Tru64 UNIX

Guide to Preparing Product Kits

Part Number: AA-RH9SB-TE

April 2000

Product Version: Tru64 UNIX Version 5.0A

This manual describes the procedures for creating, delivering, and installing layered product kits for use on Compaq Tru64™ UNIX®operating systems.

Compaq Computer Corporation
Houston, Texas

Contents

4 Producing User Product Kits

5 Producing Kernel Product Kits

Glossary

Index

Examples

Figures

Tables

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:

- Creating, testing, and installing hardware product kits (*Chapter 1*, *Chapter 6*)
- Updating `gendisk` functionality and syntax (*Section 4.2.2*, *Section 5.3.2*, *Section 6.3.2*)
- Correcting `kitcap` record examples (*Example 4–1*, *Example 4–2*, *Example 5–1*, *Example 5–2*, *Example 6–3*, *Example 6–4*)
- Including changes for clusters environments (*Section 2.3.4*, *Section 3.6.5*)
- Adding additional libraries for subset control programs (*Section 3.4*, *Table 3–4*, *Section 3.4.5.3*, *Section 3.5.3*)

Previous versions of the manual are available on the World Wide Web at the following location:

`http://www.unix.digital.com/faqs/publications/pub_page/doc_list.html`

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, create kit production files, prepare subset control programs, and produce the subsets and related control files
Chapter 4	Describes how to create, test, and deliver user product kits
Chapter 5	Describes how to create, test, and deliver kernel product kits
Chapter 6	Describes how to create, test, and deliver hardware product kits
Appendix A	Describes how to create a CD-ROM that lets you upgrade your processor firmware at the same time that you install the operating system
Glossary	Defines terms used in this manual

Related Documentation

Icons on Tru64 UNIX Printed Books

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

G	Books for general users
S	Books for system and network administrators
P	Books for programmers
D	Books for device driver writers
R	Books for reference page users

Some books in the documentation help meet the needs of several audiences. For example, the information in some system books 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 books in the Tru64 UNIX documentation set.

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

`http://www.unix.digital.com/faqs/publications/pub_page/pubs_page.html`

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

- *Sharing Software on a Local Area Network*

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

- *Writing Device Drivers*

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

- *Installation Guide*

  This manual describes the procedures to perform an Update Install\.ation or a basic 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.

- *Installation Guide — Advanced Topics*

  This manual describes the procedures for performing an advanced installation of the operating system on all supported processors and single-board computers.

- *System Administration*

  This manual 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*

  This section describes commands for system operation and maintenance. It is intended for system administrators. In printed format, this section is divided into two volumes.

- *Release Notes*

  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. The *Release Notes* are intended for anyone installing the operating system or for anyone using the operating system after it is installed.

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`

- Mail:

  Compaq Computer Corporation
  UBPG Publications Manager
  ZKO3-3/Y32
  110 Spit Brook Road
  Nashua, NH 03062-2698

  A Reader's Comment form is located in the back of each printed manual. The form is postage paid if you mail it in the United States.

Please include the following information along with your comments:

- The full title of the book and the order number. (The order number is printed on the title page of this book and on its back cover.)
- 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.
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, Ctrl/C).

1

Introducing Product Kits

This guide is intended to provide 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 contains 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. Hardware product kits can be delivered only on CD-ROM.

Note

A consolidated firmware CD-ROM lets you upgrade your processor firmware at the same time that you install the operating system. Appendix A describes how to create a consolidated firmware CD-ROM.

Before building a kit, consider the kind of product the kit represents:

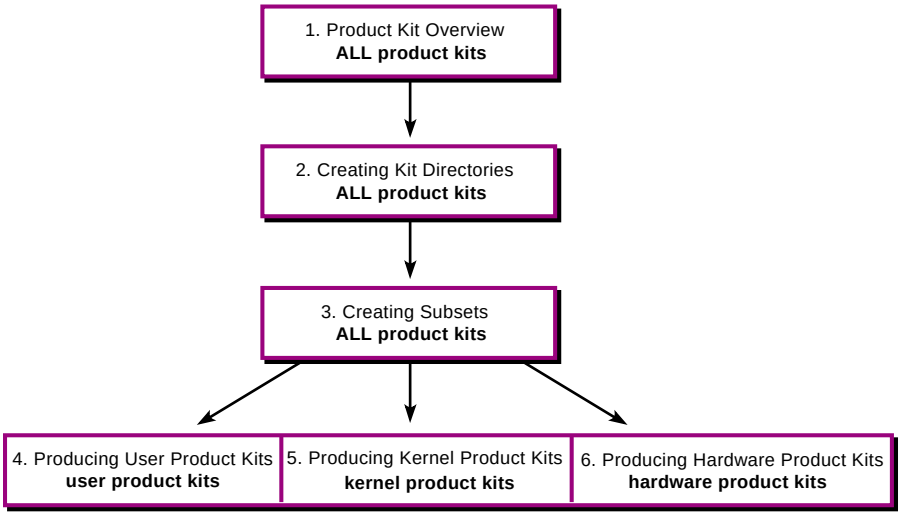
- Does it run in user space or kernel space?
- Is it used during the initial installation and bootstrap of the operating system?
- Is the kit being built for a hardware product?

