whether to create the kit in DCD format or in `tar` format. If you are creating a kit for a kernel product, you may need to modify the kit and add files to support your system's bootstrap link.

7. Test product kit media.

   After you have successfully created the kit, you should test its installation from the new media. Chapter 6 and Chapter 7 tell you how to test the installation of each of the product kit types.

2 Creating Kit Directories

After you develop a software product, you package the product files to process them into a kit. First you must organize these files by function and use, then place them into a kit-building directory structure. When you design the kit-building directory structure, consider where you want to place the product files on the customer's system and then create a kit directory structure that closely mirrors that on the customer's system.

A kit developer must perform the following actions to create kit directories:

1. Obtain a unique three-letter manufacturer's product code (Section 2.1)
2. Create the directory structure needed to build a product kit (Section 2.2)
3. Populate the source directory on the kit-building system (Section 2.3)

2.1 Obtaining a Unique Product Code

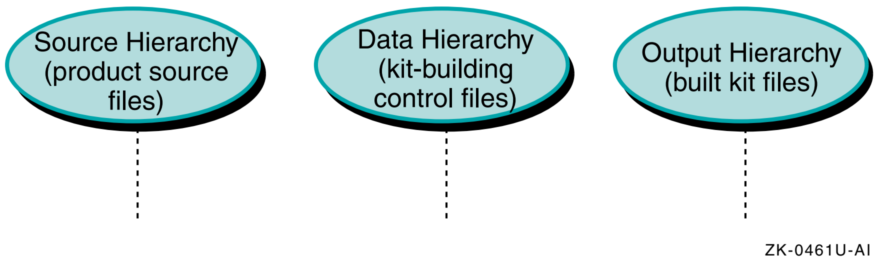
Before you can create a product kit, you must have a unique three-letter product code. To obtain this product code, send electronic mail to **`product@dssr.sqp.zko.dec.com`**. You use this product code and a product version number that you assign to name your product-specific subdirectories.

Examples in this book use `OAT` as the prefix as the unique three-letter product code for the Orpheus Document Builder (ODB) product kit. Assuming this is the first release of the product, the examples use `100` as the version number.

2.2 Creating a Kit Building Directory Structure

To create a kit, you need three separate directory hierarchies on the kit development system. Figure 2–1 shows these directory hierarchies.

Figure 2–1: Kit Directory Hierarchies


Source Hierarchy (product source files)
Data Hierarchy (kit-building control files)
Output Hierarchy (built kit files)
ZK-0461U-AI

The following definitions describe each directory hierarchy:

- Source hierarchy

  The source hierarchy exactly mirrors the directory structure into which customers install your finished kit. You must place each file that is to become part of your kit into the required directory in the source hierarchy. You can create the source hierarchy under any directory you choose.

- Data hierarchy

  The data hierarchy contains the following files to specify the contents of the kit and how it is organized:

  – A master inventory file lists each of the files in the kit and defines which subset contains each file.

  – A key file specifies the kit's attributes, such as the product name and version and whether the subsets are in compressed or uncompressed format.

  – A subdirectory named `scps` contains any subset control programs that the product requires.

  – Additional files may be required, depending on the kit type.

  The `kits` utility is run from this data directory. There is no specific requirement for the location of the data hierarchy, but it is good practice to place it under the same directory as the source hierarchy.

- Output hierarchy

  The output hierarchy contains the results of building the kit in the same format that the distribution media will contain when it is delivered to the customer. There is no specific requirement for the location of the output hierarchy, but it is good practice to place it under the same directory as the source and data hierarchies.

Use the `mkdir` command to create your kit directories. For example:

```
# mkdir -p /mykit/src /mykit/data /mykit/output
```

2.3 Populating the Source Directory

The components of a kit can be installed in any directory on the customer's system. However, guidelines exist for file placement. The standard system directory structure separates files by function and use and is designed for efficient organization.

This section discusses the following topics:

- Using standard directory structures (Section 2.3.1)
- Using context-dependent symbolic links (Section 2.3.2)
- Placing files in the kit source directory (Section 2.3.3)
- Setting up the sample product kit source directory (Section 2.3.4)

You can choose any method for populating the source hierarchy. For example, you could create a `Makefile` to use with the `make` command, or you could copy files with the `cp` command.

2.3.1 Using Standard Directory Structures

A standard directory structure has the following advantages:

- Avoids name conflicts

  When a layered product installs a file that overwrites a file shipped by another product, it is known as a name space conflict. Shipping the files in the product-specific `opt` subdirectories of root, `usr` and `var` avoids this conflict because each three-letter product code is unique to a particular manufacturer. The product-specific directories for the examples in this manual are `/opt/OAT100`, `/usr/opt/OAT100`, and `/usr/var/opt/OAT100`.

  Note

  Always place files to be installed into the `var` file system under your kit's `/usr/var` directory.

- Easy access to kit components

  If disk partition restructuring or product maintenance becomes necessary, it is easier to find all of your kit if its components are in the `/opt` directories rather than scattered throughout the standard directories.

- Serves multiple versions of the same product to different clients

  Exporting software to share across a network is simplified and more secure. You need to export only the specific directories under `/opt`, `/usr/opt`, and `/usr/var/opt` that contain the product you want, then create links on the importing system. You can set up a server with multiple versions of a given product, using the links created on the client systems to determine which version a given client uses. In this way, you can maintain software for multiple dissimilar hardware platforms on the same server.

Specific directory requirements exist for each type of product kit. In some cases, additional files are required for the kit to build successfully.

- You do not need any additional installation files for a user product.
- Section 7.2 describes the additional installation files you need for a kernel product. Figure 7–1 shows these files.

Install product files in product-specific subdirectories of the root (`/`), `/usr`, and `/usr/var` directories, as described in the following list:

- Boot files reside under `/opt`

  Files that are required at bootstrap time, such as device drivers, go into in a product-specific subdirectory of the `/opt` directory, such as `/opt/OAT100`. This also includes any files to be accessed before file systems other than root are mounted.

- Read-only files reside under `/usr/opt`

  Read-only files (such as commands), startup files (not modified by individual users), or data files go into a product-specific subdirectory of the `/usr/opt` directory, such as `/usr/opt/OAT100`.

- Read/write files reside under `/usr/var/opt`

  Files that users can read and modify, such as data lists, go into a product-specific subdirectory of the `/usr/var/opt` directory.

2.3.2 Using Context-Dependent Symbolic Links

If you are preparing a product kit that may run on a cluster, you need to create context-dependent symbolic links (CDSLs) to member-specific files in addition to using shared files in your product kit's inventory.

- A member-specific file is used by a specific cluster member. The contents of a member-specific file differ for each cluster member, and each member has its own copy of a member-specific file.

  For example, the `sysconfigtab` file contains information that can be different for each cluster member, so the `/etc/sysconfigtab` file is a CDSL that points to the member's `sysconfigtab` file.

- A context-dependent symbolic link (CDSL) is a special kind of symbolic link that points to a member-specific file. The CDSL references the member-specific file for the member that accesses the CDSL to determine its target.
- A shared file is used by all members of a cluster. There is only one copy of a shared file.

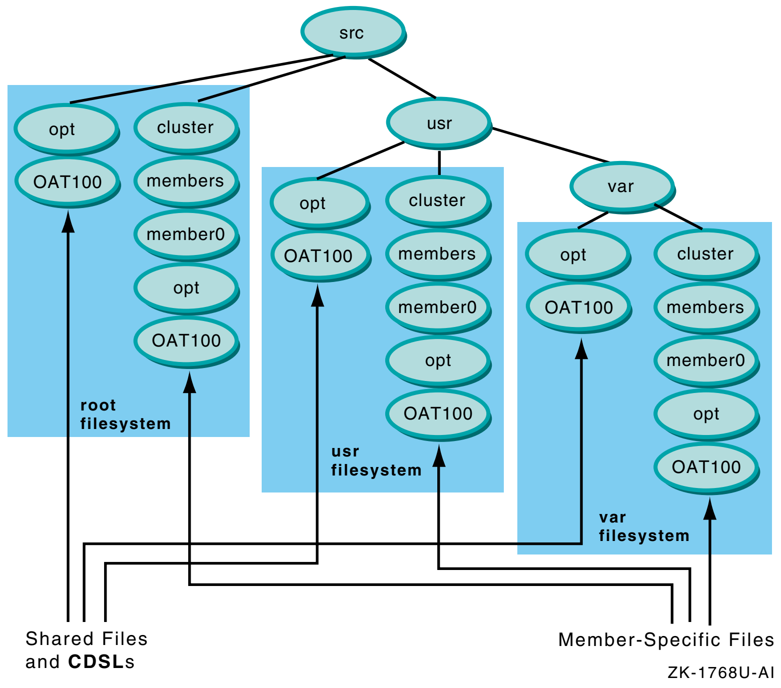
  For example, executable files may be shared by all cluster members.

This section provides the following information:

- When to use CDSLs in your product kit (Section 2.3.2.1)
- How to identify CDSLs (Section 2.3.2.2)
- How to create CDSLs for your product kit (Section 2.3.2.3)
- Restrictions on using CDSLs in your product kit (Section 2.3.2.4)

Figure 2–2 shows member-specific directories in the ODB product kit source hierarchy.

Figure 2–2: Member-Specific Directories in Source Hierarchy



Note

Layered product kits always should refer to CDSLs and not to the corresponding member-specific files. Doing this isolates CDSL awareness from the layered product and keeps it in the file system hierarchy, making it easier for you to maintain and upgrade your layered product kit. CDSL references access the correct file whether the product is installed on a cluster or on a single system, and continue to work if file types change in a future version of the product.

For example, refer to the `/usr/var/X11/Xserver.conf` file rather than the `/usr/var/cluster/members/member0/X11/Xserver.conf` file. These are the same if the `/usr/var/X11/Xserver.conf` file is a CDSL.

2.3.2.1 Knowing When to Use CDSLs

If your product kit might run on a cluster, you may need to create CDSLs.

- Use CDSLs if you need a file to be different on every machine running the product kit software.
- Use shared files if the file is always the same on every machine running the product kit software.

Do not make a directory into a CDSL. After a directory has been made a CDSL, you cannot change it back to a regular directory. If a directory contains all member-specific files, make each file a CDSL. This allows shared files to be placed in the directory in a future version.

2.3.2.2 Identifying CDSLs

You can identify a CDSL by the presence of the `{memb}` variable in its pathname. For example:

```
lrwxrwxrwx  1   root    system      57 May 19 10:54 /etc/sysconfigtab -> \
    ../cluster/members/{memb}/boot_partition/etc/sysconfigtab
```

Note

The backslash (\) character in this example indicates line continuation, and is not present in the actual output.

To resolve a CDSL's pathname, the kernel replaces the {memb} variable with the string member*N*, where *N* is the member ID of the cluster member that is referencing the file. If a cluster member with member ID 2 is accessing the /etc/sysconfigtab file in this example, the pathname is resolved to /cluster/members/member2/boot_partition/etc/sysconfigtab.

Note

Single systems always resolve the {memb} variable to member0.

2.3.2.3 Creating CDSLs

Follow these steps to create CDSLs:

1. Identify files in your kit that need to be member-specific. Looking at the sample OAT100 product kit source hierarchy shown in Figure 2–3, these member-specific files are odb.conf and odb_log.
2. Determine the file system where the CDSLs should reside: root, usr, or usr/var. The odb.conf file is in the root file system, and the odb_log file is in the usr/var file system.
3. Log in as root or use the su command to gain superuser privileges.
4. Create the necessary parent directories for the CDSLs in your source hierarchy. For example:

```
# mkdir -p /mykit/src/opt/OAT100
# mkdir -p /mykit/src/usr/var/opt/OAT100/log_files
```

5. Create the necessary member-specific directories in your source hierarchy. For example:

```
# mkdir -p /mykit/src/cluster/members/member0/opt/OAT100
# mkdir -p /mykit/src/usr/var/cluster/members/member0/opt/OAT100/log_files
```

6. Copy member-specific files to the member-specific areas you created in the previous step. For example, if your developer-supplied files are under /mnt, use commands similar to the following:

```
# cd /mykit/src/cluster/members/member0/opt/OAT100
# cp /mnt/opt/OAT100/odb.conf .
# cd /mykit/src/usr/var/cluster/members/member0/opt/OAT100/log_files
# cp /mnt/usr/var/opt/OAT100/log_files/odb_log .
```

Note

Member-specific files are always shipped in member0.

7. Use the `ln -s` command to create the necessary CDSLs in your source hierarchy. For example:

```
# cd /mykit/src/opt/OAT100
# ln -s ../../cluster/members/{memb}/opt/OAT100/odb.conf
# cd /mykit/src/usr/var/opt/OAT100/log_files
# ln -s ../../../cluster/members/{memb}/opt/OAT100/log_files/odb_log
```

   Although you created `cluster/members/member0` directories, you linked to `cluster/members/{memb}` directories. When your product kit is installed, this lets the kernel resolve the `{memb}` variable to `member`*N*, where *N* is the member ID of the cluster member that is referencing the file. Single systems are always `member0`.

8. Use the `ls -l` command to verify the CDSLs. For example:

```
# ls -l /mykit/src/opt/OAT100/odb.conf
lrwxrwxrwx …odb.conf -> ../../cluster/members/{memb}/opt/OAT100/odb.conf
# ls -l /mykit/src/usr/var/opt/OAT100/log_files/odb_log
lrwxrwxrwx …odb_log -> \
    ../../../cluster/members/{memb}/opt/OAT100/log_files/odb_log
```

Caution

Do not use the `mkcdsl` command to make CDSLs when you are creating product kits.

2.3.2.4 Restrictions

The following restrictions apply when you use CDSLs in product kits:

- If you are creating a product kit to run on a version of the operating system prior to Version 5.0, you cannot use CDSLs.
- If the system where you are creating a product kit is running a version of the operating system prior to Version 5.0, you can use CDSLs in your product kit, but you must test the kit on a system running Version 5.0 or higher of the operating system.
- If you are creating a product kit on a cluster member running Version 5.0A or higher of the operating system, CDSLs cannot be resolved to the member-specific areas in the `cluster/members/member0` directories. The kernel resolves `{memb}` to the cluster ID of the member where you are creating the kit, `member`*N*.

2.3.3 Placing Files in the Kit Source Directory

Files in the kit source directory hierarchy should match their intended installation location under the `root` (/), `/usr`, and `/var` file systems. Plan to install all of your kit files in product-specific subdirectories in the `/opt`, `/usr/opt`, and `/usr/var/opt` directories to prevent name space conflicts with the base operating system and other layered products.

Table 2–1 shows where to place kit files that will be installed in the root (/), `/usr`, and `/var` file systems:

Table 2–1: File Locations in Kit Directories

Location in Kit src Directory	Installed File System
`opt/`*PRODCODE*	root (/)
`usr/opt/`*PRODCODE*	`/usr`
`usr/var/opt/`*PRODCODE*	`/usr/var`

If you overwrite base operating system files, you may encounter the following problems:

- Your product can be corrupted during an Update Installation of the operating system. The Update Installation will overwrite any file that is on the system with the version of the file shipped with the operating system.
- An Update Installation may not complete successfully if you overwrite a base operating system file. This can make the system unusable.
- Your product may have to be removed from the system to complete an Update Installation. Your product would have to be reinstalled after the Update Installation is completed.
- Removing your product corrupts the operating system.

2.3.4 Setting Up the Sample Product Kit Source Directory

Figure 2–3 shows how the Orpheus Document Builder (ODB) product is installed in the standard directory structure, under `/opt`, `/usr/opt`, and `/usr/var/opt`. The directories shown between the `src` and the `OAT*` directories are the existing directories on the customer's system. All directories and files created by the product are shipped under the `OAT*` directories. In this example, directory names begin with `OAT` because `OAT` is the three-letter product code assigned to *Orpheus Authoring Tools, Inc.*.

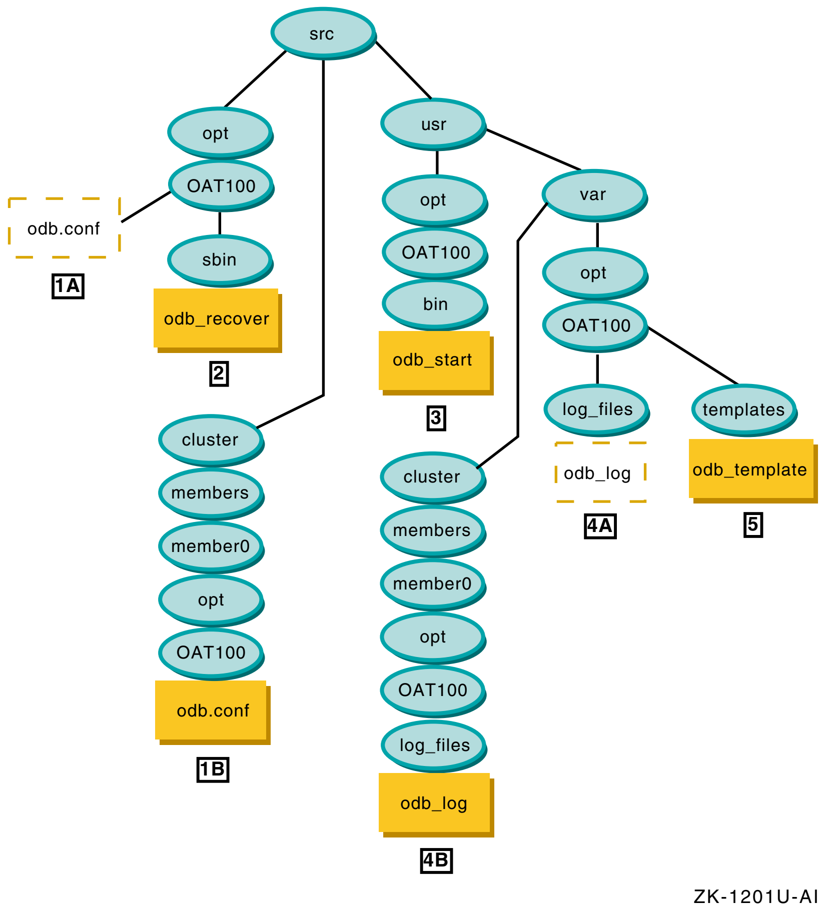
Note

File attributes (ownership and permissions) for files and directories in the kit's source hierarchy must be set exactly as they should be on the customer's system. This means that you must be superuser when populating the source hierarchy so that you can change these file attributes.

Do not attempt to circumvent this requirement by setting file attributes in your subset control programs. If a superuser on the customer's system runs the `fverify` command on your subsets, attributes that have been modified by the subset control programs are reset to their original values in the kit's master inventory files.

Figure 2–3 shows a sample source directory for the OAT product.

Figure 2–3: Sample Product Kit Source Directory



The following files have been provided by the ODB developers for the OAT kit. Each of these files has a path and an associated description, and must be placed correctly in the source hierarchy for the OAT product to build successfully.

$\boxed{1}$ `/odb.conf` — the ODB product configuration file

If the product kit can be installed on a cluster, each cluster member must have its own copy of the configuration file. To accommodate this requirement, create a context-dependent symbolic link (CDSL) targeted to the member-specific file.

A. The context-dependent symbolic link (CDSL) for the `odb.conf` file is linked to the member-specific `cluster/members/{memb}/opt/OAT100/odb.conf` file. This CDSL is installed in the root file system and placed in the `opt/OAT100` source directory.

B. The member-specific file `odb.conf` can differ on each cluster member. This file is installed in the cluster member's root file system and is placed in the `cluster/members/member0/opt/OAT100` source directory.

See Section 2.3.2 for information about CDSLs.

2 `/sbin/odb_recover` — a utility to recover corrupt ODB documents when the system boots. The `odb_recover` script executes when the system boots and the `/usr` file system may not be mounted.

This file is installed in the root file system and is placed in the `opt/OAT100/sbin` source directory.

3 `/usr/bin/odb_start` — the ODB product startup script. The `odb_start` script is a user command.

This file is installed in the `/usr` file system and is placed in the `usr/opt/OAT100/bin` source directory.

4 `/usr/var/log_files/odb_log` — the ODB product log file

If the product kit can be installed on a cluster, each cluster member must have its own copy of the log file. To accommodate this requirement, create a context-dependent symbolic link (CDSL) targeted to a member-specific file.

A. The context-dependent symbolic link (CDSL) for the `odb_log` file is linked to the member-specific `usr/var/cluster/members/{memb}/opt/OAT100/log_files/odb.log` file. This CDSL is installed in the `/usr/var` file system and placed in the `usr/var/opt/OAT100/log_files` source directory.

B. The member-specific file `odb_log` can differ on each cluster member. This file is installed in the cluster member's `/usr/var` file system and is placed in the `/usr/var/cluster/members/member0/opt/OAT100/log_files` source directory.

See Section 2.3.2 for information about CDSLs.

5 `/usr/var/templates/odb_template` — a document template that can be modified by a user.

This file is installed in the `/var` file system and is placed in the `usr/var/opt/OAT100/templates` source directory.

Each of these files will be installed in the path provided by the kit developer. They reside in the same relative location in the kit source directory with `opt/OAT100` inserted into the pathname.

For users to make effective use of your product after it is installed, they should add the directories that contain your product commands to the search path in their `.profile` or `.login` files. For example, the Orpheus Document Builder (ODB) product is installed in the standard directory structure under `/opt`, `/usr/opt` and `/usr/var/opt`. The `src` directory mirrors the `root` (`/`) directory on the customer's system. The commands for the product are located in the `/opt/OAT100/sbin` and `/usr/opt/OAT100/bin` directories. To use ODB commands without specifying the full path on the command line, the user can add these directory names to the `PATH` environment variable.

You can ship a symbolic link to make commands accessible through the standard directories. For example, the ODB kit contains the command `/usr/opt/OAT100/bin/odb_start`. A symbolic link can be created from `/usr/bin/odb_start` to `/usr/opt/OAT100/bin/odb_start`. This also makes the `odb_start` command available to users as a part of their normal search path, since `/usr/bin` is part of the standard path.

You can ship a symbolic link only if one of the following conditions apply:

- The symbolic link does not conflict with any base operating system file.

  Using our example, it means that you could create the `/usr/bin/odb_start` link only if the operating system does not already contain a `/usr/bin/odb_start` file. If the operating system did contain a `/usr/bin/odb_start` file, shipping the symbolic link would overwrite an existing file on the customer's operating system.

- The command name does not conflict with any standard operating system command.

  For example, if the `/usr/opt/OAT100/bin/odb_start` command is shipped in the ODB kit and a command with the same name was part of the standard operating system in `/bin/odb_start`, when a user entered the `odb_start` command there would be a command name conflict. Depending upon whether `/bin` or `/usr/bin` is first in the search path, the user could be accessing the operating system version or the symbolically linked ODB product version.

3
Preparing Subsets

In a product kit, a subset is the smallest installable entity compatible with the `setld` utility. The kit developer specifies how many subsets are included in the kit and what files each subset contains.

As a kit developer, you must perform the following tasks to build subsets and associated control files:

1. Organize product files into subsets. (Section 3.1)
2. Create a master inventory file containing information about each file in the subset. (Section 3.2)
3. Create a key file to define product attributes such as the product name, product version, and subset definitions. (Section 3.3)

3.1 Grouping Files into Subsets

Files that are required for the product to work should be grouped together by function. For example, if the product has two parts such as a user interface and underlying functional code, you should group them into two subsets. A file's physical location is not necessarily a factor in determining the subset to which it belongs.

Optional files also should be grouped together by function, but should be grouped separately from mandatory files. This prevents unnecessary files from being loaded when you install the mandatory subsets.

Note

Subset names are restricted to alphanumeric characters (`A-Z`, `a-z`, `0-9`) and the underscore character (`_`). If you use any other characters in subset names, the `setld` utility may fail.

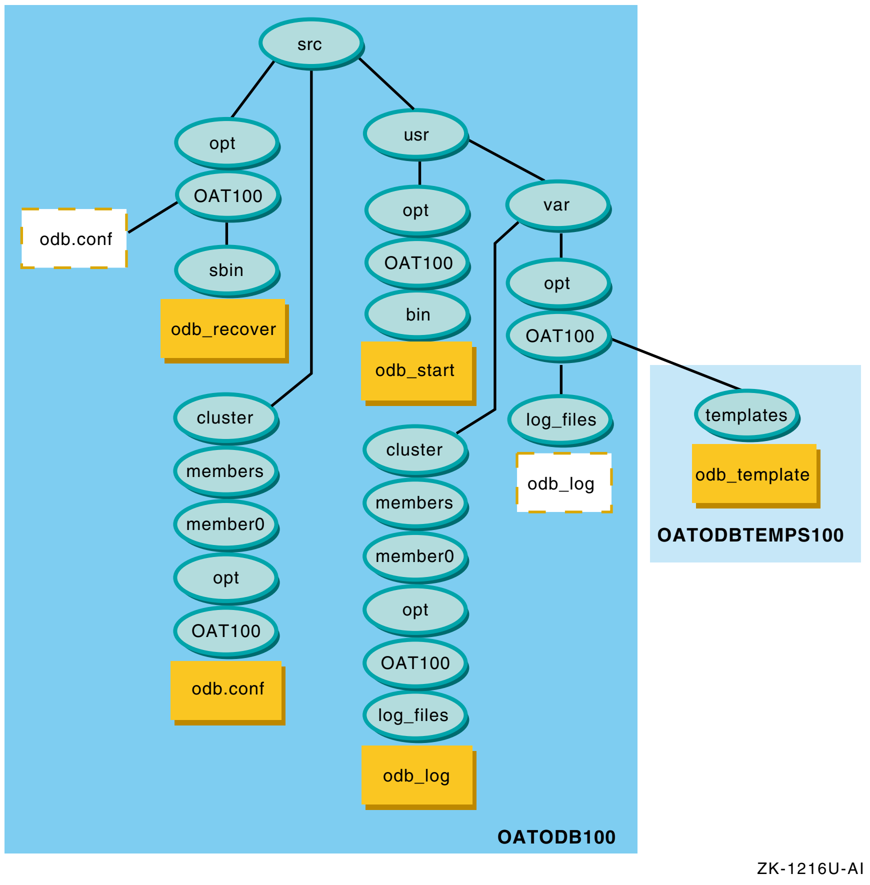
The fictitious ODB user product requires two subsets:

- `OATODB100`, a mandatory subset, contains the files needed to run the product. This includes all product files except the `odb_template` file.
- `OATODBTEMPS100`, an optional subset, contains documentation templates in the `odb_template` file.

By placing the documentation templates in a separate subset, the customer's system administrator can choose not to install them if storage space is limited.

Figure 3–1 shows how the files that make up the ODB product are grouped into subsets. The physical location of a file is not necessarily a factor in determining the subset to which it belongs.

Figure 3–1: ODB Product Subsets and Files



3.2 Creating the Master Inventory File

After choosing subset names and deciding their contents, you have to create a master inventory file to specify the subset names and the files that each subset contains.

Table 3–1 describes the fields in the master inventory file.

Table 3–1: Master Inventory File

Field	Description
Flags	16-bit unsigned integer Bit 1 is the `v` (volatility) bit. When this bit is set, changes to an existing copy of the file can occur either during or after kit installation. The remaining bits are reserved, so valid values for this field are 0 (protected) or 2 (unprotected). The volatility bit usually is set for log files such as `/usr/spool/mqueue/syslog`.
Pathname	The dot-relative (`./`*`path`*) path to the file.
Subset identifier	The name of the subset that contains the file. Subset names consist of the product code, subset mnemonic, and version number. You must not include standard system directories in your subsets. In the ODB master inventory file, several records specify directories that are part of the standard system hierarchy. Instead of a subset identifier, these records specify `RESERVED`; this keyword prevents `setld` from overwriting existing directories.

Follow these steps to create a master inventory file:

1. You can create a master inventory file with your preferred text editor or create the file with the `touch` command. The master inventory file name must consist of the product code and version with the suffix `.mi`. The file should be located in the `data` directory of the kit. For example:

   ```
   % cd /mykit/data
   % touch OAT100.mi
   ```

2. The first time you process a kit, the master inventory file is empty. You must enter one record for each file that belongs on the kit. To get an initial list of these files, use the `newinv` command with the file name of the empty master inventory file and the pathname of the source hierarchy's top-level directory. For example, to invoke `newinv` on the master inventory file for the ODB product, specify the pathname to the source hierarchy as a relative path from the current directory (`data`), similar to the following:

   ```
   % newinv OAT100.mi ../src
   Scanning new baselevel files...done.

   Sorting inventories...done.

   Joining...done.
   ```

```
Awking...done.

        *** THIS BUILD CONTAINS FILES THAT ARE NOT IN THE PREVIOUS BUILD ***

        You will be placed in the editor with the file containing
                the names of these new files.

        If you wish these new files to become part of the product,
                you must convert the line for the wanted files into
                an inventory record.

        Any records remaining in the file when you exit the editor
                will become part of the new inventory.

        Type <RETURN> when you are ready or CTRL/C to quit:
```

The `newinv` utility produces a list of files that are present in the source hierarchy and opens a working copy of the master inventory file in the `vi` editor (or the editor specified by your `$EDITOR` environment variable) to make the required changes.

Caution

The first time that you run the `newinv` utility for a product kit, the following files are created in the `/mykit/data` directory in addition to the `OAT100.mi` file:

```
OAT100.mi.bkp
OAT100.mi.dead
OAT100.mi.extra
OAT100.mi.join
OAT100.mi.tmp
```

Do not modify or delete these additional files. They are used during subsequent master inventory file updates with the `newinv` utility.

3. First, remove the entries for any files that should not appear on the kit. Second, add the flag and subset identifier to the entry for each file that should appear in the kit.

Note

Be extremely careful when you edit the master inventory file. Separate fields in this file with a single Tab character, not spaces. File names must not contain Space or Tab characters.

Use dot-relative pathnames for the files listed in the master inventory file; do not use absolute pathnames. By default, the `setld` utility operates from the system's `root (/)` directory unless you specify an alternate root with the `-D` option.

Example 3–1 shows that the ODB kit has two subsets:

- The `OATODB100` subset contains mandatory commands and utilities
- The `OATODBTEMPS100` subset contains optional document templates

Example 3–1: Sample ODB Kit Master Inventory File

```
0 ./cluster RESERVED [1]
0 ./cluster/members RESERVED [1]
0 ./cluster/members/member0 RESERVED [1]
0 ./cluster/members/member0/opt RESERVED [1]
0 ./cluster/members/member0/opt/OAT100 OATODB100 [2]
2 ./cluster/members/member0/opt/OAT100/odb.conf OATODB100 [2] [3]
0 ./opt RESERVED [1]
0 ./opt/OAT100 OATODB100 [2]
2 ./opt/OAT100/odb.conf OATODB100 [2] [3]
0 ./opt/OAT100/sbin OATODB100 [2]
0 ./opt/OAT100/sbin/odb_recover OATODB100 [2]
0 ./usr RESERVED [1]
0 ./usr/opt RESERVED [1]
0 ./usr/opt/OAT100 OATODB100 [2]
0 ./usr/opt/OAT100/bin OATODB100 [2]
0 ./usr/opt/OAT100/bin/odb_start OATODB100 [2]
0 ./usr/var RESERVED [1]
0 ./usr/var/cluster RESERVED [1]
0 ./usr/var/cluster/members RESERVED [1]
0 ./usr/var/cluster/members/member0 RESERVED [1]
0 ./usr/var/cluster/members/member0/opt  RESERVED [1]
0 ./usr/var/cluster/members/member0/opt/OAT100 OATODB100 [2]
0 ./usr/var/cluster/members/member0/opt/OAT100/log_files OATODB100 [2]
0 ./usr/var/cluster/members/member0/opt/OAT100/log_files/odb_log OATODB100 [2] [4]
0 ./usr/var/opt RESERVED [1]
0 ./usr/var/opt/OAT100 OATODB100 [2]
0 ./usr/var/opt/OAT100/log_files OATODB100 [2]
0 ./usr/var/opt/OAT100/log_files/odb_log OATODB100 [2] [4]
0 ./usr/var/opt/OAT100/templates OATODBTEMPS100 [5]
0 ./usr/var/opt/OAT100/templates/odb_template OATODBTEMPS100 [5]
```

1 Add the `RESERVED` identifier to the directories shown. This tells the `setld` utility not to overwrite the directory if it already exists on the customer's system.

2 Add the `OATODB100` subset identifier to the files and directories shown. This is the mandatory subset for the ODB product.

3 The `odb.conf` file is the ODB product configuration file. The member-specific file resides in the `./cluster/members/member0/opt/OAT100` directory, and the context-dependent symbolic link (CDSL) resides in the `./opt/OAT100` directory. Change the Flags field to `2` to show that this file can change either during or after subset installation.

4 The `odb_log` file is the ODB product log file. The member-specific file resides in the `./usr/var/cluster/members/member0/opt/OAT100/log_files` directory, and the CDSL resides in the `./usr/var/opt/OAT100/log_files` directory.

5 Add the `OATODBTEMPS100` subset identifier to the files and directories shown.

4. After you edit the file list and exit the editor, you see output similar to the following:

```
Merging...done.

Sorting...done.

Master inventory update complete.
```

3.3 Creating the Key File

The key file defines product attributes such as the product name, product version, and subset definitions, as well as the name of the kit's master inventory file. It consists of a product attributes section and a subset descriptor section. The key file name must consist of the product code and version followed by `.k`, so that `OAT100.k` is the key file for the ODB kit. Create the key file in the `data` directory of your kit-building directory structure, for example: `/mykit/data`.

Example 3–2 shows the ODB product kit key file with the two sections separated by two percent signs (`%%`) on their own line:

Example 3–2: Sample ODB Kit Key File

```
#
# Product-level attributes 1
#
NAME='Orpheus Document Builder'
CODE=OAT
VERS=100
MI=/mykit/data/OAT100.mi 2
COMPRESS=1 3
#
# Subset definitions 4
#
%% 5
OATODB100 . 0 'Document Builder Tools' 6
OATODBTEMPS100 OATODB100|OSFDCMT??? 2 'Document Builder Templates' 7
```

1 The product attributes portion of the file describes the naming conventions for the kit and provides kit-level instructions for the `kits` command. This section of the key file consists of several lines of attribute-value pairs as described in Table 3–2. The order of these attribute-value pairs is not significant. Each attribute name is separated from its value by an equal sign, for example: *`attribute=value`*. You can include comment lines, which begin with a pound sign, for example: `#` *`Comment line in key file`*.

2 The value of the `MI` attribute contains the path to the master inventory file. This may be either an absolute path or a relative path from the directory where the `kits` command is executed.

3 The `COMPRESS` attribute has a value of `0` for uncompressed subsets or `1` for compressed `tar` format subsets. If you do not specify this attribute, the default is `COMPRESS=0`. User and kernel product kit subsets may be compressed or uncompressed.

4 The subset descriptor portion of the file describes each of the subsets in the kit and provides subset-level instructions for the `kits` command. This section contains one line for each subset in the kit. Each line consists of four fields, each separated by a single Tab character. Table 3–3 describes the subset descriptor fields.

5 Separate the product-level attributes and the subset definitions with two percent signs (`%%`) on their own line. If this line is not present, the `kits` utility terminates with an error message. You cannot include comments after this line in the key file.

6 In this entry, the Dependency list field value for `OATODB100` is `.` (dot), meaning that the subset has no dependencies.

Set the Flags field to 0 (zero), indicating that the subset is mandatory.

7 In this entry, the `OATODBTEMPS100` subset is optional; its FLAGS field is set to 2 (two). This subset is dependent on both the `OATODB100` subset, part of the `ODB` kit, and the `OSFDCMT???` subset, part of the base operating system. The `???` notation is a wildcard to specify any version of the `OSFDCMT` subset.

Enclose the Subset Description field in single quotes, for example: `'`*`Subset Description`*`'`.

The key file product attributes section describes the naming conventions for the kit and provides kit-level instructions for the `kits` command. This section consists of attribute-value pairs as described in Table 3–2. Each attribute name is separated from its value by an equal sign (`=`). Comment lines in this section begin with a pound sign (`#`).

Table 3–2: Key File Product Attributes

Attribute	Description
`NAME`	The product name; for example, `'Orpheus Document Builder'`. Enclose the product name in single quotation marks (`'`) if it contains spaces.
`CODE`	A unique three-character product code, for example, `OAT`. The first character must be a letter. The first three letters of a subset name must be the same as the product code. In this guide, `OAT` is the three character code assigned to the fictional *Orpheus Authoring Tools, Inc.* company. Several product codes are reserved, including (but not limited to) the following: DNP, DNU, EPI, FOR, LSP, ORT, OSF, SNA, UDT, UDW, UDX, ULC, ULT, ULX, and UWS. Send electronic mail to **`product@dssr.sqp.zko.dec.com`** to request a product code.

Table 3–2: Key File Product Attributes (cont.)

Attribute	Description
`VERS`	A three-digit version code; for example, `100`. The `setld` utility interprets this version code as 1.0.0. The first digit should reflect the product's major release number, the second the minor release number, and the third the upgrade level, if any. The version number cannot be lower than 100. The version number is assigned by the kit developer.
`MI`	The name of the master inventory file. If the master inventory file is not in the same directory where the `kits` utility is run, you must specify the explicit relative path from the directory where you are running the `kits` utility to the directory where the master inventory file resides. The file name of the product's master inventory file consists of the product code and version plus the `.mi` extension. You create and maintain the master inventory file with the `newinv` utility.
`ROOT`	Not shown in the example. The operating system has reserved this optional attribute for the base operating system. `ROOT` has a string value that names the root image file. Do not use this attribute for a layered product.
`COMPRESS`	An optional flag that you set to 1 if you want to create compressed subset files. For kits in Direct CD–ROM (DCD) format, you must set this flag to 0 (zero) and make sure that the FLAGS field in the subset descriptors has bit 2 set to 1 to indicate that the subset is not compressed (see Table 3–3). Compressed files require less space on the distribution media (sometimes as little as 40 percent of the space required by uncompressed files), but they take longer to install than uncompressed files. If missing, this flag defaults to 0 (zero).

The key file subset descriptor section describes each of the subsets in the kit and provides subset-level instructions for the `kits` command. This section contains one line for each subset in the kit and consists of four fields, each separated by a single Tab character. You cannot include comments in this section of the key file. Table 3–3 describes the subset descriptor fields.

Table 3–3: Key File Subset Descriptors

Field	Description
Subset identifier	A character string up to 80 characters in length, composed of the product code (for example, `OAT`), a mnemonic identifying the subset (for example, `ODB`), and the three-digit version code (for example, `100`). In this example, the subset identifier is `OATODB100`. All letters in the subset identifier must be uppercase.
Dependency list	Either a list of subsets upon which this subset is dependent (`OATODB100|OSFDCMT520`), or a single period (`.`) indicating that there are no subset dependencies. Separate multiple subset dependencies with a pipe character (|).
Flags	A 16-bit unsigned integer; the operating system defines the use of the lower 8 bits. See Table 3–4 for a list of values. Set bit 0 to indicate whether the subset can be removed (0=removable, 1=protected). Set bit 1 to indicate whether the subset is optional (0=mandatory, 1=optional). Set bit 2 to indicate compression (0=compressed, 1=uncompressed). Bits 3-7 are reserved for future use. You can use bits 8-15 to relay special subset-related information to your subset control program.
Subset description	A short description of the subset, delimited by single quotation marks (`'`), for example: `'Document Builder Tools'`. The percent sign (`%`) is reserved in this field; do not use it for layered products.

Table 3–4 describes the subset properties indicated by the key file subset descriptor Flags field values. For example, if the Flags field value is 5, the subset is protected, mandatory, and uncompressed.

Table 3–4: Flags Field Values and Subset Properties

	Bit 0		**Bit 1**		**Bit 2**	
	Removable	**Protected**	**Mandatory**	**Optional**	**Compressed**	**Uncompressed**
0	×		×		×	
1		×	×		×	
2	×			×	×	
3		×		×	×	
4	×		×			×
5		×	×			×
6	×			×		×
7		×		×		×

After preparing your subsets as described in this chapter, proceed as follows:

- If you are creating subset control programs, go to Chapter 4.
- If you are not creating subset control programs, go to Chapter 5.

4 Creating Subset Control Programs

A subset control program (SCP) is called by the `setld` utility when installing or deleting your product kit. This chapter describes common tasks required to write subset control programs.

Note

This chapter applies only if you are creating a subset control program. Subset control programs are optional for user product kits and required for kernel product kits. If you are not creating a subset control program, go to Chapter 5.

This chapter discusses the following topics:

- Introducing subset control programs (Section 4.1)
- Creating SCP source files (Section 4.2)
- Setting up initial SCP processing (Section 4.3)
- Working in a cluster environment (Section 4.4)
- Working in a DMS environment (Section 4.5)
- Associating SCP tasks with `setld` processing phases (Section 4.6)
- Determining subset installation status (Section 4.7)
- Stopping the installation (Section 4.8)
- Creating SCPs for different types of product kits (Section 4.9)

4.1 Introducing Subset Control Programs

A subset control program (SCP) performs special tasks beyond the basic installation tasks managed by the `setld` utility. The following list includes some of the reasons why you might write a subset control program:

- Some of your kit's files have to be customized before the product will work properly.
- You want to register a device driver and statically or dynamically configure it.

- You need to establish nonstandard permissions or ownership for certain files.
- Your kit requires changes in system files such as `/etc/passwd`.

A subset control program can perform all of these tasks.

Caution

Code your subset control program so that it can run more than once without causing operational problems. This does not mean that you must repeat the SCP tasks, but that multiple executions will not cause the SCP to fail or to corrupt existing files.

Remove any code from existing SCPs that refers to *`subset-id`*`.lk` or *`subset-id`*`.dw` files.

- Use the `DEPS` field in the subset control file (*`subset_id`*`.ctrl`) for subset dependency processing.

  Do not use the *`subset-id`*`.lk` files to determine subset dependencies.
- Use the library routines described in Table 4–3 to determine if a subset is installed.
- In the current version of the operating system, *`subset-id`*`.dw` files are no longer created. Use the library routines described in Table 4–3 to determine if a subset is corrupt.

See Section 4.7 for more information about determining subset installation status.

4.2 Creating SCP Source Files

Create one subset control program for each subset that requires special handling during installation. You can write the program in any programming language, but your subset control program must be executable on all platforms on which the kit can be installed. If your product works on more than one hardware platform, you cannot write your subset control program in a compiled language. For this reason, it is recommended that you write your subset control program as a script for `/sbin/sh`. All of the examples in this chapter are written in this way.

Keep your subset control programs short. If written as a shell script, a subset control program should be under 100 lines in length. If your subset control program is lengthy, it is likely that you are trying to make up for a deficiency in the architecture or configuration of the product itself.

Note

Subset control programs should not require any interactive responses, and should not generate errors when run repeatedly.

Place all subset control programs that you write in the `scps` directory, a subdirectory of the `data` directory. Each subset control program's file name must match the subset name to which it belongs, and it must end with the `scp` suffix. For example, the ODB product defines two subsets, named `OATODB100` and `OATODBTEMPS100`. If each of these subsets required a subset control program, the source file names would be `OATODB100.scp` and `OATODBTEMPS.scp`.

When you produce the subsets as described in Chapter 5, the `kits` utility copies the subset control programs from the `./data/scps` directory to the `./output/instctrl` directory. If a subset has no SCP, the `kits` utility creates an empty subset control program file for it in the `./output/instctrl` directory.

4.3 Setting Up Initial SCP Processing

Your subset control program should perform the following tasks within the program:

- Including library routines (Section 4.3.1)
- Setting global variables (Section 4.3.2)

The following sections describe the resources available to perform these tasks.

4.3.1 Including Library Routines in Your SCP

The operating system provides a set of routines in the form of Bourne shell script code located in the `/usr/share/lib/shell` directory. Do not copy these routines into your subset control program. This would prevent your kit from receiving the benefit of enhancements or bug fixes made in future releases. Use the following syntax to include these library routines:

```
SHELL_LIB=${SHELL_LIB:-/usr/share/lib/shell}
. $SHELL_LIB/lib_name
```

This specific coding lets you use newer versions of the shell library if they are present in alternate locations.

In the previous example, *`lib_name`* is one of the shell scripts specified in one or more of the following tables.

Table 4–1 lists library routines in the `libscp` shell script.

Table 4–1: SCP Library Routines

Purpose	Library Routine
Architecture determination	`STL_ArchAssert`
Dependency locking	`STL_LockInit`[a]
	`STL_DepLock`[a]
	`STL_DepUnLock`[a]
Dataless environment determination	`STL_IsDataless`
	`STL_NoDataless`
Forward symbolic linking	`STL_LinkCreate`[b]
	`STL_LinkRemove`[b]
Backward symbolic linking	`STL_LinkInit`
	`STL_LinkBack`
SCP initialization	`STL_ScpInit`

[a] Do not use this library routine. It is provided for backward compatibility only. Use the `DEPS` field in the subset control file (*`subset_id`*`.ctrl`) for subset dependency processing.
[b] Do not use this library routine. It is provided for backward compatibility only.

Table 4–2 lists library routines in the `libinstall` shell script.

Table 4–2: Installation Library Routines

Purpose	Library Routine
Cluster member identification	`INST_GetMemberID`

Table 4–3 lists library routines in the `libswdb` shell script.

Table 4–3: Software Database Library Routines

Purpose	Library Routine
Installed subsets	`SWDB_FindInstalledSubsets`
3–digit version number	`SWDB_FindInstalledVersions`
	`SWDB_FindLatestVersions`
Product name in subset control file (*subset_id*.ctrl)	`SWDB_GetProductName`
Subset installation status	`SWDB_CompareStates`[a]
	`SWDB_GetState`[a]
	`SWDB_IsCorrupt`[a]
	`SWDB_IsInstalled`[a]
Subset dependencies	`SWDB_IsLocked`[a]
	`SWDB_ListLockingSubsets`

[a] For more information about this library routine, see Section 4.7.

4.3.2 Setting Global Variables

You can call the `STL_ScpInit` routine to define these variables and initialize them to their values for the current subset. This routine eliminates the need to hard code subset information in your subset control program.

Note

Use the `STL_ScpInit` routine to initialize global variables at the beginning of all `setld` utility processing phases in your SCP except the `M` phase. The control file is not read before the `M` phase.

All predefined global variable names begin with an underscore (_) for easier identification.

Table 4–4 lists global variables that the subset control program can use to access information about the current subset.