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.

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

Figure 1–1: Using This Manual



ZK-1554U-AI

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 4, Chapter 5, and Chapter 6 describe the files specific to user, kernel, and hardware 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 three 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.

  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.

- 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.

- Hardware product

  A hardware product provides the kernel modules necessary for the operating system to support new or additional hardware. Before a system manager can use the hardware, the hardware product kit must be configured into the kernel, since there are no kernel modules available to handle potential kernel and user requests for the hardware. The hardware product kit contains a kernel product — the device driver for the hardware — and other files needed for configuring the driver into a kernel at system installation time. A hardware product kit can be installed either concurrent with or after the operating system installation.

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.

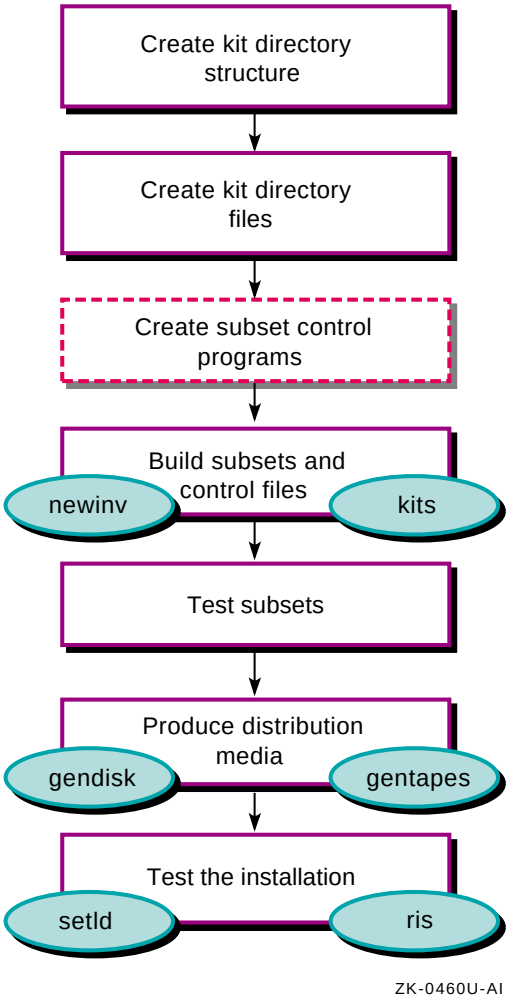
- Direct CD–ROM (DCD) format

  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. Hardware product kits must be produced on CD-ROM in DCD format.

1.4 Kit-Building Process

Figure 1–2 illustrates 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 1–2, the commands enclosed in ovals perform the associated steps of the kit-building process.

Figure 1–2: Steps in the Kit-Building Process



The kit-building process consists of the following steps:

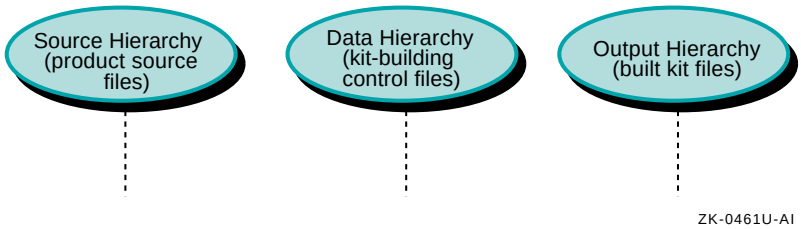
1. Create the kit directory structure that contains the source files.

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

   - A source hierarchy, which contains all the files that make up the product
   - A data hierarchy, which contains files needed to build the kit
   - An output hierarchy, which holds the result of the kit-building process — one or more subsets that make up the product kit

Figure 1–3 illustrates these directory hierarchies.

Figure 1–3: Kit Directory Hierarchies



This directory structure is the same for user products, kernel products, and hardware product kits. Only the contents of these directories differ among the product types. For example, a hardware product kit needs additional files that are unique to this specific kit type. Refer to Chapter 4, Chapter 5, and Chapter 6 for additional 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, and subset definitions, and additional files for kernel and hardware 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 and hardware 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. Refer to 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. At this point, you must decide whether to create the kit in DCD format or in `tar` format. To do this, use the `gendisk` or `gentapes` utility. If you are creating a kit for a hardware product, you also must modify the kit and add files to support your system's bootstrap link. Hardware product kits must use DCD format, and can be produced only with the `gendisk` utility.