Table 4–4: STL_ScpInit Global Variables

Variable	Description
`_SUB`	Subset identifier, for example, `OATODB100`
`_DESC`	Subset description, for example, `Document Builder Tools`
`_PCODE`	Product code, for example, `OAT`
`_VCODE`	Version code, for example, `100`
`_PVCODE`	Concatenation of product code and version code, for example, `OAT100`
`_PROD`	Product description, for example, `Orpheus Document Builder`
`_ROOT`	The root directory of the installation
`_SMDB`	The location of the subset control files, `./usr/.smdb.`
`_INV`	The inventory file, for example, `OATODB100.inv`
`_CTRL`	The subset control file, for example, `OATODB100.ctrl`
`_OPT`	The directory specifier `/opt/`
`_ORGEXT`	File extension for files saved by the `STL_LinkCreate` routine, set to `pre$_PVCODE`[a]
`_OOPS`	The NULL string, for dependency processing[b]

[a] Do not use the `STL_LinkCreate` routine. It is provided for backward compatibility only.
[b] Do not use this global variable. It is provided for backward compatibility only. Use the `DEPS` field in the subset control file (`*subset_id*.ctrl`) for subset dependency processing.

4.4 Working in a Cluster Environment

A cluster is a loosely coupled collection of servers that share storage and other resources that make applications and data highly available. A cluster consists of communications media, member systems, peripheral devices, and applications. The systems within a cluster communicate over a high-performance interconnect.

A TruCluster Server is a highly integrated synthesis of Tru64 UNIX software, AlphaServer™ systems, and storage devices that operate as a single virtual system. The cluster file system (CFS) makes the shared root, `usr`, and `var` file systems visible and available to all cluster members. Cluster members can share resources, data storage, and clusterwide file systems under a single security and management domain, yet they can boot or shut down independently without disrupting the cluster's services to clients.

TruCluster Server includes a cluster alias for the Internet protocol suite (TCP/IP) so that a cluster appears as a single system to its network clients and peers.

For more information about clusters, see the TruCluster Server documentation set.

Note

This manual uses the term current member to mean the cluster member where an operation is taking place. This is not necessarily the cluster member where that operation was invoked.

When you create your subset control programs, consider the following restrictions so that your SCP tasks do not cause operational problems:

- Any `setld` utility phase of your SCP must be able to run more than once without causing operational problems. This does not mean that you must repeat the SCP tasks, but that multiple executions will not cause the SCP to fail or corrupt existing files.
- In a cluster, some `setld` utility phases will run on each cluster member. Any changes made by the SCP must first determine if the change already has been made to that cluster member. If so, the SCP should not attempt to make the change again.
- The SCP should never change the file type of an existing CDSL.

  Do not try to change a CDSL to a file or directory. This would create a shared file where the cluster expects to find a link to a member-specific file and would cause cluster-wide problems.
- The SCP should never change base operating system or cluster inventory.
- The SCP should not decline a delete operation.

  If a deconfiguration fails, report the error and continue the deletion, but do not exit with a nonzero status. The user must fix the problem after the software is removed.

Table 4–5 describes how `setld` utility phases behave when your SCP runs on a cluster.

Table 4–5: SCP Operations on a Cluster

setld Phase	Cluster Behavior
all phases	All phases of `setld` utility processing must be able to run more than once without causing operational problems. This does not mean that you must repeat the SCP tasks, but that multiple executions will not cause the SCP to fail.
`M`	Runs only on the cluster member where you run the `setld` utility, which invokes the SCP on that member.
`PRE_L`	Runs only once for the entire cluster. If you must run an operation on each cluster member, do it in the `C INSTALL` phase.[a]
`POST_L`	Runs only once for the entire cluster. If you must run an operation on each cluster member, do it in the `C INSTALL` phase.[a]
`C INSTALL`	Runs multiple times, once on each cluster member. If your SCP needs to access member-specific files, perform those operations here.
`C DELETE`	Runs multiple times, once on each cluster member. Always return a zero exit status from the `C DELETE` phase. A nonzero status tells the `setld` utility not to delete the software, but if the `setld` utility has run the `C DELETE` phase on other cluster members then the software already may be marked as corrupt. If the operation fails, report the error and continue processing. The user must fix the problem after the software is removed.[b]
`PRE_D`	Runs only once for the entire cluster. If you must run an operation on each cluster member, do it in the `C DELETE` phase.[a] Always return a zero exit status from the `PRE_D` phase. A nonzero status tells the `setld` utility not to delete the software, but since the `setld` utility has run the `C DELETE` phase on other cluster members then the software already may be marked as corrupt. If the operation fails, report the error and continue processing. The user must fix the problem after the software is removed.[b]
`POST_D`	Runs only once for the entire cluster. If you must run an operation on each cluster member, do it in the `C DELETE` phase.[a] Always return a zero exit status from the `POST_D` phase. If the operation fails, report the error and continue processing. The user must fix the problem after the software is removed.[b]
`V`	No cluster-specific restrictions.

[a] There may be circumstances where you must run an operation on each cluster member in this phase. It is possible to include code here that will execute on each cluster member, but it is not recommended as a standard practice.
[b] See the *Installation Guide* and `setld`(8) for recovery information.

4.5 Working in a Dataless Environment

In a Dataless Management Services (DMS) environment, one computer acts as a server by storing the operating system software on its disk. Other computers, called clients, access this software across the Local Area Network (LAN) rather than from their local disks. See *Sharing Software on a Local Area Network* for more information about DMS.

The `setld` utility uses an alternate root directory in a Dataless Management Services (DMS) environment. To make your subset control program DMS compliant, use dot-relative pathnames for file names and full absolute pathnames starting from `root` (/) for commands in your subset control program. This ensures that the proper command is executed when running on either the server or the client in the dataless environment.

The following example shows the default path for SCP processing commands to be run from the server in a DMS environment:

```
/sbin:/usr/lbin:/usr/sbin:/usr/bin:.
```

A subset control program may need to perform differently in a dataless environment or disallow installation of the subset on such a system. If the product will be installed onto a DMS server, use relative pathnames in your SCP. The dataless environment root is the DMS area rather than the DMS server's root file system.

Caution

When running on a dataless client, the `/usr` area is not writable. You cannot install the product kit if any files reside in the `/usr` directory. Make sure the subset control program does not attempt to write to any files located in the `/usr` directory.

You can use the following routines to perform SCP processing in dataless environments:

`STL_IsDataless`

Determines if a subset is being installed into a dataless environment.

`STL_NoDataless`

Declines installation of a subset into a dataless environment.

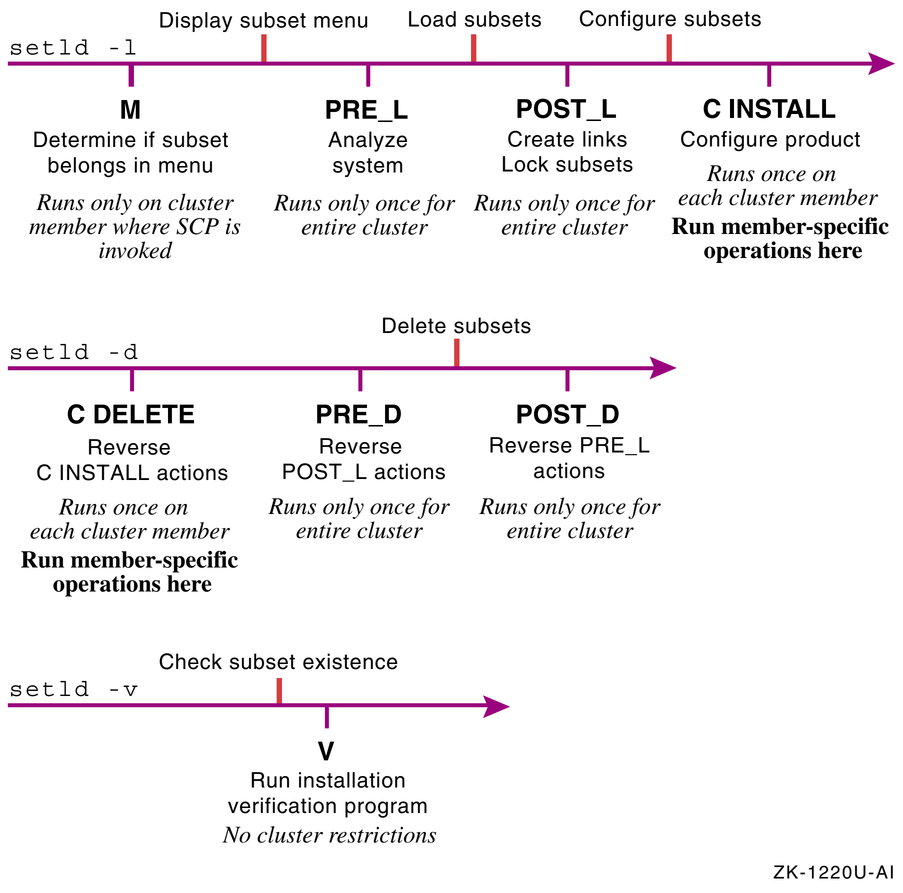
4.6 Associating SCP Tasks with setld Utility Phases

The `setld` utility invokes the subset control program during different phases of its processing. The SCP can perform certain tasks during any of these phases, such as creating or deleting a file or displaying messages. Other tasks that may be required, such as creating links, should be performed only during specific phases.

Some tasks must take place during specific phases. For example, creating links between product files and the standard directory structure occurs during the `POST_L` phase.

Figure 4–1 shows `setld` utility time lines for the `–l`, `–d`, and `–v` options.

Figure 4–1: Time Lines for setld Utility Phases



setld -l
Display subset menu
Load subsets
Configure subsets
M
Determine if subset belongs in menu
Runs only on cluster member where SCP is invoked
PRE_L
Analyze system
Runs only once for entire cluster
POST_L
Create links
Lock subsets
Runs only once for entire cluster
C INSTALL
Configure product
Runs once on each cluster member
Run member-specific operations here
setld -d
Delete subsets
C DELETE
Reverse C INSTALL actions
Runs once on each cluster member
Run member-specific operations here
PRE_D
Reverse POST_L actions
Runs only once for entire cluster
POST_D
Reverse PRE_L actions
Runs only once for entire cluster
setld -v
Check subset existence
V
Run installation verification program
No cluster restrictions
ZK-1220U-AI

The actions taken by the `setld` utility are shown above the time lines. The SCP actions taken during each `setld` processing phase are shown below the time lines, along with any restrictions when the SCP is run on a cluster.

When the `setld` utility enters a new phase, it first sets the `ACT` environment variable to a corresponding value, then invokes the subset control program. The SCP determines the value of the `ACT` environment variable and any command line arguments to determine the required action.

Note

Do not include wildcard characters in your subset control program's option-parsing routine. Write code only for the cases the subset control program actually handles. For example, the subset control programs in this chapter provide no code for several conditions under which they could be invoked, for example, the `V` phase.

The following sections describe the tasks that a subset control program may perform in each `setld` processing phase:

- Displaying the subset menu: `M` phase (Section 4.6.1)
- Before loading the subset: `PRE_L` phase (Section 4.6.2)
- After loading the subset: `POST_L` phase (Section 4.6.3)
- Configuring after securing the subset: `C INSTALL` phase (Section 4.6.4)
- Deconfiguring before deleting a subset: `C DELETE` phase (Section 4.6.5)
- Before deleting a subset: `PRE_D` phase (Section 4.6.6)
- After deleting a subset: `POST_D` phase (Section 4.6.7)
- Verifying the subset: `V` phase (Section 4.6.8)

Caution

Any `setld` utility phase of your SCP must be able to run more than once without causing operational problems. This does not mean that you must repeat the SCP tasks, but that multiple executions will not cause the SCP to fail.

See `setld`(8) and `stl_scp`(4) for more information about the `setld` utility and conventions for subset control programs.

4.6.1 Displaying the Subset Menu (M Phase)

Whenever it performs an operation, the `setld` utility uses the `M` phase to determine if the subset should be included in that operation. Before displaying the menu, `setld` sets the `ACT` environment variable to `M` and calls the subset control program for each subset. At this time, the subset control program can determine whether to include its subset in the menu. The subset control program should return a value of 0 (zero) if the subset can be included in the menu.

Note

In a cluster environment, the `M` phase runs only on the cluster member where the `setld` utility is invoked.

Example 4–1 shows a sample `setld` installation menu, listing the subsets available for installation.

Example 4–1: Sample setld Installation Menu

```
     1) Kit One Name: Subset Description
     2) Kit Two Name: Subset Description
     3) Kit Three Name: Subset Description
     4) Kit Four Name: Subset Description
     5) ALL of the above
     6) CANCEL selections and redisplay menus
     7) EXIT without installing any subsets

Enter your choices or press RETURN to redisplay menus.

Choices (for example, 1 2 4-6):
```

When it calls the subset control program during this phase, the `setld` utility passes one argument, which can have one of two values:

- The `–l` argument indicates that the operation is a subset load.
- The `–x` argument is reserved for extraction of the subset into a RIS server's product area.

When `setld` extracts a subset into a RIS server's product area, the server also executes the subset control program to make use of the program's code for the `M` phase of installation. You should code the `M` phase to detect the difference between extraction of the subset into a RIS area and loading of the subset for use of its contents. To make this determination, determine the value of the `$1` command argument (either `–x` for RIS extraction or `–l` for loading). For RIS extraction, the subset control program should take no action during the `M` phase. When loading subsets, the SCP should perform a `machine` test.

The following Bourne shell example illustrates one way to code the `M` phase. In Example 4–2, the subset control program is determining the type of processor on which it is running. In this example, there is no special code for the RIS extract case.

Example 4–2: Sample Test for Alpha Processor During M Phase

```
#
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
#
# This is the menu phase of the SCP
#
M)
 #
 # Setld invokes the M phase with an argument and if
 # the argument is "-l" it means that a software load
 # is occurring.
 #
 case $1 in
 -l)
  #
  # Examine the machine architecture to be sure
  # that this software is being installed on an
  # alpha machine.  If it is not, exit with an
  # error status so that setld will not display
  # this subset on the menu of subsets to load.
  #
  ARCH=`./bin/machine`
  [ "$ARCH" = alpha ] || exit 1
  ;;
 esac
 ;;
 .
 .
 .
```

In Example 4–2, the SCP returns the following codes to the `setld` utility:

`0` - display the subset in the menu
`1` - do not display the subset in the menu

Note

Installation for a dataless client uses the client's local copy of the `machine` shell script even though the installation is performed in a DMS area on the server. See Section 4.5 and *Sharing Software on a Local Area Network* for more information about DMS.

4.6.2 Before Loading the Subset (PRE_L Phase)

After presenting the menu and before loading the subset, the `setld` utility sets the `ACT` environment variable to `PRE_L` and calls the subset control program. At this time, the subset control program can take any action required to prepare the system for subset installation, such as protecting existing files.

Note

In a cluster environment, the `PRE_L` phase is run only once for the whole cluster. If you must run member-specific operations when your subset is loaded, include the code in the `C INSTALL` phase.

Do not decline software loading because of one cluster member.

Use the `DEPS` field in the subset control file (*`subset_id`*`.ctrl`) for subset dependency processing.

If you overwrite base operating system files, you may encounter the following problems:

- Your product can be corrupted during an Update Installation of the operating system. The Update Installation will overwrite any file that is on the system with the version of the file shipped with the operating system.
- An Update Installation may not complete successfully if you overwrite a base operating system file. This can make the system unusable.
- Your product may have to be removed from the system to complete an Update Installation. Your product would have to be reinstalled after the Update Installation is completed.
- Removing your product corrupts the operating system.

If your subset control program is designed to overwrite existing files, it first should make a backup copy of the original file during the `PRE_L` phase and restore the copy in the `POST_D` phase described in Section 4.6.7.

In Example 4–3, the subset control program examines a list of files to be backed up if they already exist on the system. If it finds any, it creates a backup copy with an extension of .OLD.

Example 4–3: Sample Backup of Existing Files During PRE_L Phase

```
       :
#
# Here is a list of files to back up if found on
# the installed system.
#
BACKUP_FILES="\
 ./usr/var/opt/$_PVCODE/templates/odb_template \ 1
 ./usr/var/opt/$_PVCODE/log_files/odb_log" 1
       :
#
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
       :
#
# This is the pre-load phase of the SCP
#
PRE_L)
 #
 # Loop through the list of backup files and create
 # backup copies for any file that is found on the
 # system.
 #
 for FILE in $BACKUP_FILES
 do
  #
  # If the file to be backed up exists, create
  # a backup copy with a .OLD extension.
  #
  if [ -f $FILE -a ! -f $FILE.OLD ]
  then
   cp $FILE $FILE.OLD 2
  fi
 done
 ;;
       :
```

1 The STL_ScpInit routine sets the value of the $_PVCODE global variable to OAT100. Using this variable allows the SCP to be used for the next version of the product, OAT200, without changing the pathnames.

2 A backup copy is made if the specified file exists and if a backup copy does not already exist. This will be restored in the POST_D phase.

In Example 4–3, the SCP returns the following codes to the setld utility:

0 - load the subset
1 - do not load the subset

4.6.3 After Loading the Subset (POST_L Phase)

After loading the subset, the `setld` utility sets the `ACT` environment variable to `POST_L` and calls the subset control program for each subset. At this time the subset control program can make any modifications required to subset files that usually are protected from modification when the installation is complete. The subset control program should create backward links at this time.

Note

In a cluster environment, the `POST_L` phase is run only once for the whole cluster. If you must run member-specific operations when your subset is loaded, include the code in the `C INSTALL` phase.

Do not decline software configuration because of one cluster member.

Sometimes you may need to create links within your product–specific directories that refer to files in the standard hierarchy. Such backward links must be created carefully because the layered product directories themselves can be symbolic links. This means that you cannot rely on knowing in advance the correct number of directory levels (`../`) to include when you create your backward links. For example, `/var` is frequently a link to `/usr/var`.

When a kit is installed on a Network File System (NFS) server, the SCP should make the backward links in the server's kit area. When the server's kit area is exported to clients, the links are already in place and you do not need to create any backward links in the client area. This is done so that installation on an NFS client cannot overwrite any existing backward links in the server's kit areas. You do not run the subset control program on an NFS client. Your subset control program should create and remove backward links in the `POST_L` and `PRE_D` phases, respectively.

Caution

NFS clients importing products with backward links must have directory hierarchies that exactly match those on the server. Otherwise, the backward links fail.

Use the `STL_LinkInit` and `STL_LinkBack` routines to create backward links as follows, and use the `rm` command to remove them:

`STL_LinkInit`

Used in the `POST_L` phase to establish internal variables for the `STL_LinkBack` routine. Before you use `STL_LinkBack` to create a link, you must execute `STL_LinkInit` once. This routine has no arguments and returns no status.

`STL_LinkBack` *`link_file file_path link_path`*

Creates a valid symbolic link from your product area (under `/usr/opt` or `/usr/var/opt`) to a directory within the standard UNIX directory structure. In this example, *`link_file`* is the file to link, *`file_path`* is the dot-relative path of the directory where the file actually resides, and *`link_path`* is the dot-relative path of the directory where you should place the link. You can use `STL_LinkBack` repeatedly to create as many links as required. This routine returns no status.

The SCP in Example 4–4 uses `STL_LinkBack` in the `POST_L` phase to create a link named `/opt/OAT100/sbin/ls`. This link refers to the real file `/sbin/ls`. The SCP removes the link in the `PRE_D` phase.

Example 4–4: Sample Backward Link Creation During POST_L Phase

```
case $ACT in
        .
        .
        .
#
# This is the post-load phase of the SCP
#
POST_L)
 #
 # Initializes the variables so that STL_LinkBack can be executed
 #
 STL_LinkInit
 #
 # Create a symbolic link in the ./opt/$_PVCODE/sbin
 # directory that points to the ./sbin/ls file.
 #
  STL_LinkBack ls ./sbin ./opt/$_PVCODE/sbin [1]
 ;;
PRE_D)
 #
 # Remove the links created in the POST_L phase
 #
  rm -f ./opt/$_PVCODE/sbin/ls [2]
 ;;
        .
        .
        .
```

1 The `STL_LinkBack` routine creates a backward link in the product-specific area, as described in the comment above the code. The `STL_ScpInit` routine sets the value of the `$_PVCODE` global variable to `OAT100`. Using this variable allows the SCP to be used for the next version of the product, `OAT200`, without changing the pathnames.

2 The SCP uses the `rm` command to remove the links created in the `POST_L` phase.

In Example 4–4, the SCP returns the following codes to the `setld` utility:

`0` - continue subset configuration
`1` - terminate subset configuration; leave the subset corrupt

The `setld` utility creates an empty *`subsetID`*`.lk` lock file when it loads a subset. After successful installation, that subset is then available for dependency processing and locking is performed when other subsets are installed later. A subset's lock file can then contain any number of records, each naming a single dependent subset.

For example, the ODB kit requires that some version of the Orpheus Document Builder base product must be installed for the ODB product to work properly. Suppose that the `OATBASE200` subset is present. When the `setld` utility installs the `OATODBTEMPS100` subset from the ODB kit, it inserts a record that contains the subset identifier `OATODBTEMPS100` into the `OATBASE200.lk` file. When the system administrator uses the `setld` utility to remove the `OATBASE200` subset, the `setld` utility examines `OATBASE200.lk` and finds a record that indicates that `OATODBTEMPS100` depends on `OATBASE200`, displays a warning message, and requires confirmation that the user really intends to remove the `OATBASE200` subset.

If the administrator removes the `OATODBTEMPS100` subset, the `setld` utility removes the corresponding record from the `OATBASE200.lk` file. Thereafter, the administrator can remove the `OATBASE200` subset without causing a dependency warning.

The SCP uses the `DEPS` field in the subset control file (*`subset-id`*`.ctrl`) to perform dependency locking.

4.6.4 After Securing the Subset (C INSTALL Phase)

After securing the subset, the `setld` utility sets the `ACT` environment variable to `C` (configuration) and calls the subset control program for each subset, passing `INSTALL` as an argument. At this time, the subset control program can perform any configuration operations required for product-specific tailoring. For example, a kernel kit can statically or dynamically configure a device driver at this point.

The `setld` utility enters the `C INSTALL` phase when `setld` is invoked with the `-l` (load) option.

Note

In a cluster environment, the `C INSTALL` phase runs multiple times, once on each cluster member. You must be able to run any SCP operations in the `C INSTALL` phase more than once without causing a problem.

If your SCP needs to access member-specific files, perform those operations during the `C INSTALL` phase.

The subset control program cannot create a layered product's symbolic links during the `C INSTALL` phase.

Example 4–5 shows the `C INSTALL` portion of the SCP that issues a message upon successful subset installation.

Example 4–5: Sample Message Output During C INSTALL Phase

```
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
 .
 .
 .
#
# This is the configuration phase of the SCP
#
C)
 #
 # Setld invokes the C phase with an argument that is
 # either INSTALL or DELETE.  The INSTALL argument is
 # used on a setld load, while the DELETE argument is
 # used on a setld delete.
 #
 case $1 in
 INSTALL) [1]
  #
  # Output a message letting the user know
  # that they should check the odb.conf file
  # before using the product.
  #
    echo "
```

Example 4–5: Sample Message Output During C INSTALL Phase (cont.)

```
The installation of the $_DESC ($_SUB) [2]
software subset is complete.
Please check the /opt/$_PVCODE/odb.conf file before [3]
using the $_DESC product." [2]
  ;;
 .
 .
 esac
 ;;
 .
 .
 .
```

[1] During the `C` phase, the SCP determines if the first argument passed by the `setld` utility has the value of `INSTALL`. If so, the program displays a message indicating that the installation is complete.

[2] The `STL_ScpInit` routine sets the value of the `$_DESC` global variable to `Orpheus Document Builder` and the `$_SUB` global variable to `OATODB100`, resulting in the following message:

```
The installation of the Orpheus Document Builder (OATODB100)
software subset is complete.
Please check the /opt/OAT100/odb.conf file before
using the Orpheus Document Builder product."
```

[3] The `STL_ScpInit` routine sets the value of the `$_PVCODE` global variable to `OAT100`. Using this variable allows the SCP to be used for the next version of the product, `OAT200`, without changing the pathnames.

In Example 4–5, the SCP returns the following codes to the `setld` utility:

`0` - successful load and configure
`1` - unsuccessful load and configure; leave the subset corrupt

4.6.5 Before Deleting a Subset (C DELETE Phase)

When the user invokes the `setld` utility with the `–d` option, the utility sets the `ACT` environment variable to `C` and calls the subset control program for each subset, passing `DELETE` as an argument. At this time, the subset control program can make configuration modifications to remove evidence of the subset's existence from the system. For example, a kernel kit would deconfigure a statically or dynamically configured driver during this phase. The `C DELETE` phase should reverse any changes made during the `C INSTALL` phase.

Note

In a cluster environment, the `C DELETE` phase runs multiple times, once on each cluster member. You must be able to run any SCP operations in the `C DELETE` phase more than once without causing a problem.

The SCP always should return a zero exit status in the `C DELETE` phase. A nonzero return status tells the `setld` utility not to delete the software, but if the SCP has run the `C DELETE` phase on other cluster members the software already may be marked as corrupt.

If an operation fails, report the error and continue processing. The user must fix the problem after the software is removed.

The subset control program cannot remove a layered product's links during the `C DELETE` phase.

Example 4–6 shows the `C DELETE` portion of the SCP that would reverse any changes made during the `C INSTALL` phase.

Example 4–6: Sample C DELETE Phase

```
#
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
 .
 .
 .
#
# This is the configuration phase of the SCP
#
C)
 #
 # Setld invokes the C phase with an argument that is
 # either INSTALL or DELETE.  The INSTALL argument is
 # used on a setld load, while the DELETE argument is
 # used on a setld delete.
 #
```

Example 4–6: Sample C DELETE Phase (cont.)

```
 case $1 in
 INSTALL)
  #
  # Output a message letting the user know
  # that they should check the odb.conf file
  # before using the product.
  #
    echo "
The installation of the $_DESC ($_SUB)
software subset is complete.
Please check the  /opt/$_PVCODE/odb.conf file before
using the $_DESC product."
  ;;
   DELETE)
       ;; [1]
   esac
 ;;
 .
 .
 .
```

[1] This phase should reverse any changes made during the `C INSTALL` phase. Since no changes were made in Example 4–5, no action is taken in the `C DELETE` phase.

In Example 4–6, the SCP returns the following codes to the `setld` utility:

`0` - continue with the deletion
`1` - terminate the deletion

4.6.6 Before Deleting a Subset (PRE_D Phase)

When the user invokes the `setld` utility with the `–d` option, the utility sets the `ACT` environment variable to `PRE_D` and calls the subset control program for each subset. At this time, the subset control program can reverse modifications made during the `POST_L` phase of installation, such as removing links.

Note

In a cluster environment, the `PRE_D` phase runs only once for the entire cluster. If you must run member-specific operations when your subset is deleted, include the code in the `C DELETE` phase.

The SCP always should return a zero exit status in the `PRE_D` phase. A nonzero return status tells the `setld` utility not to delete the software, but since the SCP has run the `C DELETE` phase the software is already marked as corrupt.

If an operation fails, report the error and continue processing. The user must fix the problem after the software is removed.

The SCP uses the `DEPS` field in the subset control file (*subset-id*`.ctrl`) to perform dependency unlocking.

In Example 4–4, the SCP used `STL_LinkBack` in the `POST_L` phase to create the `/opt/OAT100/sbin/ls` link, referring to the `/sbin/ls` file. Example 4–7 shows the SCP removing this link in the `PRE_D` phase.

Example 4–7: Sample PRE_D Phase Reversal of POST_L Phase Actions

```
case $ACT in
        .
        .
        .
#
# This is the pre-deletion phase of the SCP
#
PRE_D)
 #
 # Remove the links created in the POST_L phase
 #
  rm -f ./opt/$_PVCODE/sbin/ls [1]
 ;;
        .
        .
        .
```

[1] The SCP uses the `rm` command to remove the links created in the `POST_L` phase.

In Example 4–7, the SCP returns the following codes to the `setld` utility:

`0` - continue with the deletion
`1` - terminate the deletion

4.6.7 After Deleting a Subset (POST_D Phase)

During the `POST_D` phase, after deleting a subset, the `setld` utility sets the `ACT` environment variable to `POST_D` and calls the subset control program for each subset. At this time the subset control program can reverse any modifications made during the `PRE_L` phase of installation.

Note

In a cluster environment, the `POST_D` phase is run only once for the whole cluster. If you must run member-specific operations when your subset is deleted, include the code in the `C DELETE` phase.

The SCP always should return a zero exit status in the `POST_D` phase. A nonzero return status tells the `setld` utility not to delete the software, but the subset already has been removed. This causes cluster corruption.

If an operation fails, report the error and continue processing. The user must fix the problem after the software is removed.

In Example 4–8, the subset control program examines a list of files to be backed up if they already exist on the system. If it finds any, it restores the backup copy.

Example 4–8: Sample File Restoration During POST_D Phase

```
        :
#
# Here is a list of files to back up if found on the installed system.
#

BACKUP_FILES="\
 ./usr/var/opt/$_PVCODE/templates/odb_template \ [1] ./usr/var/opt/$_PVCODE/log_files/odb_log"

#
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
        :
#
# This is the post-deletion phase of the SCP
#

POST_D)

 #
 # Restore the backup copies created during the PRE_L phase
 #

 for FILE in $BACKUP_FILES
```

Example 4–8: Sample File Restoration During POST_D Phase (cont.)

```
 do
   [ -f $FILE.OLD ] &&
   mv $FILE.OLD $FILE [2]
 done
 ;;
        .
        .
        .
esac
```

[1] The `STL_ScpInit` routine sets the value of the `$_PVCODE` global variable to `OAT100`. Using this variable allows the SCP to be used for the next version of the product, `OAT200`, without changing the pathnames.

[2] Restores any files backed up in the `PRE_L` phase, as shown in Example 4–3.

In Example 4–8, the SCP returns the following codes to the `setld` utility:

`0` - deletion was successful; no errors
`1` - errors during deletion, software was removed

4.6.8 Verifying the Subset (V Phase)

When the user invokes the `setld` utility with the `–v` option, the utility sets the `ACT` environment variable to `V` and calls the subset control program for the subset. Any `V` phase processing included in the subset control program is executed at this time.

The `setld` utility determines the existence of the installed subset and if the subset exists, the `setld` utility verifies the size and checksum information for each file in the subset. The `setld` utility does not execute subset control program `V` phase processing during the installation process.

4.7 Determing Subset Installation Status

Subset installation status can drive conditional actions in subset control programs, and accurately determining this status is especially important for recovery processing.

The following topics are discussed in this section:

- How the current version of the operating system defines installation and determines subset status (Section 4.7.1)
- Internal installation states and how they reflect software subset status (Section 4.7.2)
- Using subset installation status library routines in your SCP (Section 4.7.3)

4.7.1 Defining Installation and Subset Status

In the current operating system environment that includes TruCluster Server, software installation status is defined as follows:

- A subset is loaded when its software is copied onto the system
- A subset is installed after it is loaded and its SCP has completed successfully

A software subset can be loaded on one cluster member, corrupt on another, and fully installed and configured on a third. The *subset*`.lk` and *subset*`.dw` files are no longer sufficient to describe software subset installation status.

The `setld` utility generates a r, *subset*`.sts`, for each subset. This file describes the subset's current installation state and reflects the success or failure of specific phases of the installation or deletion process. The `setld` utility removes the subset status file when it deletes the software subset.

Each `/usr/.smdb./`*subset*`.sts` file is a context-dependent symbolic link (CDSL) to the corresponding `/usr/cluster/members/{memb}/.smdb./`*subset*`.sts` file.

The *subset*`.lk` file still exists but now is used only to list dependent subsets.

4.7.2 Understanding Internal Codes for Installation States

Subset installation status corresponds to `setld` utility processing phases as shown in Figure 4–1. Internal states are provided to describe software subset installation status.

Caution

Although subset installation status is reflected in more than one place, your SCP must use these internal states to determine subset installation status. Do not use the contents of the subset status file or the output of the `setld -i` command.

Table 4–6 describes these internal code states:

Table 4–6: Software Subset Installation Status

Internal Code State	Description
`_not_installed`	The software is not on the system. Either the subset was not installed or it was deleted successfully.
`_deleting`	The `setld` utility is in the process of deleting the software. The `setld -d` process has not completed or failed to delete the subset.
`_pre_load_failed`	The `setld` utility failed in the PRE_L phase. Although the software did not load onto the system, some system changes may have been made.
`_pre_load_completed`	The `setld` utility finished the PRE_L phase and the software is ready to load onto the system.
`_verify_failed`	The software is loaded onto the system, but one or more of the files failed the verification process and the software is not installed and cannot be used. The verification process compares the size and checksum of the file on the system against the inventory record. If either value does not match, the file fails verification.
`_verify_completed` [a]	The software is loaded onto the system; all of the subset files are present and verified. The software is not yet installed on the system, as the POST_L and C INSTALL phases must run and protected system files must be moved into place.
`_post_load_failed`	The software is loaded onto the system but the `setld` utility failed in the POST_L phase. The subset files are present but the software is not installed.
`_post_load_completed`	The software is loaded onto the system and the `setld` utility completed the POST_L phase. The subset files are present and the software is ready to be configured in the C INSTALL phase.
`_populate_failed`	The software is loaded onto the cluster, the `setld` utility completed the POST_L phase, but the copying of member-specific files to the current cluster member failed or did not complete.

Table 4–6: Software Subset Installation Status (cont.)

Internal Code State	Description
`_populate_completed`	The software is loaded onto the cluster, the `setld` utility completed the POST_L phase, and all member-specific files were copied to the current cluster member. The software is ready to be configured in the C INSTALL phase on the current cluster member.
`_c_install_failed`	The software is loaded onto the system but the `setld` utility failed in the C INSTALL phase. The subset files are present but the software is not configured or installed.
`_c_install_completed`	The `setld` utility completed the C INSTALL phase and the software is installed on the system. There are no other installation steps required by the `setld` utility, although the software may require additional setup before use.
any other value	The subset status file is corrupted and an unsupported string cannot be mapped to a valid state. This is a serious condition and may require that you remove and reinstall the software.

[a] In previous versions of the operating system, software that reached this point was considered installed as long as the POST_L and C INSTALL phases did not fail, whether or not they actually ran and completed.

Caution

Subset control programs (SCPs) cannot use the *`subset`*`.lk` and *`subset`*`.dw` files to determine if a subset is installed.

The distinction between whether a subset is loaded or installed is important because existing SCPs can fail if they only determine the existence of *`subset`*`.lk` files. For example:

- If your software is dependent upon another subset being installed, confirm that the other subset's installation status is `_c_install_completed`.
- If your software requires that another subset's files are present on the system, you only need to confirm that the other subset's installation status is `_verify_completed`.

See Section 4.7.3 for information about using subset installation status library routines to determine subset installation status.

See `stl_sts`(4) for more information about subset status files.

See `setld`(8) for more information about how the `setld` utility processes and displays software subsets.

4.7.3 Using Subset Installation Status Library Routines

Library routines for determining subset installation status are located in the `/usr/share/lib/shell/libswdb` shell script. These routines are applicable to both single systems and clusters.

Caution

Do not use subset status file values or output from the `setld -i` command to determine software subset installation status. Although these values correspond to the internal code state reflecting subset status, they are not identical and are subject to change. Use the supplied library routines to determine subset installation status in your SCPs.

To use these library routines in your subset control program, you first must source the library shell script as in the following example:

```
SHELL_LIB=${SHELL_LIB:-/usr/share/lib/shell}
. $SHELL_LIB/libswdb
```

The `/usr/share/lib/shell/libswdb` shell script includes the following library routines:

`SWDB_IsInstalled` *`subset`*

Returns `0` if *`subset`* is installed (in the `_c_install_completed` state) on the current member, otherwise returns `1`.

Example:

```
SWDB_IsInstalled OSFBASE520
```

`SWDB_IsLocked` *`subset`*

Returns `0` if *`subset`* is installed and required by another subset, otherwise returns `1`, indicating either that *`subset`* is not installed or that *`subset`* is installed but that no other subset is dependent upon it.

Example:

```
SWDB_IsLocked OSFBASE520
```

`SWDB_IsCorrupt` *`subset`*

Returns `0` if *`subset`* either failed to load or if the SCP failed in one of the `setld` utility phases, otherwise returns `1`.

Example:

```
SWDB_IsCorrupt OSFBASE520
```

`SWDB_GetState` *`subset`*

Returns the internal code state reflecting the installation status of *`subset`*.

Example:

```
STATE=`SWDB_GetState OSFBASE520`
```

This example retrieves the current state of the `OSFBASE520` subset and stores the value in the `STATE` variable. See Table 4–6 for descriptions of subset installation status internal code states.

`SWDB_CompareStates` *`state1 state2`*

Returns `0` if `state1` is before `state2` in the `setld` utility processing sequence, `1` if `state1` is the same as `state2`, or `2` if `state1` is after `state2`.

Use the results of `SWDB_GetState` as one of the arguments to the `SWDB_CompareStates` library routine to compare a subset's internal code state with a fixed code state. This lets you know if the subset status has reached a particular code state and reflects the progress of the subset installation. This routine is useful in determining where to resume processing during a recovery procedure.

Example:

```
STATE=`SWDB_GetState OSFBASE520`
SWDB_CompareStates $STATE _verify_completed
```

This example determines if the `OSFBASE520` subset has reached the `_verify_completed` state on the current member.

The following list shows samples of old SCP code along with new code that uses the new library routines to determine subset installation status.

- In the first case, the SCP must determine if the `OSFBASE520` subset is installed.
  - Old SCP code determines if the `OSFBASE520.lk` file exists and is a regular file. For example:

    ```
    [ -f OSFBASE520.lk ] && {
    … code to execute if software is installed…
    }
    ```

  - New SCP code uses the `SWDB_IsInstalled` routine and looks for a return code of `0`. For example:

    ```
    SWDB_IsInstalled OSFBASE520
    [ "$?" = "0" ] && {
    … code to execute if software is installed…
    }
    ```

- In the second case, the SCP must determine if the `OSFBASE520` subset is corrupt.
  - Old SCP code determines if the `OSFBASE520.dw` file exists. For example:

    ```
    [ -f OSFBASE520.dw ] && {
    … code to execute if software is corrupt…
    }
    ```

  - New SCP code uses the `SWDB_IsCorrupt` routine and looks for a return code of `0`. For example:

    ```
    SWDB_IsCorrupt OSFBASE520
    [ "$?" = "0" ] && {
    … code to execute if software is corrupt…
    }
    ```

- In the third case, the SCP must determine if the software is loaded regardless of whether it is installed.
  - This case did not exist in previous versions of the operating system.
  - New SCP code uses the `SWDB_GetState` and `SWDB_CompareState` routines and looks for a return code that is not zero. For example:

    ```
    STATE=‘SWDB_GetState OSFBASE520‘
    SWDB_CompareStates $STATE _verify_loaded
    [ "$?" != "0" ] && {
    … code to execute if software is loaded …
    }
    ```

4.8 Stopping the Installation

Depending on the tests performed, your subset control program could decide to stop the installation or deletion of its subset. For example, if it finds a later version of the product already installed, the subset control program can stop the process.

To stop the installation or deletion of the subset, the subset control program must return a nonzero status to the `setld` utility when it exits the phase. If the subset control program returns a status of 0 (zero), the `setld` utility continues, assuming that the SCP is satisfied.

4.9 Creating SCPs for Different Product Kit Types

This section provides examples of subset control programs for each of the different product kit types. The following topics are discussed:

- Creating SCPs for user product kits (Section 4.9.1)
- Creating SCPs for kernel product kits (Section 4.9.2)

4.9.1 Creating User Product Kit SCPs

User product kits do not require subset control programs. You may need to provide one if your user product requires special installation tasks.

Example 4–9 shows a subset control program for the ODB user product, illustrating the types of operations that can be performed during different `setld` phases and illustrating one method of using the value of the `ACT` environment variable to determine what actions to perform.

Example 4–9: Sample ODB User Product SCP

```
#!/sbin/sh
#
# Load all of the standard SCP library routines
#
SHELL_LIB=${SHELL_LIB:-/usr/share/lib/shell}
. $SHELL_LIB/libswdb
#
# Initialize the global variables, except in the M phase
#
if [ "$ACT" != "M" ]
then
 STL_ScpInit
fi
#
# Here is a list of files to back up if found on
# the installed system.
#
BACKUP_FILES="\
 ./usr/var/opt/$_PVCODE/templates/odb_template \
 ./usr/var/opt/$_PVCODE/log_files/odb_log"
#
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
#
# This is the menu phase of the SCP
#
M)
 #
 # Setld invokes the M phase with an argument and if
 # the argument is "-l" it means that a software load
 # is occurring.
 #
 case $1 in
 -l)
  #
  # Examine the machine architecture to be sure
```

Example 4–9: Sample ODB User Product SCP (cont.)

```
  # that this software is being installed on an
  # alpha machine.  If it is not, exit with an
  # error status so that setld will not display
  # this subset on the menu of subsets to load.
  #
  ARCH=`./bin/machine`
  [ "$ARCH" = alpha ] || exit 1
  ;;
 esac
 ;;
PRE_L)
 #
 # Loop through the list of backup files and create
 # backup copies for any file that is found on the
 # system.
 #
 for FILE in $BACKUP_FILES
 do
  #
  # If the file to be backed up exists, create
  # a backup copy with a .OLD extension.
  #
  if [ -f $FILE ]
  then
   cp $FILE $FILE.OLD
  fi
 done
 ;;
POST_L)
 #
 # Initializes the variables so that the STL_LinkBack
 # routine can be executed
 #
 STL_LinkInit
 #
 # Create a symbolic link in the ./opt/$_PVCODE/sbin
 # directory that points to the ./sbin/ls file.
 #
 STL_LinkBack ls ./sbin ./opt/$_PVCODE/sbin
 ;;
PRE_D)
 #
 # Remove the links created in the POST_L phase
 #
 rm -f ./opt/$_PVCODE/sbin/ls
 ;;
POST_D)
 #
 # Restore the backup copies created during the PRE_L phase
 #
 for FILE in $BACKUP_FILES
 do
   [ -f $FILE.OLD ] &&
   mv $FILE.OLD $FILE
 done
 ;;
C)
 #
 # Setld invokes the C phase with an argument that is
 # either INSTALL or DELETE.  The INSTALL argument is
 # used on a setld load, while the DELETE argument is
```

Example 4–9: Sample ODB User Product SCP (cont.)

```
 # used on a setld delete.
 #
 case $1 in
 INSTALL)
  #
  # Output a message letting the user know
  # that they should check the odb.conf file
  # before using the product.
  #
  echo "
The installation of the $_DESC ($_SUB)
software subset is complete.

Please check the /opt/$_PVCODE/odb.conf file before
using the $_DESC product."
  ;;
 DELETE)
  ;;
 esac
 ;;
esac
exit 0
```

4.9.2 Creating Kernel Product Kit SCPs

In addition to the optional processing described in Section 4.6, a subset control program for a kernel product such as a device driver also must configure the driver into the kernel. When building subset control programs for a kernel product, you can choose one of the following configuration strategies:

- Write one subset control program for a kit that contains the software subset associated with the single binary module for a statically configured driver.
- Write one subset control program for a kit that contains the software subset associated with the single binary module for a dynamically configured driver.
- Write one subset control program for a kit that contains the software subsets associated with the device driver that can be statically or dynamically configured.

Example 4–10 shows the subset control program for the single binary module associated with the `odb_kernel` driver. The user can choose to configure this single binary module into the kernel either statically or dynamically. The subset control program runs the `doconfig` utility to configure the driver into the kernel.

Example 4–10: Sample ODB Kernel Product SCP

```
#!/sbin/sh
#
# Load all of the standard SCP library routines
#
SHELL_LIB=${SHELL_LIB:-/usr/share/lib/shell}
. $SHELL_LIB/libswdb
# Load the standard Error library routines
# (location of the Error routine)
#
. $SHELL_LIB/Error
#
# Load the standard String library routines
# (location of the ToUpper routine)
#
. $SHELL_LIB/Strings
#
# This routine rebuilds the static kernel
#
Rebuild_Static_Kernel()
{
 HNAME=‘/sbin/hostname -s‘
 HOSTNAME=‘ToUpper $HNAME‘
 if doconfig -c $HOSTNAME
 then
  echo "\nThe /sys/${HOSTNAME}/vmunix kernel has been"
  echo "moved to /vmunix and the changes will take effect"
  echo "the next time the system is rebooted."
  return 0
 else
  Error "
An error occurred while building the static kernel."
  return 1
 fi
}
KERNEL=/cluster/members/{memb}/boot_partition/vmunix
#
# Initialize the global variables, except in M phase
#
if [ "$ACT" != "M" ]
then
 STL_ScpInit
fi
#
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
C)
 #
 # The kreg database file where all of the kernel
 # layered products are registered.
 #
 KREGFILE=./usr/sys/conf/.product.list
 case $1 in
 INSTALL)
  #
  # Merge the graphics support into the existing
  # /etc/sysconfigtab file
  #
  sysconfigdb -m -f ./opt/$_PVCODE/etc/sysconfigtab odb_graphics
  echo "*** $_DESC Product Installation Menu ***\n"
  echo "1. Statically configure the graphics support"
```

Example 4–10: Sample ODB Kernel Product SCP (cont.)

```
 echo "2. Dynamically configure the graphics support"
 echo "\nType the number of your choice []: \c"
 read answer
 case ${answer} in
 1)
  #
  # Determine if the product is already registered
  # with the kreg database, and if it is, skip
  # registering it.
  #
  grep -q $_SUB $KREGFILE
  if [ "$?" != "0" ]
  then
   #
   # Register the product with the
   # kernel using kreg
   #
   /sbin/kreg -l $_PCODE $_SUB \
    ./opt/$_PVCODE/sys/BINARY
  fi
  #
  # Rebuild the static kernel
  #
  Rebuild_Static_Kernel
  #
  # Successful rebuild, so back up the existing
  # kernel and move the new one into place.
  #
  if [ "$?" = "0" ]
  then
   #
   # Make a backup copy of the kernel
   # as it existed prior to installing
   # this subset.  Since a subset can
   # be installed more than once (due to
   # load/configuration failures or even
   # because the user removed files) make
   # sure that the backup does not already
   # exist.
   #
   if [ ! -f $KERNEL.pre_${_SUB)
   then
    mv $KERNEL $KERNEL/pre_${_SUB)
   fi

   #
   # Move the new kernel into place
   #
   mv /sys/${HOSTNAME}/vmunix /vmunix
   #
   # Place a marker on the system so that
   # upon subset removal the SCP can
   # determine if it needs to remove a
   # static or dynamic configuration.
   #
   touch ./opt/$_PVCODE/sys/BINARY/odb_graphics_static
  fi
  ;;
 2)
  #
  # Dynamically load the odb_graphics subsystem
```

Example 4–10: Sample ODB Kernel Product SCP (cont.)

```
   # into the kernel
   #
   sysconfig -c odb_graphics
   ;;
  esac
  ;;
 DELETE)
  #
  # If the marker is present then the kernel option
  # was added statically.
  #
  if [ -f ./opt/$_PVCODE/sys/BINARY/odb_graphics_static ]
  then
   #
   # Clean-up the marker
   #
   rm -f ./opt/$_PVCODE/sys/BINARY/odb_graphics_static
   #
   # Deregister the product using kreg
   #
   /sbin/kreg -d $_SUB
   #
   # Rebuild the static kernel
   #
   Rebuild_Static_Kernel
   #
   # Successful rebuild, remove the old backup
   # copy that was created when we installed.
   #
   if [ "$?" = "0" ]
   then
    mv /sys/${HOSTNAME}/vmunix /vmunix
    rm -f $KERNEL.pre_${_SUB}
   fi
  else
   #
   # Unload the dynamic kernel module
   #
   sysconfig -u odb_graphics
  fi
  #
  # Remove the entry from the /etc/sysconfigtab file
  #
  sysconfigdb -d odb_graphics
  ;;
 esac
 ;;
esac
exit 0
```

5 Producing and Testing Subsets

After preparing your subsets as described in Chapter 3 (and optionally creating subset control programs as described in Chapter 4), perform the following tasks to produce your subsets:

1. Run the `kits` utility to produce subsets and control files (Section 5.1)
2. Test your subsets to make sure that they can be installed, that the product works, and that the kit can be deleted Section 5.2)
3. Optionally, update your kit inventory after producing subsets (Section 5.2)

5.1 Running the kits Utility

After you create the master inventory and key files as described in Chapter 3, run the `kits` utility to produce subsets and control files. The `kits` utility creates the following files:

- The compression flag file (Section 5.1.1)
- The image data file (Section 5.1.2)
- The subset control files (Section 5.1.3)
- The subset inventory files (Section 5.1.4)

Caution

Do not create these files before you run the `kits` utility.

Use the following syntax for the `kits` command:

kits *key-file input-path output-path* [*subset*]...

`key-file`

This mandatory parameter is the pathname of the key file created in Section 3.3.

`input-path`

This mandatory parameter specifies the top of the file hierarchy that contains the source files.

output-path

This mandatory parameter specifies the directory used to store the subset images and data files produced.

subset

This optional parameter specifies the name of an individual subset to be built. You may specify multiple subsets in a space-separated list. If you use the *subset* argument, the `kits` utility assumes the following:

- Only the subsets named as arguments to this parameter are to be built.
- The *key-file* contains descriptors for each of the named subsets.
- All other subsets in the product have been built already.
- The *output-path* directory contains images of the previously built subsets.

If you do not use the *subset* argument, the `kits` utility builds all subsets listed in the key file.

See `kits`(1) for more information.

Note

The master inventory file (`*.mi`) and the key file (`*.k`) are typically in the same directory. If they are not, the `MI=` attribute in the key file must contain the explicit relative path from the directory where you are running the `kits` utility to the directory where the master inventory file resides. The `scps` directory that contains any subset control programs must be in the same directory where the `kits` utility is invoked.

Example 5–1 shows a sample of using the `kits` utility to build the subsets for the ODB product kit:

Example 5–1: Using the kits Utility to Build ODB Subsets

```
# cd /mykit/data
# kits OAT100.k ../src ../output
%%
Creating 2 Orpheus Document Builder subsets.
1   Subset OATODB100 [1]
        Generating media creation information...done
        Creating OATODB100 control file...done.
        Making tar image...done. [2]
        Compressing [3]
           OATODB100: Compression: 92.64%
 -- replaced with OATODB100.Z
```

Example 5–1: Using the kits Utility to Build ODB Subsets (cont.)

```
        *** Finished creating media image for OATODB100. ***

2   Subset OATODBTEMPS100 1
        Generating media creation information...done
        Creating OATODBTEMPS100 control file...done.
        Making tar image...done.
        Compressing 3
           OATODBTEMPS100: Compression: 98.39%
 -- replaced with OATODBTEMPS100.Z
        Null subset control program created for OATODBTEMPS100.

        *** Finished creating media image for OATODBTEMPS100. ***

Creating OAT.image 4

Creating INSTCTRL 5
a OAT.image 1 Blocks
a OAT100.comp 0 Blocks
a OATODB100.ctrl 1 Blocks
a OATODB100.inv 2 Blocks
a OATODB100.scp 7 Blocks
a OATODBTEMPS100.ctrl 1 Blocks
a OATODBTEMPS100.inv 0 Blocks
a OATODBTEMPS100.scp 0 Blocks

Media image production complete.
```

In Example 5–1, the `kits` utility performs the following steps and reports its progress:

1. Creates the subsets.
2. If the subset is not in DCD format, creates a `tar` image of the subset.
3. Compresses each subset if you set the key file's `COMPRESS` attribute to `1`
4. Creates the image data file `OAT.image`.
5. Creates the `INSTCTRL` file, which contains a `tar` image of all the following installation control files:
    - Compression flag file *product-id*`.comp`
    - Image data file *product-code*`.image`
    - Subset control file *subset-id*`.ctrl`
    - Subset inventory file *subset-id*`.inv`
    - Subset control program file *subset-id*`.scp`

      If you created an SCP, the `kits` utility copies it from the kit's `data/scps` directory to the kit's `output/instctrl` directory. If not, the `kits` utility creates an empty *subset-id*`.scp` file in the kit's `output/instctrl` directory.

These files are described in Table 5–1.

The `INSTCTRL` file is placed in the output directory.

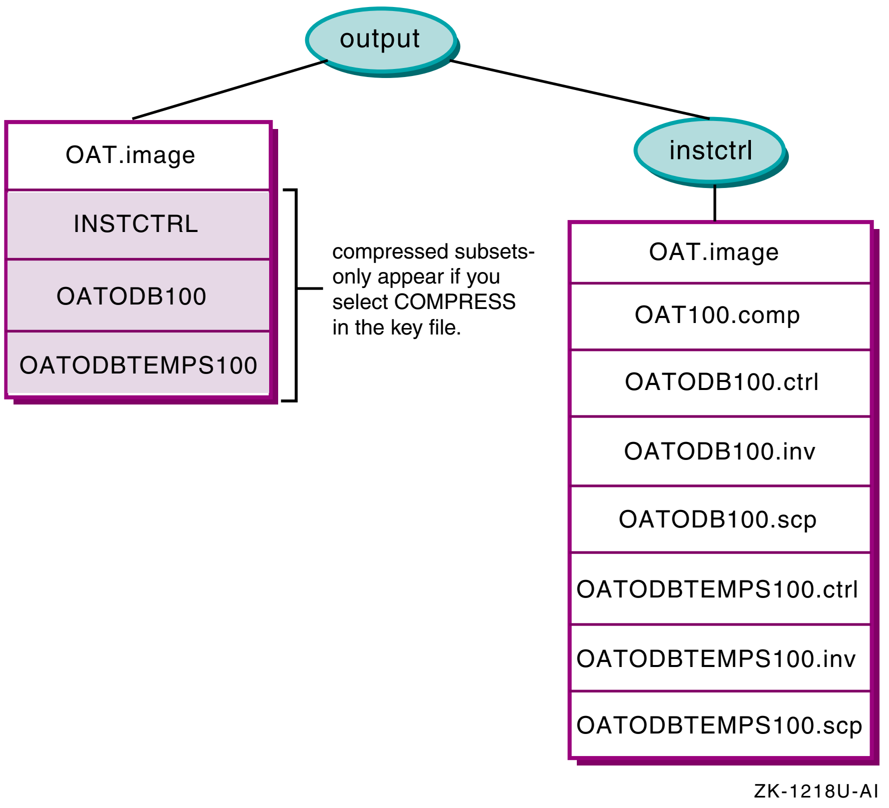
Table 5–1 shows the installation control files in the `instctrl` directory after you run the `kits` utility.

Table 5–1: Installation Control Files in the instctrl Directory

File	Description
product-id.comp	Compression flag file. This empty file is created only if you set the key file's `COMPRESS` attribute to `1`. The ODB kit's compression flag file is named `OAT100.comp`. The contents of the compression flag file are described in Section 5.1.1.
product-code.image	Image data file. This file contains size and checksum information for the subsets. The ODB kit's image data file is named `OAT.image`. The contents of the image data file are described in Section 5.1.2.
subset-id.ctrl	Subset control file. This file contains the `setld` utility control information. There is one subset control file for each subset. The ODB kit's subset control files are named `OATODB100.ctrl` and `OATODBTEMPS100.ctrl`. The contents of the subset control file are described in Section 5.1.3.
subset-id.inv	Subset inventory file. This file contains an inventory of the files in the subset. Each record describes one file. There is one subset inventory file for each subset. The ODB kit's subset inventory files are named `OATODB100.inv` and `OATODBTEMPS100.inv`. The contents of the subset inventory file are described in Section 5.1.4.
subset-id.scp	Subset control program (SCP) . If you created subset control programs for your kit, these files are copied from the `scps` directory to the `instctrl` directory. There is one subset control program for each subset; if you have not created a subset control program for a subset, the `kits` utility creates a blank file. The ODB kit's subset control program files are named `OATODB100.scp` and `OATODBTEMPS100.scp`. Subset control programs are described in Chapter 4.

Figure 5–1 shows the contents of the `output` directory after you run the `kits` utility.

Figure 5–1: ODB output Directory



The subset files and the files in the `instctrl` directory are constituents of the final kit. The following sections describe the contents of the installation control files created by the `kits` utility.

5.1.1 Compression Flag File

The compression flag file is an empty file whose name consists of the product code and the version number with the string `.comp` as a suffix; for example, `OAT100.comp`. The `kits` utility creates a compression flag file if the key file's `COMPRESS` attribute is set to `1`.

5.1.2 Image Data File

The image data file is used by the `setld` utility to verify subset image integrity before starting the actual installation process. The image data file name consists of the product code with the string `.image` as a suffix. The image data file contains one record for each subset in the kit, with three fields in each record.

Table 5–2 describes the image data file.

Table 5–2: Image Data File Field Descriptions

Field	Description
Checksum	The modulo-65536 (16-bit) checksum of the subset file, as provided by the `sum` utility. If the file is compressed, the checksum after compression.[a]
Size	The size of the subset file in kilobytes. If the file is compressed, the size after compression.
Subset identifier	The product code, subset mnemonic, and version number. For example, `OATODB100`.

[a] See `sum`(1) for more information.

Example 5–2 shows the `OAT.image` image data file for the ODB kit:

Example 5–2: Sample Image Data File

```
13601    10 OATODB100
12890    10 OATODBTEMPS100
```

5.1.3 Subset Control Files

The `setld` utility uses the subset control files as a source of descriptive information about subsets. The subset control file's name consists of the subset name followed by the suffix `.ctrl`, for example: `OATODB100.ctrl`.

Note

Do not confuse subset control files with subset control programs (SCPs). Subset control programs are described in Chapter 4.

Subset control file fields are described in Table 5–3.

Table 5–3: Subset Control File Field Descriptions

Field	Description
`NAME`	Specifies the product name. This value is from the Name field in the Key File.
`DESC`	Briefly describes the subset. This value is from the Subset Description field in the Subset Descriptor section of the Key File.
`ROOTSIZE`	Specifies (in bytes) the space the subset requires in the `root (/)` file system.
`USRSIZE`	Specifies (in bytes) the space the subset requires in the `usr` file system. This value is calculated by the `kits` utility.

Table 5–3: Subset Control File Field Descriptions (cont.)

Field	Description
`VARSIZE`	Specifies (in bytes) the space the subset requires in the `var` file system. This value is calculated by the `kits` utility.
`NVOLS`	Specifies disk volume identification information as two colon-separated integers (the volume number of the disk that contains the subset archive and the number of disks required to contain the subset archive). This value is calculated by the `kits` utility.
`MTLOC`	Specifies the tape volume number and subset's location on the tape as two colon-separated integers (the volume number of the tape that contains the subset archive and the file offset at which the subset archive begins). On tape volumes, the first three files are reserved for a bootable operating system image and are not used by the `setld` utility. An offset of 0 (zero) indicates the fourth file on the tape. The fourth file is a `tar` archive named `INSTCTRL`, which contains the kit's installation control files (listed in Table 5–1). This value is calculated by the `kits` utility.
`DEPS`	Specifies either a list of subsets upon which this subset is dependent (`DEPS="OATODB100 OSFDCMT520"`), or a single period (`DEPS="."`) indicating that there are no subset dependencies. If there is more than one subset dependency, each subset name is separated by a Space character. This value is from the Dependency List field in the Subset Descriptor section of the Key file. You may use the following wildcard characters when you specify subset names in the `DEPS` field: • An asterisk (`*`) represents any number of characters. For example, `OAT*100` will match `OAT100`, `OATODB100`, `OATODBTEMPS100`, and so on. • A question mark (`?`) represents a single numeric character. For example, `OATODB1??` matches `OATODB100`, `OATODB101`, and so on up to `OATODB199`.
`FLAGS`	Specifies the value in the flags field of the subsets record in the key file. This value is from the Flags field in the Subset Descriptor section of the Key file. Bit 0 indicates whether the subset can be removed (0=removable, 1=protected). Bit 1 indicates whether the subset is mandatory (0=mandatory, 1=optional). Bit 2 indicates whether the subset is compressed (0=compressed, 1=uncompressed). Bits 3 to 7 are reserved; bits 8 to 15 are undefined.

Example 5–3 shows the `OATODB100.ctrl` subset control file for the ODB kit's `OATODB100` subset:

Example 5–3: Sample Subset Control File

```
NAME='Orpheus Document Builder OATODB100'
DESC='Document Builder Tools'
ROOTSIZE=16668
USRSIZE=16459
VARSIZE=16384
NVOLS=1:0
MTLOC=1:1
DEPS="."
FLAGS=0
```

5.1.4 Subset Inventory File

The subset inventory file describes each file in the subset, listing its size, checksum, permissions, and other information. The subset inventory file's name consists of the subset name followed by the suffix `.inv`, for example: `OATODB100.inv`. The `kits` utility generates this information, reflecting the exact state of the files in the source hierarchy from which the kit was built. The `setld` utility uses the information to duplicate that state, thus transferring an exact copy of the source hierarchy to the customer's system. Table 5–4 describes subset inventory file fields.

Each record of the inventory is composed of 12 fields, each separated by single Tab characters. Table 5–4 describes the contents of these fields.

Table 5–4: Subset Inventory File Field Descriptions

Name	Description
Flags	A 16-bit unsigned integer. Bit 1 is the `v` (volatility) bit. When this bit is set, changes to an existing copy of the file can occur either during or after kit installation. The volatility bit usually is set for log files such as `/usr/spool/mqueue/syslog`. Valid values for this field are 0 (protected) or 2 (unprotected).
Size	The actual number of bytes in the file.
Checksum	The modulo-65536 (16-bit) checksum of the file.
uid	The user ID of the file's owner.
gid	The group ID of the file's owner.
Mode	The six-digit octal representation of the file's mode.
Date	The file's last modification date.
Revision	The version code of the product that includes the file.

Table 5–4: Subset Inventory File Field Descriptions (cont.)

Name	Description
Type	A letter that describes the file: b – Block device. c – Character device. d – Directory containing one or more files. f – Regular file. For regular files with a link count greater than one, see file type l. l – Hard link. Other files in the inventory have the same inode number. The first (in ASCII collating sequence) is listed in the referent field. p – Named pipe (FIFO). s – Symbolic link.
Pathname	The dot-relative (./) pathname of the file.
Referent	For file types l and s, the path to which the file is linked; for types b and c, the major and minor numbers of the device; for all other types, none.
Subset identifier	The name of the subset that contains the file.

Example 5–4 shows the OATODB100.inv inventory file for the ODB kit's OATODB100 subset.

Example 5–4: Sample ODB Product Subset Inventory File

```
0 512 00000 0 0 040755 5/15/00 100 d\
 ./cluster/members/member0/opt/OAT100 none OATODB100
2 44 56771 0 0 100644 5/15/00 100 f\
 ./cluster/members/member0/opt/OAT100/odb.conf none OATODB100
0 512 56771 0 0 040755 5/15/00 100 d\
 ./opt/OAT100 none OATODB100
2 51 00000 0 0 120777 5/15/00 100 s\
 ./opt/OAT100/odb.conf\
 ../../../cluster/members/{memb}/opt/OAT100/odb.conf OATODB100
0 512 00000 0 0 040755 5/15/00 100 d\
 ./opt/OAT100/sbin none OATODB100
0 28 06280 0 0 100644 5/15/00 100 f\
 ./opt/OAT100/sbin/odb_recover none OATODB100
0 512 06280 0 0 040755 5/15/00 100 d\
 ./usr/opt/OAT100 none OATODB100
0 512 06280 0 0 040755 5/15/00 100 d\
 ./usr/opt/OAT100/bin none OATODB100
0 27 33168 0 0 100644 5/15/00 100 f\
 ./usr/opt/OAT100/bin/odb_start none OATODB100
0 512 33168 0 0 040755 5/15/00 100 d\
 ./usr/var/cluster/members/member0/opt/OAT100 none OATODB100
0 23 43390 0 0 100644 5/15/00 100 f\
 ./usr/var/cluster/members/member0/opt/OAT100/odb_log none OATODB100
0 512 43390 0 0 040755 5/15/00 100 d\
 ./usr/var/opt/OAT100 none OATODB100
```

Example 5–4: Sample ODB Product Subset Inventory File (cont.)

```
0 512 43390 0 0 040755 5/15/00 100 d\
 ./usr/var/opt/OAT100/log_files none OATODB100
0 58 00000 0 0 120777 5/15/00 100 s\
 ./usr/var/opt/OAT100/log_files/odb_log\
 ../../../usr/var/cluster/members/{memb}/opt/OAT100/odb_log OATODB100
```

Note

The backslashes (\ ) in Example 5–4 indicate line continuation and are not present in the actual file.

Fields are separated by single Tab characters.

5.2 Testing Subsets

You must test your subsets to ensure that they can be loaded onto a running system, that the product runs on the system, and that the subsets can be deleted. You must perform these tests in the following sequence::

1. Loading all of the subsets onto a running system. (Section 5.2.1)
2. Deleting all of the subsets from a running system. (Section 5.2.2)
3. If your kit includes optional subsets, loading only the mandatory subsets onto a running system. (Section 5.2.3)
4. If your kit can be run in a cluster environment, testing on a cluster. (Section 5.2.4)

See the *Installation Guide* for information about recovering from software load and delete failures.

5.2.1 Loading All Subsets

The examples in this section assume that your kit consists of the mandatory `OATODB100` subset and the optional `OATODBTEMPS100` subset, and that it resides in the `/mykit/output` directory.

Follow these steps to load all subsets:

1. Log in to the system as `root` or use the `su` command to gain superuser privileges.
2. Use the `setld` utility to load all of your subsets onto the system, as in the following example:

   ```
   # setld -l /mykit/output
   ```

3. When prompted, select the option to install all subsets from the `setld` installation menu.

4. Verify that all files in your subsets were loaded. If any files are missing, check the master inventory file. Subset inventory files are created from master inventory file entries.

5. Verify each file's installed location, permissions, owner, and group. The `setld` utility uses the information in the subset inventory file to determine these attributes. If any are incorrect, modify the file in the source directory and rebuild the master inventory file and the subsets.

6. If you supplied SCP files, verify any actions that should have occurred in the `M`, `PRE_L`, `POST_L`, and `C INSTALL` phases. See Section 4.6 for discussions of SCP tasks associated with these phases.

7. After successful installation, test all commands or utilities included with your product. Since file locations may have changed, especially if you installed in the `/opt`, `/usr/opt`, or `/usr/var/opt` directories, it is important that you test your product thoroughly to verify that everything works correctly.

8. Repeat the test to confirm that the SCP does not fail when it runs more than once.

   a. Use the `setld -l` command to reload all of your subsets onto the system.

   b. Verify that all files in your subsets were loaded. If any files are missing, check the master inventory file.

   c. Verify each file's installed location, permissions, owner, and group. If any are incorrect, modify the file in the source directory and rebuild the master inventory file and the subsets.

   d. If you supplied SCP files, verify any actions that should have occurred in the `M`, `PRE_L`, `POST_L`, and `C INSTALL` phases. See Section 4.6 for discussions of SCP tasks associated with these phases.

   e. After successful installation, test all commands or utilities included with your product.

5.2.2 Removing All Subsets

The examples in this section assume that your kit consists of the mandatory `OATODB100` subset and the optional `OATODBTEMPS100` subset, and that it resides in the `/mykit/output` directory.

Follow these steps to remove all subsets:

1. Log in to the system as `root` or use the `su` command to gain superuser privileges.
2. Use the `setld` utility to delete all of your subsets from the system, as in the following example:

   ```
   # setld -d OATODB100 OATODBTEMPS100
   ```
3. Verify that all files loaded onto your system in Section 5.2.1 were deleted.
4. If you supplied SCP files, verify any actions that should have occurred in the `C DELETE`, `PRE_D`, and `POST_D` phases. See Section 4.6 for discussions of SCP tasks associated with these phases.

5.2.3 Loading Mandatory Subsets Only

The examples in this section assume that your kit consists of the mandatory `OATODB100` subset and the optional `OATODBTEMPS100` subset, and that it resides in the `/mykit/output` directory.

Follow these steps to load only the mandatory subsets:

1. Log in to the system as `root`.
2. Use the `setld` utility to load all of your subsets onto the system, as in the following example:

   ```
   # setld -l /mykit/output
   ```
3. When prompted, select the option to install only mandatory subsets from the `setld` installation menu.
4. Verify that all mandatory files in your subsets were loaded. If any files are missing, check the master inventory file. Subset inventory files are created from master inventory file entries.
5. Verify each file's installed location, permissions, owner, and group. The `setld` utility uses the information in the subset inventory file to determine these attributes. If any are incorrect, modify the file in the source directory and rebuild the master inventory file and the subsets.

6. If you supplied SCP files, verify any actions that should have occurred in the `M`, `PRE_L`, `POST_L`, and `C INSTALL` phases. See Section 4.6 for discussions of SCP tasks associated with these phases.
7. After successful installation, test all commands or utilities included with your product. Since file locations may have changed, especially if you installed in the `/opt`, `/usr/opt`, or `/usr/var/opt` directories, it is important that you test your product thoroughly to verify that everything works correctly.

   If your product does not work correctly, some of the files in your optional subsets may need to be moved to mandatory subsets.

5.2.4 Testing in a Cluster

To test your product kit in a cluster, you must ensure that your subsets can be loaded onto a running cluster, that the product runs on the cluster, and that the subsets can be deleted from the cluster. You must perform these tests in the following sequence:

1. Loading all of the subsets onto a cluster. (Section 5.2.4.1)
2. Deleting all of the subsets from a cluster. (Section 5.2.4.2)

The examples in this section assume that your kit consists of the mandatory `OATODB100` subset and the optional `OATODBTEMPS100` subset and that it resides in the `/mykit/output` directory.

5.2.4.1 Loading the Kit onto a Cluster

Follow these steps to load the product kit onto a cluster:

1. Log in to the cluster as `root`.
2. Use the `setld` utility to load all of your subsets onto the cluster, as in the following example:

   ```
   # setld -l /mykit/output
   ```

3. When prompted, select the option to install all subsets from the `setld` installation menu.
4. Verify that all files in your subsets were loaded. If any files are missing, check the master inventory file. Subset inventory files are created from master inventory file entries.
5. Verify each file's installed location, permissions, owner, and group. The `setld` utility uses the information in the subset inventory file to determine these attributes. If any are incorrect, modify the file in the source directory and rebuild the master inventory file and the subsets.

6. Perform the following checks on each cluster member:

   a. Verify each member-specific file's location, permissions, owner, and group. The `setld` utility uses the information in the subset inventory file to determine these attributes. If any are incorrect, modify the file in the source directory and rebuild the master inventory file and the subsets.

   b. Verify that each CDSL can be accessed and that it contains the correct information for each member.

   c. If you supplied SCP files, verify any actions that should have occurred in the `M`, `PRE_L`, `POST_L`, and `C INSTALL` phases. See Section 4.6 for discussions of SCP tasks associated with these phases.

   d. After successful installation, test all commands or utilities included with your product. Since file locations may have changed, especially if you installed in the `/opt`, `/usr/opt`, or `/usr/var/opt` directories, it is important that you test your product thoroughly to verify that everything works correctly.

5.2.4.2 Deleting the Kit from a Cluster

Follow these steps to delete the kit from a cluster:

1. Log in to the cluster as `root` or use the `su` command to gain superuser privileges.

2. Use the `setld` utility to delete all of your subsets from the cluster, as in the following example:

   ```
   # setld -d OATODB100 OATODBTEMPS100
   ```

3. Verify that all files loaded onto your system in Section 5.2.1 were deleted.

4. Perform the following checks on each cluster member:

   a. Verify that each member-specific file was removed.

   b. If you supplied SCP files, verify any actions that should have occurred in the `C DELETE`, `PRE_D`, and `POST_D` phases. See Section 4.6 for discussions of SCP tasks associated with these phases.

   c. Verify that all files not in the inventory are deleted. This includes any files created when your kit was installed.

5.3 Updating Inventory After Creating Subsets

You may have to update the master inventory file after you have created subsets. For example, kernel product kits require additional files, some of which must be added to your kit's inventory. If you create or modify a subset control program after you run the `kits` utility, you also must update the subset's master inventory file.

Run the `newinv` utility to update the master inventory file, using the existing master inventory file as input. The `newinv` utility performs the following additional steps:

1. Creates a backup file, *`inventory-file`*`.bkp`.
2. Finds all the file and directory names in the source hierarchy.
3. Produces the following sorted groups of records:
   - Records that contain pathnames only, representing files now present that were not in the previous inventory (new records).
   - Records that represent files now present that were also present in the previous inventory. This list is empty the first time you create the inventory.
   - Records that were in the previous inventory but are no longer present (defunct records). This list is also empty the first time you create the inventory.
4. Lets you edit the group of defunct records, deleting records for files that no longer belong in the kit.
5. Lets you edit the group of new records by adding the flags and subset identification fields (see Table 3–1).
6. Merges the three groups of records and sorts the result to produce a finished master inventory file that matches the source hierarchy.

Run the `newinv` utility to update the master inventory file any time that you add, modify, or remove files in the kit's source directory. After you update the master inventory file, run the `kits` utility as described in Section 5.1 to produce updated subsets and control files.

6 Producing User Product Kits

This chapter tells you how to produce a user product kit. A user product runs in user space. This includes commands and utilities as well as applications such as text editors and database systems. Users interact directly with user products through commands or window interfaces.

Note

The information in this chapter describes how to produce user product kits. If you want to create a kernel product kit, go to Chapter 7.

Follow these steps to create and test a user product kit:

1. Read Chapter 1 for an overview of product kits.
2. Design the kit directory structure as described in Chapter 2.
3. Prepare subsets and associated control files as described in Chapter 3.
4. Optionally, create subset control programs (Chapter 4). Subset control programs are optional for user product kits.
5. Produce and test subsets as described in Chapter 5.
6. Create the kit distribution media as described in Section 6.2.
7. Test the distribution media as described in Section 6.3.

No additional installation files are required for user product kits.

6.1 Overview

A user product is a layered product that contains software run directly by users. Commands and utilities are in this category, as are applications such as text editors and database systems. Users interact directly with user products through such means as commands or graphical interfaces.

6.2 Producing Distribution Media

After you have tested the subsets as described in Section 5.2, you can produce the distribution media.

Distribution media production consists of the following tasks:

1. Edit the `/etc/kitcap` file. (Section 6.2.1)
2. Build the user product kit on the distribution media:
   - Use the `gendisk` utility to build a kit on disk media (Section 6.2.2)
   - Use the `gentapes` utility to build a `tar` format kit on magnetic tape (Section 6.2.3)

Produce user product kits in `tar` format. You can use direct CD-ROM (DCD) format if you require access to kit files before or during installation, but installation time for DCD format kits is slower than for `tar` format kits.

- `tar` format

  In `tar` format, the product files in each subset are written to the distribution media as a single file. During installation, the `setld` utility uncompresses the files and moves them onto the target system, preserving the files' original directory structure. Kits distributed in `tar` format install more quickly and consume less space on the distribution media.

- direct CD-ROM (DCD) format

  In DCD format, the files are written to the distribution media as a UNIX file system where the product files are organized into a directory structure that mirrors the target system. Subsets distributed in DCD format cannot be compressed.

You can distribute user product kits on diskette, CD-ROM, or magnetic tape, as follows:

- Diskette

  Diskettes are a good media for testing purposes or for small products. The product must fit on a single diskette; it cannot span multiple diskettes. Use the `gendisk` utility to produce kits for diskette media.

- CD-ROM

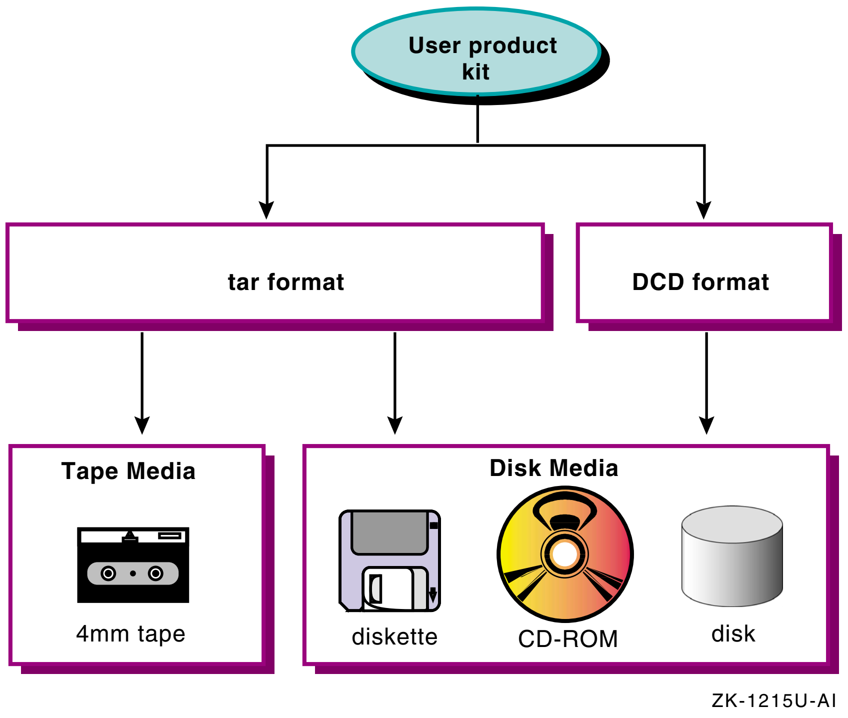
  CD-ROM media can support large kits or multiple kits on a single media. The kit is first produced on the hard disk, then written onto the CD-ROM. Use the `gendisk` utility to produce the master kit on hard disk. Follow the CD-ROM manufacturer's instructions for writing the kit onto the CD-ROM media.

- Magnetic tape

  You can distribute kits for user products on magnetic tape. Tape media does not support DCD format. Use the `gentapes` utility to produce kits for magnetic tape media.

Figure 6–1 shows the types of file formats and distribution media that are available for user product kits.

Figure 6–1: User Product Kit File Formats



6.2.1 Editing the /etc/kitcap File

The `gendisk` and `gentapes` utilities refer to the `/etc/kitcap` file, a database containing information about the kits to be built on the system. Each record contains a product code and the names of the directories, files, and subsets that make up the product kit. Before you can build your kit, you must add a media descriptor record to the `/etc/kitcap` database.

Note

If you use the gendisk utility to produce your kit on disk distribution media, you can specify an alternate kit descriptor database. See gendisk(1) for more information.

Use the following conventions when you add a record to the /etc/kitcap file:

- Separate the first field from the rest of the record by a colon (:) for disk media descriptors and by a pipe character (|) for tape media descriptors.
- Separate all other fields with colons (:).
- Indicate continuation with a backslash (\ ) at the end of the line.
- Lines starting with a pound sign (#) are comments and are ignored.
- Comments within the record start with pound sign (#) and end with a colon (:). Use this feature sparingly.

The contents of a kitcap record differ depending on whether you are producing disk or tape media. You must add one record for each media type on which you plan to distribute your kit.

The contents of the record also can depend on the product type you are delivering. See kitcap(4) for more information about the contents of the /etc/kitcap file.

6.2.1.1 Disk Media Descriptor

Create a disk media kitcap record when you produce kits for distribution on diskette or CD-ROM. The kitcap record for disk media contains the following elements:

- The kit name, consisting of two parts:
  - The product code, consisting of the product code and version number specified in the CODE and VERS fields of the kit's key file. See Section 3.3 for information about the key file.
  - The media code HD to indicate disk media. This element is followed by a colon (:).
- The partition on the disk media where the product should be placed. The partition is a letter between a and h. Partition c is used most often, as it spans the entire disk.
- The destination directory for the subsets on the disk media. This allows a hierarchical structure so you can put multiple products on one disk, or put parts of one product on different areas of the same disk. You can use multiple destination directories in a kitcap record.

- The product description. This entry is taken from key file NAME field. Replace any spaces with an underscore (_) character, for example: Product Description becomes Product_Description.
- The name of the output directory where you created the kit, where the gendisk utility can find the product subsets.
- The instctrl directory, relative to the output directory specification.
- The names of the subsets that make up the kit.

See kitcap(4) for more detailed information about the disk media record format.

See Section 3.3 for information about the key file.

Example 6–1 shows the record to be added to the /etc/kitcap file to produce the ODB kit on disk media:

Example 6–1: Sample Disk Media Descriptor for User Product

```
OAT100HD:c:/:\
  dd=/OAT100:Orpheus_Document_Builder:/mykit/output:\
  instctrl:OATODB100:OATODBTEMPS100
```

Based on the information shown in Example 6–1, the gendisk utility places the kit on the c partition in the / (root) directory of the disk media. The product description is Orpheus_Document_Builder and the output directory where you created the kit is /mykit/output. The kit consists of two subsets: OATODB100 and OATODBTEMPS100.

6.2.1.2 Tape Media Descriptor

The kitcap record for tape media contains the following elements:

- The kit name, consisting of two parts:
  - Product code, consisting of the product code and version number specified in the CODE and VERS fields of the kit's key file. See Section 3.3 for information about the key file.
  - The media code, either TK for TK50 tapes or MT for 9-track magnetic tape. This element is followed by a pipe character (|).
- Product description. This entry is taken from the NAME field of the key file.

- Name of the output directory where you created the kit, where the `gentapes` utility can find the subsets.

  Since the `gentapes` utility can take subsets from multiple products and merge them on tape as a combined product, you can specify multiple directories where the `gentapes` utility can find the subsets. There must be one directory entry for each `kitcap` descriptor.

- Three empty `SPACE` files to ensure compatibility with operating system kits. To create the `SPACE` file in the output area of the kit directory structure, use comments similar to the following:

  ```
  # cd /mykit/output
  # touch space
  # tar -cf SPACE space
  ```

- The `INSTCTRL` image in the `output` directory containing `setld` control information.
- The names of the subsets that make up the kit. Each subset listed must be stored in one of the specified directories.
- Optional volume identifiers `%%`*N*, followed by the names of the subsets to be placed on that volume. You can use multiple tapes.

See `kitcap`(4) for more detailed information about the tape media record format.

Example 6–2 shows the record to be added to the `/etc/kitcap` file to produce the ODB kit on TK50 tapes:

Example 6–2: Sample Tape Media Descriptor for User Product

```
OAT100TK|Orpheus Document Builder: \
    /mykit/output:SPACE:SPACE:SPACE: \
    INSTCTRL:OATODB100:OATODBTEMPS100
```

The product name, `OAT100`, is the same name that appears in the key file. The product description, `Orpheus Document Builder`, also appears in the key file. The name of the output directory is `/mykit/output`, and three `SPACE` files are included for compatibility with operating system kits. The last line of the record contains the `INSTCTRL` image in the output directory and the names of the subsets that make up the kit: `OATODB100` and `OATODBTEMPS100`.

6.2.2 Building a User Product Kit on Disk Media

After the product subsets are located in the output area of the kit directory structure, use the `gendisk` utility to produce the kit on a disk.

Note

The `gendisk` utility supports diskettes but does not let you create a chained diskette kit. A kit written to diskette must fit on a single diskette or be packaged as a set of kits on separate diskettes.

Use the following syntax for the `gendisk` command:

gendisk [-d] [-i] [-k *filename*] [-w] [-v] [*hostname*:] *prodID* *devname*

Note

If you do not use either the `-w` or `-v` options, the `gendisk` utility writes and then verifies the product media.

`-d`

Creates a distribution disk in direct CD format. This means that the distribution disk contains uncompressed file systems that are laid out just as the software is installed on the system.

Note

We recommend that you do not use the `-d` option when you use the `gendisk` utility to produce user product kits.

`-i`

Creates a distribution disk in ISO 9660 format. This means that the distribution disk contains an ISO 9660-compliant CD-ROM file system (CDFS).

`-k` *`filename`*

Uses an alternate kit descriptor database, *`filename`*, on the local system. You may use either a full absolute pathname or a relative pathname from the directory where you run the `gendisk` utility. The file does not have to be named `kitcap`.

`-w`

Writes the product media without verification, if used without the `-v` option. If used with the `-v` option, the `gendisk` utility writes and then verifies the product media.

`-v`

Verifies the product media without writing it first, if used without the `-w` option. This assumes that you already have written kit files to the distribution media. If used with the `-w` option, the `gendisk` utility writes and then verifies the product media.

`hostname``:`

The optional *`hostname`*`:` operand is the name of a remote machine that contains the kit descriptor database. The `gendisk` utility searches the kit descriptor database on the remote machine for the kit identifier (*`prodID`*`HD`) and uses it to create the distribution media. The colon (`:`) is a required delimiter for TCP/IP networks, and space is permitted between the colon and the *`prodID`*. For example, if the product code is OAT100 and you are using the kit descriptor database on node `mynode`, use `mynode:OAT100` for this option.

`prodID`

The mandatory *`prodID`* operand is a kit identifier consisting of the product code and version number specified in the `CODE` and `VERS` fields of the kit's key file. See Section 3.3 for information about the key file.

`devname`

The mandatory *`devname`* operand specifies the device special file name for a raw or character disk device such as `/dev/rdisk/dsk1`. The `gendisk` utility uses the disk partition specified in the kit descriptor and ignores any partition specified on the command line.

The command shown in Example 6–3 creates a `tar` format user product kit for `OAT100` on `dsk0`:

Example 6–3: Sample gendisk Command for User Product

```
# gendisk OAT100 /dev/rdisk/dsk0
```

See `gendisk(1)` for more information about this utility.

6.2.3 Building a User Product Kit on Magnetic Tape

After the product subsets are located in the output area of the kit directory structure, use the `gentapes` utility to build the kit on magnetic tape.

Use the following syntax for the `gentapes` command:

/usr/bin/gentapes [-w | -v] [*hostname*:] *prodID* *devname*

`-w`

Writes the product media without verification. Do not use the `-w` option with the `-v` option.

`-v`

Verifies the product media without writing it first. Do not use the `-v` option with the `-w` option.

`hostname:`

The optional *`hostname:`* argument is the name of a remote network machine that contains the kit descriptor database. The `gentapes` utility searches the kit descriptor database on the remote machine for the kit identifier (*`prodID`*`[TK|MT]`) and uses it to create the media. The colon (`:`) is a required delimiter for TCP/IP networks, and space is permitted between the colon and the *`prodID`*. For example, if the product code is OAT100, and the kitcap file to be used is on node `mynode`, use `mynode:OAT100` for this option.

`prodID`

The mandatory *`prodID`* operand is a kit identifier consisting of the product code and version number specified in the `CODE` and `VERS` fields of the kit's key file. See Section 3.3 for information about the key file.

`devname`

The mandatory *`devname`* operand specifies the device special file name for a no-rewind tape device such as `/dev/ntape/tape01`. The `gentapes` utility uses the default tape density for the device and ignores any suffix specified on the command line.

Note

If you do not use either the `-w` or `-v` option, the `gentapes` utility writes the tape, rewinds it, and then verifies the files in the kit descriptor.

The command shown in Example 6–4 creates a `tar` format user product kit for `OAT100` on the magnetic tape in `/dev/ntape/dat`:

Example 6–4: Sample gentapes Command for User Product

```
# gentapes OAT100 /dev/ntape/dat
```

See `gentapes`(1) for more information about this utility.

6.3 Testing the Distribution Media

Before shipping a user product kit to customers, you should test the kit with the same procedures that your customers will use on configurations that resemble your customers' systems.

Use the `setld` utility to test a user product kit as described in the following procedure for the `OAT100` kit:

1. Log in to the system as `root` or use the `su` command to gain superuser privileges.
2. Place the CD–ROM in the drive.
3. Create a directory to be the media mount point, such as `/cdrom`:

   ```
   # mkdir /cdrom
   ```
4. Mount the CD–ROM on `/cdrom`. For example, if the CD–ROM device is located on the `c` partition of `cdrom0`, enter the following command:

   ```
   # mount -r /dev/disk/cdrom0c /cdrom
   ```

   After mounting the CD–ROM, you can change to the `/cdrom` directory and view the directories on the CD–ROM.
5. Install the user product subsets:

   ```
   # setld -l /cdrom/OAT100/kit

   *** Enter subset selections ***

   The following subsets are mandatory and will be installed automatically
   unless you choose to exit without installing any subsets:

         * Document Builder Tools

   The subsets listed below are optional:

    - Other:
        1) Document Builder Templates
   ```

```
Or you may choose one of the following options:

     2) ALL mandatory and all optional subsets
     3) MANDATORY subsets only
     4) CANCEL selections and redisplay menus
     5) EXIT without installing any subsets

Estimated free diskspace(MB) in root:54.5 usr:347.0

Enter your choices or press RETURN to redisplay menus.

Choices (for example, 1 2 4-6): 2

You are installing the following mandatory subsets:

        Document Builder Kernel Support
        Document Builder Tools

You are installing the following optional subsets:

 - Other:
        Document Builder Templates

Estimated free diskspace(MB) in root:54.5 usr:347.0

Is this correct? (y/n): y

Checking file system space required to install selected subsets:

File system space checked OK.

3 subset(s) will be installed.

Loading subset 1 of 2 ...

Document Builder Tools
   Copying from /mykit/output (disk)
   Verifying

Loading subset 2 of 2 ...

Document Builder Templates
   Copying from /mykit/output (disk)
   Verifying

2 of 2 subset(s) installed successfully.

Configuring "Document Builder Tools" (OATODB100)

The installation of the Document Builder Tools (OATODB100)
software subset is complete.

Please check the /opt/OAT100/odb.conf file before
using the Document Builder Tools product.

Configuring "Document Builder Templates" (OATODBTEMPS100)

#
```

The `setld` utility displays prompts and messages to guide you through the process of selecting the subsets you want to install. As each subset is loaded, the `setld` utility calls the subset control program as needed.

6. After the installation finishes, unmount the CD–ROM:

   ```
   # umount /cdrom
   ```

7. Verify that the installed product functions correctly.

See the *Installation Guide* and `setld`(8) for more information about using the `setld` utility to install layered products.

7 Producing Kernel Product Kits

This chapter tells you how to produce a kernel product kit, what additional files are required, and how to test the kit installation.

Note

The information in this chapter describes how to produce kernel product kits. If you want to create a user product kit, go to Chapter 6.

Follow these steps to create and test a kernel product kit:

1. Read Chapter 1 for an overview of product kits.
2. Design the kit directory structure as described in Chapter 2.
3. Prepare subsets and associated control files as described in Chapter 3.
4. Create subset control programs (Chapter 4). Subset control programs are required for kernel product kits.
5. Produce and test subsets as described in Chapter 5.
6. Create additional installation files as described in Section 7.2.
7. Rerun the `newinv` utility as described in Section 5.3 to update the master inventory file with the additional installation files.
8. Rerun the `kits` utility as described in Chapter 5 to produce updated subsets and control files.
9. Create the kit distribution media as described in Section 7.3.
10. Test the distribution media as described in Section 7.4.

7.1 Overview

A kernel product is a layered product that contains kernel support. Users do not run kernel products directly; the operating system and utilities access kernel products to perform their work. For example, a device driver is one common type of kernel product. A user runs an application or utility, which generates system requests to perform operations such as opening a file or writing data to a disk. The system determines which device driver should service this request and then calls the required driver interface.

The kernel modules and the kit support files are distributed as part of the kernel product kit, and can be installed either directly from the distribution media or loaded onto a Remote Installation Services (RIS) area for installation by RIS clients over a local area network (LAN).

Caution

The following restrictions apply to kernel product kits that support hardware provided by outside manufacturers, commonly referred to as third party hardware support:

- Third party hardware devices are not supported in clusters environments.
- Third party storage devices are supported as data devices only. Boot support is not available for third party storage devices.
- Third party graphics devices can be supported, but the device reverts to VGA mode after an Update Installation if that device's support is not included in the update package. Reinstalling the device's kernel kit after the Update Installation can restore full graphics support.
- Third party platforms can be supported, but only for Full Installation.

  If the system kernel is destroyed, you cannot restore it from `genvmunix`. You would have to do a Full Installation of the operating system and reinstall the third party platform support.

  With the number of restrictions on third party platform support, Compaq recommends that all third party platform support be delivered by the Compaq new hardware delivery (NHD) process. For more information, please send electronic mail to **`upmbosstaff@compaq.com`**.

7.2 Creating Additional Installation Files

The files needed to build a kernel product kit depend on whether the kit will be configured statically or dynamically on the customer's system. For example:

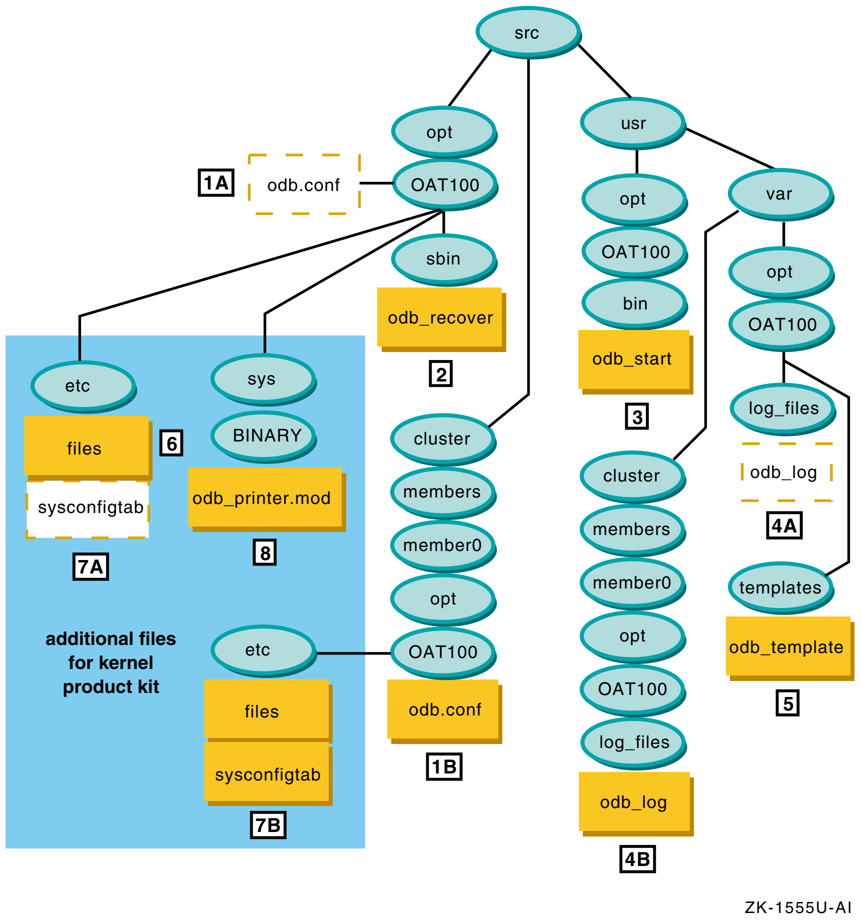
- A statically configured product is linked statically into the kernel at build (or bootlink) time and configured at boot time. A static product can be built from source files, binary objects, modules, or all three.
- A dynamically configured product is loaded into a running kernel after it has been booted. It is not part of the permanent kernel and is configured when it is loaded. It must be reloaded after each boot of the system. A dynamic product can be built from source files, binary objects, modules, or all three.

Note:

A module that can be loaded dynamically also can be linked statically. The only difference is the call to configure the product. For more information on static and dynamic drivers, see the *Writing Device Drivers* manual.

Figure 7–1 shows the files that make up the ODB product kit source directory. Additional kernel product files are highlighted.

Figure 7–1: Kernel Product Source Directory



Note

The files locations in the following list describe where you would find the files when you receive the product from the kit developer. Figure 7–1 shows where these files are located in the actual kit source hierarchy.

1 `/odb.conf` — the ODB product configuration file.

If your product kit may run on a cluster, each cluster member must have its own configuration file. To accommodate this requirement, create a context-dependent symbolic link (CDSL) targeted to the member-specific file.

A. The context-dependent symbolic link (CDSL) for the `odb.conf` file is linked to the member-specific `cluster/members/{memb}/opt/OAT100/odb.conf` file. This CDSL is installed in the root file system and placed in the `opt/OAT100` source directory.

B. The member-specific file `odb.conf` file can differ on each cluster member. This file is installed in the cluster member's root file system and is placed in the `cluster/members/member0/opt/OAT100` source directory.

2 `/sbin/odb_recover` — a utility to recover corrupt ODB documents when the system boots.

The `odb_recover` script executes when the system boots and the `/usr` file system may not be mounted. This file is installed in the root file system and is placed in the `opt/OAT100/sbin` source directory.

3 `/usr/bin/odb_start` — the ODB product startup script.

The `odb_start` script is a user command. This file is installed in the `/usr` file system and is placed in the `usr/opt/OAT100/bin` source directory.

4 `/usr/var/log_files/odb_log` — the ODB product log file.

If the product kit can be installed on a cluster, each cluster member must have its own copy of the log file. To accommodate this requirement, create a context-dependent symbolic link (CDSL) targeted to a member-specific file.

A. The context-dependent symbolic link (CDSL) for the `odb_log` file is linked to the member-specific `usr/var/cluster/members/{memb}/opt/OAT100/log_files/odb_log` file. This CDSL is installed in the `/usr/var` file system and placed in the `usr/var/opt/OAT100/log_files` source directory.

B. The member-specific file `odb_log` file can differ on each cluster member. This file is installed in the cluster member's `/usr/var` file system and is placed in the `/usr/var/cluster/members/member0/opt/OAT100/log_files` source directory.

See Section 2.3.2 for information about CDSLs.

[5] `/usr/var/templates/odb_template` — a document template that can be modified by a user.

This file is installed in the `/var` file system and is placed in the `usr/var/opt/OAT100/templates` source directory.

[6] `/etc/files` — the `files` file fragment contains information about the location of the source code and modules associated with the driver, tags indicating when the driver is loaded into the kernel, and whether the source or binary form of the driver is supplied to the customer.

Note

This is not a complete `/etc/files` file.

[7] `/etc/sysconfigtab` — the `sysconfigtab` file fragment contains device special file information, bus option data, and information on contiguous memory usage for statically and dynamically configured drivers.

Note

This is not a complete `/etc/sysconfigtab` file.

The `sysconfigtab` file fragment gets appended to the `/etc/sysconfigtab` database when the kernel product kit is installed. See `sysconfigtab`(4) for more information.

If the product kit can be installed on a cluster, each cluster member must have its own copy of the `sysconfigtab` file fragment. To accommodate this requirement, create a context-dependent symbolic link (CDSL) targeted to the member-specific file.

A. The context-dependent symbolic link (CDSL) for the `sysconfigtab` file fragment is linked to the member-specific `cluster/members/{memb}/opt/OAT100/etc/sysconfigtab` file. This CDSL is installed in the root file system and placed in the `opt/OAT100/etc` source directory.

B. The member-specific `sysconfigtab` file can differ on each cluster member. This file is installed in the cluster member's root file system and is placed in the `cluster/members/member0/opt/OAT100/etc` source directory.

See Section 2.3.2 for information about CDSLs.

8 `/sys/BINARY/odb_printer.mod` — the object module file containing the single binary module for the ODB kernel product.

A kernel product kit requires you to include the following files on the distribution media to make the kernel product accessible during installation:

- The `files` file fragment (Section 7.2.1)
- The `sysconfigtab` file fragment (Section 7.2.2)
- The *`driver`*`.mod` object module file (Section 7.2.3)
- The `*.c` (source) and `*.h` (header) files (Section 7.2.4)
- The *`device`*`.mth` method files (Section 7.2.5)

7.2.1 The files File Fragment

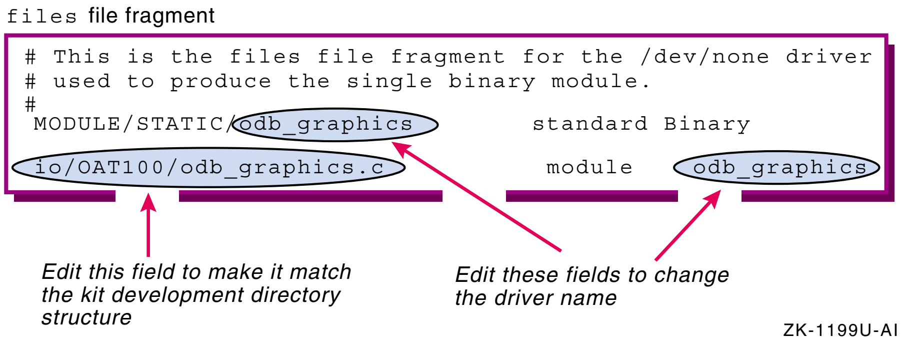
The `files` file fragment contains information about the location of the source code and modules associated with the driver, tags indicating when the driver is loaded into the kernel, and whether the source or binary form of the driver is supplied to the customer. You need to edit this file if the kit development directory structure differs from the driver development directory structure or if you must change the driver name for any reason.

Caution

The `files` file fragment must be in the same directory as the kernel modules or the `kreg` and `doconfig` utilities will not work properly.

Figure 7–2 shows which fields within the `files` file fragment need to change.

Figure 7–2: Editing the files File Fragment


files file fragment
Edit this field to make it match the kit development directory structure
Edit these fields to change the driver name
ZK-1199U-AI

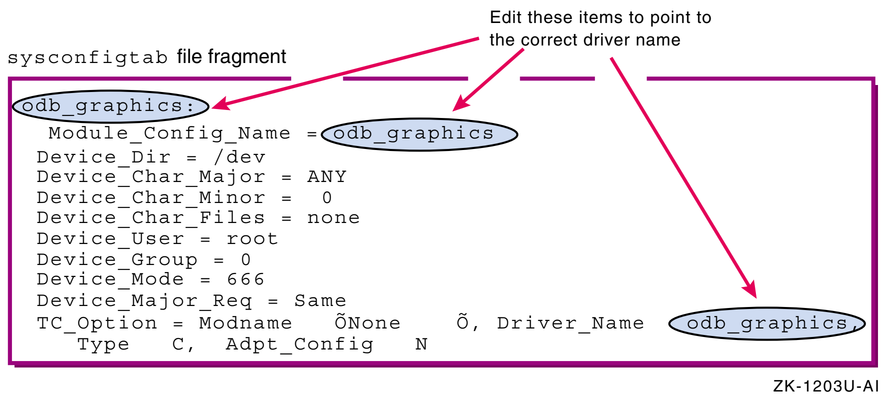
```
# This is the files file fragment for the /dev/none driver
# used to produce the single binary module.
#
 MODULE/STATIC/odb_graphics        standard Binary
 io/OAT100/odb_graphics.c           module   odb_graphics
```

7.2.2 The sysconfigtab File Fragment

The `sysconfigtab` file fragment contains device special file information, bus option data information, and information on contiguous memory usage for statically and dynamically configured drivers. When the user installs a kernel product kit, the driver's `sysconfigtab` file fragment gets appended to the `/etc/sysconfigtab` database. You should place this file fragment in a product directory, such as `/opt/OAT100/etc`.

You do not need to change the `sysconfigtab` file fragment unless you change the driver (subsystem) name. The driver name appears in three places within the file, as shown in Figure 7–3. In the example, the driver runs on a TURBOchannel bus (indicated by the `TC_Option` entry), but a similar set of bus options would be specified for other bus types.

Figure 7–3: Editing the sysconfigtab File Fragment


Edit these items to point to the correct driver name
sysconfigtab file fragment
ZK-1203U-AI

```
odb_graphics:
  Module_Config_Name = odb_graphics
 Device_Dir = /dev
 Device_Char_Major = ANY
 Device_Char_Minor =  0
 Device_Char_Files = none
 Device_User = root
 Device_Group = 0
 Device_Mode = 666
 Device_Major_Req = Same
 TC_Option = Modname   ÕNone    Õ, Driver_Name   odb_graphics,
    Type   C,  Adpt_Config   N
```

See `sysconfigtab`(4) for more information about the `/etc/sysconfigtab` file.

7.2.3 The Object Module File

The *driver*.mod object module file contains the single binary module for both statically and dynamically configured drivers. Include this file in a product directory, such as /opt/OAT100/sys/BINARY, in the root (/) file system.

Caution

You cannot use RIS to link compressed modules. Use the file command to determine if a file is compressed. You see output similar to the following:

```
# file odb_graphics.mod
odb_graphics.mod:
    alpha compressed COFF executable or object module not stripped
```

7.2.4 The Source and Header Files

The *.c (source) and *.h (header) files contain the source code for the device driver. Include these files in a product directory, such as /usr/opt/OAT100/src, when the driver is statically configured and distributed in source form.

7.2.5 The Method Files

The *device*.mth method files contain driver methods that are called during automatic configuration to create device special files for dynamically configured drivers. The subset control program creates links to these device special files in the customer's subsys directory when the driver is installed. The driver method files are on the distribution media and are not installed onto the customer's system. The device driver developer can tell you which method files the subset control program should link to, typically /subsys/*device*.mth. Link the method in a device driver kernel kit only if the driver needs to have device special files created for its devices.

7.3 Producing Distribution Media

After you have tested the subsets, you can produce the distribution media. Distribution media production consists of the following tasks:

1. Rerun the newinv utility to update the master inventory file with the additional installation files. (Section 5.3)
2. Rerun the kits utility to produce updated subsets and control files. (Chapter 5)
3. Edit the /etc/kitcap file. (Section 7.3.1)

4. Build the kernel product kit on the distribution media.
   - Use the `gendisk` utility to build a kit on disk media. (Section 7.3.2)
   - Use the `gentapes` utility to build a `tar` format kit on magnetic tape. (Section 7.3.3)

You can produce the kernel product kit in either `tar` format or direct CD-ROM (DCD) format. Installation time for `tar` format kits is faster than for DCD format kits.

- If your product kit does not access kernel modules during boot, use the `tar` format to compress your kit and save space on the media.

  In `tar` format, the product files in each subset are written to the distribution media as a single file. During installation, the `setld` utility uncompresses the files and moves them onto the target system, preserving the files' original directory structure. Kits distributed in `tar` format install more quickly and consume less space on the distribution media.
- If your product kit accesses kernel modules during the boot process, you must use the DCD format and cannot produce the kit on magnetic tape media.

  In DCD format, the files are written to the distribution media as a UNIX file system where the product files are organized into a directory structure that mirrors the target system. Subsets distributed in DCD format cannot be compressed.

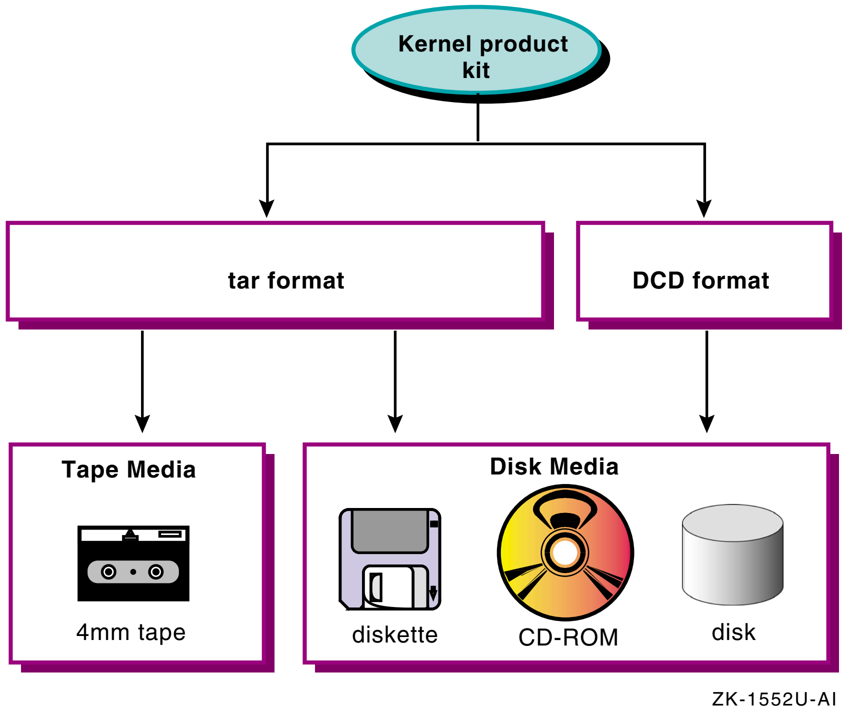
You can distribute kernel product kits on diskette, CD-ROM, or magnetic tape, as follows:

- Diskettes are good for testing purposes or for small products. The product must fit on a single diskette; it cannot span multiple diskettes. Use the `gendisk` utility to produce kits for diskette media.
- CD-ROMs can support large kits or multiple kits on a single media. The kit is first produced on the hard disk, then written onto the CD-ROM. Use the `gendisk` utility to produce the master kit on hard disk. Follow the CD-ROM manufacturer's instructions for writing the kit onto the CD-ROM media.
- Magnetic tape

  Magnetic tape can be used for kernel product distribution if you do not need to access the kernel modules during the boot process. You must use `tar` format; magnetic tape does not support DCD format. Use the `gentapes` utility to produce kits for magnetic tape media.

Figure 7–4 shows the file formats and distribution media available for kernel product kits.

Figure 7–4: Kernel Product Kit File Formats



7.3.1 Editing the /etc/kitcap File

The `gendisk` and `gentapes` utilities refer to the `/etc/kitcap` file, a database containing information about the kits to be built on the system. Each record contains a product code and the names of the directories, files, and subsets that make up the product kit. Before you can build your kit, you must add a media descriptor record to the `/etc/kitcap` database.

Note

If you use the `gendisk` utility to produce your kit on disk distribution media, you can specify an alternate kit descriptor database. See `gendisk(1)` for more information.

Use the following conventions when you add a record to the `/etc/kitcap` file:

- Separate the first field from the rest of the record by a colon (:) for disk media descriptors and by a pipe character (| for tape media descriptors.
- Separate all other fields with colons (:).

- Indicate continuation with a backslash (\ ) at the end of the line.
- Lines starting with a pound sign (#) are comments and are ignored.
- Comments within the record start with pound sign (#) and end with a colon (:). Use this feature sparingly.

The contents of a `kitcap` record differ depending on whether you are producing disk or tape media. You must add one record for each media type on which you plan to distribute your kit.

The contents of the record also depend on the product type you are delivering. For example, the `kitcap` record for a kernel product may require the `kk=true` parameter and the `rootdd=` option. See `kitcap`(4) for more information about the contents of the `/etc/kitcap` file.

7.3.1.1 Disk Media Descriptor

Create a disk media `kitcap` record when you produce kits for distribution on diskette or CD-ROM. The `kitcap` record for disk media contains the following elements:

- The kit name, consisting of two parts:
  - The product code, consisting of the product code and version number specified in the `CODE` and `VERS` fields of the kit's key file (*prodcode*`.k`).
  - The media code `HD` to indicate disk media. This element is followed by a colon (:).
- The partition on the disk media where the product should be placed. The partition is a letter between `a` and `h`. Partition `c` is used most often, as it spans the entire disk.
- The destination directory for the subsets on the disk media. This allows a hierarchical structure so you can put multiple products on one disk, or put parts of one product on different areas of the same disk. You can use multiple destination directories in a `kitcap` record.

  The destination directory field also may include these parameters:

  - `kk=true`

    Indicates that the kit is needed during the boot process. When the `gendisk` utility finds this option, it automatically generates a *kitname*`.kk` file.

  - `rootdd=`*dirname*

    Specifies kit file placement on the distribution media, relative to the kit-specific directory such as `/OAT100/kit`. For example, `rootdd=..` would place the kit's root under the `/OAT100` directory.

- The product description. This entry is taken from key file NAME field. Replace any spaces with an underscore (_) character, for example: Product Description becomes Product_Description.
- The name of the output directory where you created the kit, where the gendisk utility can find the product subsets.
- The instctrl directory, relative to the output directory specification.
- The names of the subsets that make up the kit.

See kitcap(4) for more information about the disk media record format.

See Section 3.3 for information about the key file.

Example 7–1 shows the record to be added to the /etc/kitcap file to produce the ODB kit on disk media:

Example 7–1: Sample Disk Media Descriptor for Kernel Product

```
OAT100HD:c:\
  dd=/OAT100:Orpheus_Document_Builder:/mykit/output:\
  instctrl:OATODB100:OATODBTEMPS100:OATODBKERNEL100
```

Based on the information shown in Example 7–1, the gendisk utility places the kit on the c partition in the / (root) directory of the disk media. The product description is Orpheus_Document_Builder and the kit output directory is named /mykit/output. The kit consists of three subsets: OATODB100, OATODBTEMPS100, and OATODBKERNEL100.

7.3.1.2 Tape Media Descriptor

The kitcap record for tape media contains the following elements:

- The kit name, consisting of two parts:
  - Product code, consisting of the product code and version number specified in the CODE and VERS fields of the kit's key file (*prodcode*.k).
  - The media code, either TK for TK50 tapes or MT for 9-track magnetic tape. This element is followed by a pipe character (|).
- Product description. This entry is taken from the NAME field of the key file.

- Name of the output directory where you created the kit, where the `gentapes` utility can find the subsets.

  Since the `gentapes` utility can take subsets from multiple products and merge them on tape as a combined product, you can specify multiple directories where the `gentapes` utility can find the subsets. There must be one directory entry for each `kitcap` descriptor.
- Three empty `SPACE` files to ensure compatibility with operating system kits. To create the `SPACE` file in the output area of the kit directory structure, issue the following commands:

  ```
  # cd /mykit/output
  # touch space
  # tar -cf SPACE space
  ```
- The `INSTCTRL` image in the `output` directory containing `setld` control information.
- The names of the subsets that make up the kit. Each subset listed must be stored in one of the specified directories.
- Optional volume identifiers `%%`*N*, followed by the names of the subsets to be placed on that volume. You can use multiple tapes.

See `kitcap`(4) for more detailed information about the tape media record format.

Example 7–2 shows the record to be added to the `/etc/kitcap` file to produce the ODB kit on TK50 tapes:

Example 7–2: Sample Tape Media Descriptor

```
OAT100TK|Orpheus Document Builder:\
  /mykit/output:SPACE:SPACE:SPACE:\
  INSTCTRL:OATODB100:OATODBTEMPS100:OATODBKERNEL100
```

The product name, `OAT100`, is the same name that appears in the key file. The product description, `Orpheus Document Builder` also appears in the key file. The name of the output directory is specified as `/mykit/output`, and three `SPACE` files are included for compatibility with operating system kits. The last line of the record contains the `INSTCTRL` image in the output directory and the names of the subsets that make up the kit: `OATODB100`, `OATODBTEMPS100`, and `OATODBKERNEL100`.

7.3.2 Building a Kernel Product Kit on Disk Media

When the product subsets are located in the output area of the kit directory structure, use the `gendisk` utility to create the kit on a disk.

Note

The `gendisk` utility supports diskettes but does not allow you to create a chained diskette kit. A kit written to diskette must fit on a single diskette or be packaged as a set of kits on separate diskettes.

Use the following syntax for the `gendisk` command:

gendisk [-d] [-i] [-k *filename*] [-w] [-v] [*hostname*:] *prodID devname*

`-d`

Creates a distribution disk in direct CD (DCD) format. This means that the distribution disk contains uncompressed file systems that are arranged in the same way as the software will be installed on the system.

`-i`

Creates a distribution disk in ISO 9660 format. This means that the distribution disk contains an ISO 9660-compliant CD-ROM file system (CDFS).

`-k` *`filename`*

Uses an alternate kit descriptor database, *`filename`*, on the local system. You may use either a full absolute pathname or a relative pathname from the directory where you run the `gendisk` utility. You do not have to name the file `kitcap`.

`-w`

If used without the `-v` option, writes the product media without verification. If used with the `-w` option, the `gendisk` utility writes and then verifies the product media.

`-v`

If used without the `-w` option, verifies the product media without writing it first. This assumes that you already have written kit files to the distribution media. If used with the `-w` option, the `gendisk` utility writes and then verifies the product media.

hostname:

The optional *hostname*: operand is the name of a remote machine that contains the `kitcap` file. The utility searches the `/etc/kitcap` file on the remote machine for the *prodID* and uses it for creating the media. The colon (`:`) is a required delimiter for TCP/IP networks, and space is permitted between the colon and the *prodID*. For example, if the product code is OAT100 and you are using the kit descriptor database on node `mynode`, use `mynode:OAT100` for this option.

prodID

The mandatory *prodID* operand is a kit identifier consisting of the product code and version number specified in the `CODE` and `VERS` fields of the kit's key file. See Section 3.3 for information about the key file.

devname

The mandatory *devname* operand specifies the device special file name for a raw or character disk device such as `/dev/rdisk/dsk1`. The `gendisk` utility uses the disk partition specified in the kit descriptor and ignores any partition specified on the command line.

Note

If you do not use either the `-w` or `-v` options, the `gendisk` utility writes and then verifies the product media.

The command shown in Example 7–3 creates a `tar` format kernel product kit for `OAT100` on `dsk0`:

Example 7–3: Sample gendisk Command

```
# gendisk OAT100 /dev/rdisk/dsk0
```

See `gendisk`(1) for more information.

7.3.3 Building a Kernel Product Kit on Magnetic Tape

When the product subsets are located in the output area of the kit directory structure, use the `gentapes` utility to create the kit on magnetic tape. Use the following syntax for the `gentapes` command:

gentapes [-w | -v] [*hostname*:] *prodID* *devname*

`-w`

Writes the product media without verification. Do not use the `-w` option with the `-v` option.

`-v`

Verifies the product media without writing it first. Do not use the `-v` option with the `-w` option.

`hostname:`

The optional *`hostname:`* argument is the name of a remote network machine that contains the kit descriptor database. The `gentapes` utility searches the kit descriptor database on the remote machine for the kit identifier (*`prodID`*`[TK|MT]`) and uses it to create the media. The colon (`:`) is a required delimiter for TCP/IP networks, and space is permitted between the colon and the *`prodID`*. For example, if the product code is OAT100 and you are using the kit descriptor database on node `mynode`, use `mynode:OAT100` for this option.

`prodID`

The mandatory *`prodID`* operand is a kit identifier consisting of the product code and version number specified in the `CODE` and `VERS` fields of the kit's key file. See Section 3.3 for information about the key file.

`devname`

The mandatory *`devname`* operand specifies the device special file name for a no-rewind tape device such as `/dev/ntape/tape0l`. The `gentapes` utility uses the default tape density for the device and ignores any suffix specified on the command line.

Note

If you do not use either the `-w` or `-v` option, the `gentapes` utility writes the tape, rewinds it, and then verifies the files in the kit descriptor.

The command shown in Example 7–4 creates a `tar` format user product kit for `OAT100` on the magnetic tape in `/dev/ntape/dat`:

Example 7–4: Sample gentapes Command

```
# gentapes OAT100 /dev/ntape/dat
```

See `gentapes`(1) for more information about this utility.

7.4 Testing the Distribution Media

Before shipping a kernel product kit to customers, you should test the kit with the same procedures that your customers will use on configurations that resemble your customers' systems.

Run the following tests to test a kernel product kit:

1. Use the `setld` utility to verify that the subsets have been built correctly and that the files get installed into the correct locations on the target system. (Section 7.4.1)
2. Use the `ris` utility to add a kernel product kit into a RIS area and verify that the correct files are present on the kit. Then, register the client system to the RIS area and use the `setld -l` command to install the product on the client system. (Section 7.4.2)

7.4.1 Testing a Kernel Product Kit with the setld Utility

Use the `setld` utility to test a kernel product kit as described in the following procedure for the `OAT100` kit:

1. Log in to the system as `root` or use the `su` command to gain superuser privileges.
2. Place the CD–ROM in the drive.
3. Create a directory to be the media mount point, such as `/cdrom`:

   ```
   # mkdir /cdrom
   ```

4. Mount the CD–ROM on `/cdrom`. For example, if the CD–ROM device is located on the `c` partition of `cdrom0`, enter the following command:

   ```
   # mount -r /dev/disk/cdrom0c /cdrom
   ```

5. Use the `setld` utility to install the kernel product subsets:

   ```
   # setld -l /cdrom/OAT100/kit
   ```

   Your session looks similar to the following:

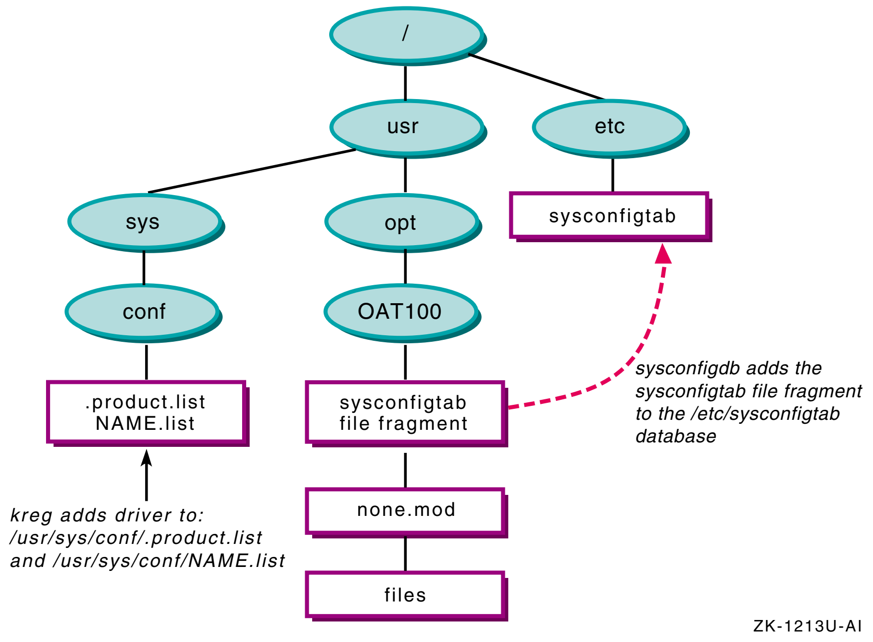
   ```
   *** Enter subset selections ***
   ```

```
The following subsets are mandatory and will be installed automatically
unless you choose to exit without installing any subsets:

      * Document Builder Kernel Support
      * Document Builder Tools

The subsets listed below are optional:

 - Other:
     1) Document Builder Templates

Or you may choose one of the following options:

     2) ALL mandatory and all optional subsets
     3) MANDATORY subsets only
     4) CANCEL selections and redisplay menus
     5) EXIT without installing any subsets

Estimated free diskspace(MB) in root:54.5 usr:347.0

Enter your choices or press RETURN to redisplay menus.

Choices (for example, 1 2 4-6): 2

You are installing the following mandatory subsets:

        Document Builder Kernel Support
        Document Builder Tools

You are installing the following optional subsets:

 - Other:
        Document Builder Templates

Estimated free diskspace(MB) in root:54.5 usr:347.0

Is this correct? (y/n): y

Checking file system space required to install selected subsets:

File system space checked OK.

3 subset(s) will be installed.

Loading subset 1 of 3 ...

Document Builder Tools
   Copying from /cdrom/OAT100/kit (disk)
   Verifying

Loading subset 2 of 3 ...

Document Builder Templates
   Copying from /cdrom/OAT100/kit (disk)
   Verifying

Loading subset 3 of 3 ...

Document Builder Kernel Support
   Copying from /cdrom/OAT100/kit (disk)
   Verifying

3 of 3 subset(s) installed successfully.
```

```
Configuring "Document Builder Tools" (OATODB100)

The installation of the Document Builder Tools (OATODB100)
software subset is complete.

Please read the /opt/OAT100/README.odb file before
using the Document Builder Tools product.

Configuring "Document Builder Templates" (OATODBTEMPS100)

Configuring "Document Builder Kernel Support" (OATODBKERNEL100)

*** Document Builder Kernel Support Product Installation Menu ***

1. Statically configure the graphics support
2. Dynamically configure the graphics support

Type the number of your choice []: 1

*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***

Saving /sys/conf/TEST01 as /sys/conf/TEST01.bck

Do you want to edit the configuration file? (y/n) [n]: n

*** PERFORMING KERNEL BUILD ***
        Working....Fri May 9 14:49:25 EDT 2001

The new kernel is /sys/TEST01/vmunix

The /sys/TEST01/vmunix kernel has been
moved to /vmunix and the changes will take effect
the next time the system is rebooted.
#
```

The `setld` utility displays prompts and messages to guide you through the process of selecting the subsets you want to install. As each subset is loaded, the `setld` utility calls the subset control program as needed, including static or dynamic driver configuration. Figure 7–5 shows the steps the subset control program takes to statically configure the driver.

Figure 7–5: Static Configuration of a Driver



/
usr
etc
sys
opt
sysconfigtab
conf
OAT100
.product.list NAME.list
sysconfigtab file fragment
sysconfigdb adds the sysconfigtab file fragment to the /etc/sysconfigtab database
none.mod
files
kreg adds driver to: /usr/sys/conf/.product.list and /usr/sys/conf/NAME.list
ZK-1213U-AI

6. When the installation is complete, unmount the CD–ROM:

```
# umount /cdrom
```

7. If the product was configured statically, restart the system with the new kernel:

```
# /usr/sbin/shutdown -r now
```

   When the system restarts, the device driver is available on the system.

8. Verify that the installed product functions correctly.

9. Delete the ODB subsets with the `setld -d` command.

```
# setld -d OATODB100 OATODBTEMPS100 OATODBKERNEL100
```

   You see output similar to the following:

```
Deleting "Document Builder Templates" (OATODBTEMPS100).

*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***

Saving /sys/conf/TEST01 as /sys/conf/TEST01.bck

Do you want to edit the configuration file? (y/n) [n]: n

*** PERFORMING KERNEL BUILD ***
        Working....Fri May 9 14:55:31 EDT 2001

The new kernel is /sys/TEST01/vmunix
The /sys/TEST01/vmunix kernel has been
moved to /vmunix and the changes will take effect
the next time the system is rebooted.

Deleting "Document Builder Kernel Support" (OATODBKERNEL100).

Deleting "Document Builder Tools" (OATODB100).
#
```

10. Use the `shutdown` command to restart the system, ensuring that it reboots with the new kernel after the product is removed:

    ```
    # /usr/sbin/shutdown -r now
    ```

    When the system restarts, the product should not be available on the system.

See *Installation Guide* and the `setld`(8) for more information about using the `setld` utility to install layered products.

7.4.2 Testing a Kernel Product Kit in a RIS Area

Use the `ris` utility to test a kernel product kit on a RIS server as described in the following procedure for the `OAT100` kit. See *Sharing Software on a Local Area Network* manual for more information about Remote Installation Services (RIS).

1. Log in to the RIS server as `root` or use the `su` command to gain superuser privileges.
2. Place the CD–ROM in the drive.
3. Create a directory to be the media mount point, such as `/cdrom`:

   ```
   # mkdir /cdrom
   ```

4. Mount the CD–ROM on `/cdrom`. For example, if the CD–ROM device were located on the `c` partition of `cdrom0`, enter the following command:

   ```
   # mount -r /dev/disk/cdrom0c /cdrom
   ```

5. Enter **`/usr/sbin/ris`** to start the `ris` utility.

   You see the RIS Utility Main Menu:

```
*** RIS Utility Main Menu ***

Choices without key letters are not available.

  ) ADD a client
  ) DELETE software products
 i) INSTALL software products
  ) LIST registered clients
  ) MODIFY a client
  ) REMOVE a client
  ) SHOW software products in remote installation environments
 x) EXIT

Enter your choice:
```

6. Enter **`i`** to select `Install software products`. You see the RIS Software Installation Menu:

```
RIS Software Installation Menu:
    1)  Install software into a new area
    2)  Add software into an existing area
    3)  Return to previous menu

Enter your choice:
```

7. Depending on your test environment, enter **`1`** to select `Install software into a new area` or **`2`** to `Add software into an existing area`.

8. Install the software as described in *Sharing Software on a Local Area Network*.

To install the product kit from the RIS server onto the client system, first register the client system with the RIS server and then use the `setld` utility as described in the following procedure:

1. Log in to the RIS server as `root` or use the `su` command to gain superuser privileges.

2. Enter **`/usr/sbin/ris`** to start the `ris` utility. You see the RIS Utility Main Menu:

```
*** RIS Utility Main Menu ***

Choices without key letters are not available.

 a) ADD a client
 d) DELETE software products
 i) INSTALL software products
  ) LIST registered clients
  ) MODIFY a client
  ) REMOVE a client
 s) SHOW software products in remote installation environments
 x) EXIT

Enter your choice:
```

3. Enter **a** to select `ADD a client`.
4. Enter the client information as described in *Sharing Software on a Local Area Network*.
5. Log in to the RIS client as `root` or use the `su` command to gain superuser privileges.
6. Use the `setld -l` command to load the product subsets from the RIS area. For example, if the RIS server is named `test01`, enter the following command:

   ```
   # setld -l test01:
   ```

   The `setld` utility displays prompts and messages to guide you through the installation process as described in *Sharing Software on a Local Area Network*.

   See the *Installation Guide* and the `setld`(8) for more information on using the `setld` utility to install layered products.

Glossary

This glossary defines terms used in this manual.

A

attribute-value pair

In a product kit's key file, attribute-value pairs specify the names and values of the attributes of the kit, such as the name and version of the product. Attribute-value pairs control how the `kits` utility builds the kit and how the `setld` utility installs it.

B

Backus-Naur form

A conventional notation for describing context-free grammars, commonly used for defining syntax in computer languages. It is named for John Backus, developer of FORTRAN, and Peter Naur, developer of ALGOL. The term BNF is often used to refer to grammar specifications based on this form.

See also *postfix*

backward link

A backward link is a symbolic link from the directories in a layered product area to files in the standard hierarchy. The subset control program for a product creates backward links during installation.

boot utility

The `boot` utility performs the initial installation and bootstrap of the operating system. You invoke the `boot` utility from the `>>>` console prompt. See your hardware documentation for information about valid parameters for the `boot` utility on your system.

C

CDFS

A CD-ROM file system (CDFS) is formatted to be compliant with ISO 9660. This lets you read a CD-ROM from multiple computer platforms.

See also *ISO 9660*

CDSL

A context-dependent symbolic link (CDSL) is a special form of symbolic link that dynamically resolves to a member-specific file, depending upon the cluster member accessing the file. CDSLs make it possible to maintain system-specific configuration and data files on file systems shared by the cluster.

See also *cluster*, *cluster member*, *member-specific file*, *shared file*

cluster

A cluster is a loosely coupled collection of servers that share storage and other resources, making applications and data highly available. A cluster consists of communications media, member systems, peripheral devices, and applications. The systems within a cluster communicate over a high-performance interconnect.

See also *cluster alias*, *cluster member*

cluster alias

An IP address used to address all or a subset of the cluster members. A cluster alias makes some or all of the systems in a cluster look like a single system when viewed from outside the cluster.

See also *cluster*, *cluster member*

cluster member

A system configured with TruCluster Server software that is capable of joining a cluster. A cluster member must be physically connected to both a private physical bus for intracluster communications and to at least one shared SCSI bus.

See also *cluster*

compression flag file

The compression flag file is an empty file whose name consists of the product code and the version number with the string `comp` as a suffix; for example, `OAT100.comp`. If the compression flag file exists, the `setld` utility knows that the subset files are compressed.

context-dependent symbolic link

See *CDSL*

control file

One of a collection of files that the `kits` utility places in the `instctrl` directory. These files include the compression flag file, image data file, subset control file, subset inventory file, and subset control programs.

D

Dataless Management Services

See *DMS*

DMS

Dataless Management Services. A service where a server maintains the root (`/`), `/usr`, and `/var` file systems for client computer systems connected to the server by a local area network (LAN).

data hierarchy

In the kit-building directory structure, the data hierarchy contains the files that direct the `setld` utility in making subsets for the kit, such as the master inventory and key files. An `scps` subdirectory contains subset control programs written by the kit developer.

DCD format

A disk media format where files are written to any disk media (CD–ROM, hard disk, or diskette) as a UNIX file system (UFS). Subsets distributed in DCD format cannot be compressed.

See also *tar format*

dependency expression

A dependency expression is a postfix logical expression consisting of subset identifiers and relational operators to describe the current subset's relationship to the named subsets. Subset control programs evaluate dependency expressions under control of the `setld` utility.

See also *Backus-Naur form*, *locking*, *postfix*, *subset dependency*

direct CD-ROM format

See *DCD format*

distribution media

The distribution media for a product kit may be diskette, CD-ROM, or tape. A hard disk is sometimes referred to as a distribution media because it is used as the master copy for a CD-ROM kit.

E

/etc/sysconfigtab

See *sysconfigtab database*

/etc/kitcap

See *kitcap database*

F

forward link

A forward link is a symbolic link that connects a product file in the `/opt`, `/usr/opt`, or `/var/opt` directory to a standard UNIX directory, such as `/usr/bin`. Forward links allow layered products to be installed in a central location (the `opt` directories) and still be accessible to users through the standard directory structure.

G

gendisk utility

The `gendisk` utility is used to produce disk distribution media for a product kit. See `gendisk`(1) for more information.

See also *kitcap database*

gentapes utility

The `gentapes` utility is used to produce magnetic tape distribution media for a product kit. See `gentapes`(1) for more information.

See also *kitcap database*

I

image data file

The image data file is used by the `setld` utility to verify subset image integrity before starting the actual installation process, and contains one record for each subset in the kit.

See also *setld utility*

installed subset status

For the purposes of determining software subset installation status, a subset is installed after it is loaded and its subset control program (SCP) has competed successfully.

See also *loaded subset status*, *SCP*, *subset status file*

ISO 9660

ISO 9660 is an international file system standard adopted by major operating system manufacturers. A file system in this format can be read by most of the standard operating systems. Multiple specification levels allow different file naming conventions. ISO 9660-compliant file systems usually are provided on CD-ROM media.

K

kernel

The kernel is a software entity that runs in supervisor mode and does not communicate with a device except through calls to a device driver.

kernel product

A kernel product is a layered product that runs in kernel space. Users do not directly run kernel products, but the operating system and utilities access them to perform their work.

See also *layered product*, *user product*

key file

A key file identifies the product that the kit represents. You create this file in the `data` directory before running the `kits` utility.

kit

A kit is a collection of files and directories that represent one or more layered products. It is the standard mechanism by which layered product modifications are delivered and maintained on the operating system.

See also *layered product*

kitcap database

The `kitcap` file (located in `/etc/kitcap`) is a kit descriptor database for the `gentapes` and `gendisk` utilities. This database contains product codes, media codes, and the names of the directories, files, and subsets that make up a product description used by these utilities to create distribution media.

The `gentapes` and `gendisk` utilities can specify substitute `kitcap` files in alternate locations.

See also *gendisk utility*, *gentapes utility*

kit descriptor database

See *kitcap database*

kits utility

The `kits` utility creates subsets according to the specifications you define in the master inventory file and key file.

See also *key file*, *master inventory file*, *subset*

L

layered product

A layered product is an optional software product designed to be installed as an added feature of the operating system.

See also *kernel product*, *user product*

loaded subset status

For the purposes of determining software subset installation status, a subset is loaded when its software is copied onto the system. The subset control program may not have completed and you cannot use the software yet.

See also *installed subset status*, *SCP*, *subset status file*

locking

In products installed by the `setld` utility, locking inserts a subset name in the lock file of another subset. Any attempt to remove the latter subset warns the user of the dependency. The user can choose whether to remove the subset in spite of the dependency.

See also *dependency expression*, *subset dependency*

M

master inventory file

A master inventory file lists all the product files and the subsets in which they belong. You create this file in the `data` directory by running the `newinv` utility. The file must exist before you can create the product subsets.

See also *newinv utility*, *subset*

{memb}

A system variable used to support context-dependent symbolic links (CDSLs). The kernel resolves the {`memb`} variable in a CDSL pathname to the string `member`*N*, where *N* is the member ID of the cluster member that is referencing the link. If a cluster member with member ID 2 is accessing a CDSL, the kernel resolves the {`memb`} variable in the pathname to `member2`.

See also *CDSL*, *cluster*, *cluster member*

member-specific file

A file used by a specific cluster member. The contents of a member-specific file can differ for each cluster member, and each member has its own copy of a member-specific file.

See also *cluster*, *cluster member*, *shared file*

N

Network File System
See *NFS*

new hardware delivery
See *NHD*

newinv utility
The `newinv` utility creates the master inventory file from the list of files in the current working directory. The list does not contain all the information needed in the master inventory file. You must edit this file to include information about the subsets to which the files belong.

See also *master inventory file*

NFS
Network File System, an open operating system that allows all network users to access shared files stored on computers of different types. Users can manipulate shared files as if they were stored locally on the user's own hard disk.

NHD
New hardware delivery (NHD) provides installable kernel support for new hardware without requiring a new release of the operating system. NHD kits are available on CD-ROM and from the World Wide Web, and include installation instructions and release notes.

O

output hierarchy
The output hierarchy contains the result of the kit-building process, including the subsets that make up the kit and installation control files to direct the `setld` utility during the installation of the product.

P

postfix
A form of logical expression where the operators follow the operands, rather than being placed between them. Also known as reverse Polish notation, or RPN.

See also *Backus-Naur form*

product code

A unique three-letter code that identifies the manufacturer of a product kit. The examples in this manual use the `OAT` product code for the fictional *Orpheus Authoring Tools, Inc.* You request a product code by electronic mail to `product@dssr.sqp.zko.dec.com`.

product kit

See *kit*

R

Remote Installation Services

See *RIS*

RIS

Remote Installation Services. A remote software distribution method where a server is set up to allow installation of software products over a local area network (LAN). RIS clients are registered on the RIS server to allow them access to specific software products. Using a RIS server makes installation of layered products faster and easier for all the clients on the network.

S

SCP

Subset control program. A program written by the kit developer to perform installation operations that the `setld` utility would not otherwise perform. The `setld` utility invokes the subset control program several times during the installation of the kit.

setld utility

The `setld` utility is the standard software management utility. It allows you to load, delete, inventory, configure, and extract software subsets. See `setld`(8) for more information.

shared file

A file used by all members of a cluster. There is only one copy of a shared file.

See also *cluster*, *cluster member*, *member-specific file*

source hierarchy

In the kit-building directory structure, the source hierarchy contains the files that make up the product. These files are grouped into subsets by the `kits` utility.

subset

The smallest installable software kit module that is compatible with the operating system's `setld` software installation utility. It can contain files of any type, usually related in some way.

subset control program

See *SCP*

subset dependency

A subset dependency is the condition under which a given subset requires the presence (or absence) of other subsets in order to function properly.

See also *dependency expression*, *locking*

subset inventory file

The subset inventory file, generated by the `kits` utility, describes each file in the subset to reflect the exact state of the files in the source hierarchy from which the kit was built. The `setld` utility uses this file to duplicate that state, transferring an exact copy of the source hierarchy to the customer's system.

See also *kits utility*, *setld utility*, *source hierarchy*, *subset*

subset status file

The subset status file (`*subset*.sts`) describes the subset's current installation state and reflects the success or failure of specific phases of the installation or deletion process. The setld utility generates a subset status file for each subset, and removes the file when deleting the subset.

The `/usr/.smdb./*subset*.sts` files are symbolic links to the `/usr/cluster/members/{memb}/.smdb./*subset*.sts` files.

See also *loaded subset status*, *installed subset status*, *SCP*

sysconfigdb utility

The `sysconfigdb` utility is a system management tool that maintains the `sysconfigtab` database.

See also *sysconfigtab database*

sysconfigtab database

The `sysconfigtab` database (located in the `/etc/sysconfigtab` file) contains information about the attributes of subsystems, such as device drivers. Device drivers supply attributes in `sysconfigtab` file fragments, which get appended to the `sysconfigtab` database when the subset control program calls the `sysconfigdb` utility during the installation of a kit.

See also *sysconfigdb utility*

T

tar format

A media format where the product files belonging to the same subset are written to the distribution media as a single file by the `tar` command. During installation, the `setld` utility uncompresses the files, then moves them onto the customer's system, preserving the files' original directory structure. See `tar`(1) for more information.

See also *DCD format*

U

user product

A user product is a layered product that runs in user space. Commands, utilities, and user applications fall into this category.

See also *kernel product*, *layered product*

Index