7. Test product kit media.

   After you have successfully created the kit, you should test its installation from the new media. Chapter 4, Chapter 5, and Chapter 6 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.

This chapter discusses the following topics:

- Obtaining a unique three-letter manufacturer's product code (Section 2.1)
- Creating the directory structure needed to build a product kit (Section 2.2)
- Populating the source directory on the kit-building system (Section 2.3)

2.1 Obtaining a Unique Product Code

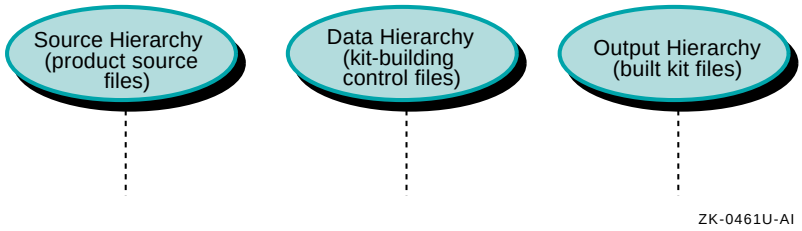
Before you can create a product kit, you must have a unique three-letter product code. To obtain this product code, send electronic mail to `Product@DSSR.enet.dec.com`. You use this product code and a product version number that you assign to name your product-specific subdirectories.

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 illustrates these directory hierarchies.

Figure 2–1: Kit Directory Hierarchies



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.

To create the kit-building directory structure, issue the `mkdir` command for each required directory, as in the following example:

Example 2–1: Creating the Kit-Building Directories

```
# mkdir /mykit
# mkdir /mykit/src
# mkdir /mykit/data
# mkdir /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 kit 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)
- Placing files in the kit source directory (Section 2.3.2)
- Setting up the sample product kit source directory (Section 2.3.3)
- Using context-dependent symbolic links (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

File placement in a standard directory structure depends upon whether the system has a separate `var` file system.

- If the system has a separate `var` file system, use the `/var/opt` directory.
- If the system does not have a separate `var` file system, use the `/usr/var/opt` directory.

Examples in this manual assume that `var` is not a separate file system, and show the `/usr/var/opt` directory.

A standard directory structure has the following advantages:

- Avoiding 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`.

- Locating 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.

- Serving 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 5.2 describes the additional installation files you need for a kernel product. Figure 5–1 illustrates these files.
- Section 6.2 describes the additional installation files you need for a hardware product. Figure 6–1 illustrates 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`

  Install files that are required at bootstrap time, such as device drivers, 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

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

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

  Install files that users can read and modify, such as data lists, in a product-specific subdirectory of the /usr/var/opt directory.

2.3.2 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 was shipped as part of the old version of the operating 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.

2.3.3 Setting Up the Sample Product Kit Source Directory

Figure 2–2 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.*.

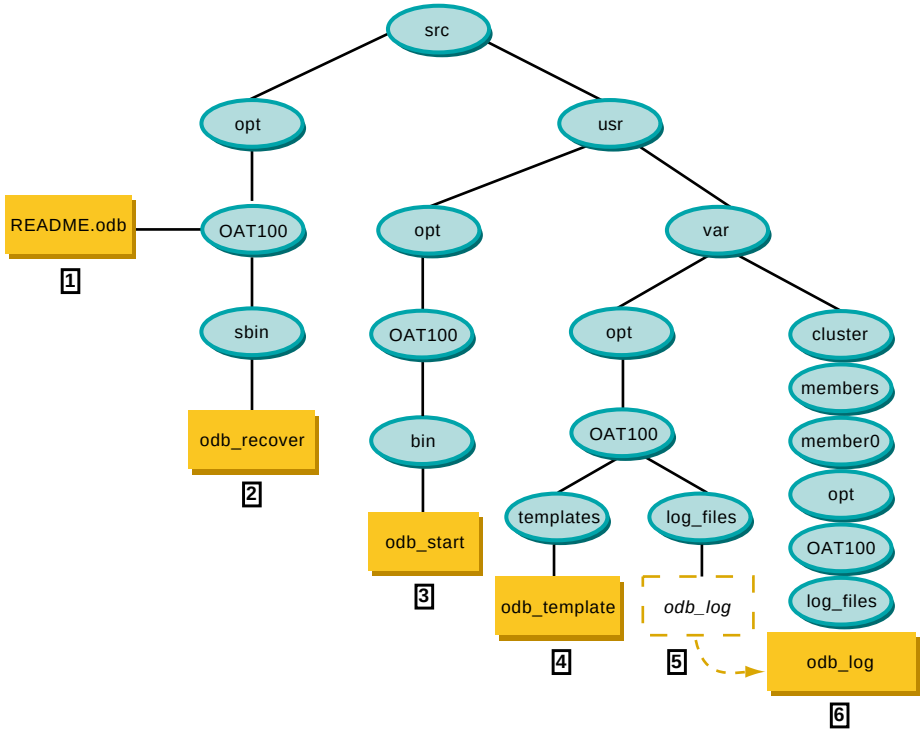
Caution

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

Figure 2–2: Sample Product Kit Source Directory



ZK-1201U-AI

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. The file paths in the source directory are different from the paths provided by the kit developers to avoid name space conflicts with other products.

1 `/README.odb` — a text file containing general product information

This file is installed in the `root (/)` file system and is placed in the `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 tool 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/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.

5 /usr/var/log_files/odb_log — the context-dependent symbolic link (CDSL) for the odb_log file.

   This symbolic link is installed in the /usr/var file system and is placed in the usr/var/opt/OAT100/log_files source directory.

6 /usr/var/cluster/members/member0/log_files/odb_log — the file listing all of the ODB documents created by a user. The odb_log file can be modified by users.

   This file is installed in the /usr/var file system and is placed in the usr/var/cluster/members/member0/opt/OAT100/log_files 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 would make the `odb_start` command available to the user 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, the `/usr/opt/OAT100/bin/odb_start` command is shipped in the ODB kit and as part of the standard operating system in `/bin/odb_start`. When the user enters 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.

2.3.4 Using Context-Dependent Symbolic Links

If you are preparing a product kit that may run on a cluster, you need to use a context-dependent symbolic link (CDSL) to a member-specific file instead of using a shared file 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 `/etc/sysconfigtab` file contains hardware information that can be different for each cluster member.

- A shared file is used by all members of a cluster. There is only one copy of a shared file.

  For example, log files can be shared by all cluster members.

This section provides the following information:

- When to use CDSLs in your product kit (Section 2.3.4.1)
- How to identify CDSLs (Section 2.3.4.2)
- How to create CDSLs for your product kit (Section 2.3.4.3)
- Restrictions on using CDSLs in your product kit (Section 2.3.4.4)

2.3.4.1 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. Once a directory is a CDSL you cannot change it back to a regular directory. If a directory contains all member-specific files, make each file a CDSL.

2.3.4.2 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.

2.3.4.3 Creating CDSLs

Follow these steps to create CDSLs:

1. Determine the file system where the CDSLs should reside: root (/), /usr, or /usr/var.
2. Log in as root or use the su command to gain superuser privileges.

3. Use the `mkdir` command to create the member-specific directories for each file system. For example, if you are building the `OAT100` kit in the `/mykit` directory, enter the following commands for the root (`/`), `/usr`, and `/usr/var` file systems:

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

4. Copy the `odb_log` file to the member-specific area. Refer to the sample `OAT100` product kit source directory shown in Figure 2–2.

5. Use the `ln -s` command to create the CDSLs in the shared file locations. Specify relative pathnames, as shown in the following example:

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

Note

In the previous step, you created member specific files in the `cluster/members/member0` directory, but in this step you created symbolic links to the `cluster/members/{memb}` directories. When your product kit is installed on a cluster, this allows the kernel to resolve the `{memb}` variable to `member`*N*, where *N* is the member ID of the cluster member that is referencing the file.

6. Use the `ls -l` command to verify the symbolic link, as shown in the following example:

```
# ls -l odb_log
lrwxrwxrwx  1 root  system    36 May 19 15:46 odb_log@ -> \
  ../../../cluster/members/member0/opt/OAT100/log_files/odb_log
```

Note

The backslash (\) character shows line continuation and is not present in the actual output.

2.3.4.4 Restrictions

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

1. If you are preparing a product kit to run on a version of the operating system prior to Version 5.0, you cannot use CDSLs.
2. If you are creating a product kit on a system that 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 it on a system running Version 5.0 or higher of the operating system.
3. If you are creating a product kit on a system that is a cluster member running Version 5.0 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`} link to the cluster ID of the member where you are creating the kit, `member`*N*.

3

Creating 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.

Note

A file's physical location is not necessarily a factor in determining the subset to which it belongs.

A kit developer 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)
4. Optionally, create subset control programs (SCPs). (Section 3.4)
5. Use the `kits` utility to produce the subsets and related control files. (Section 3.5)
6. 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. (Section 3.6)
7. Optionally, update the master inventory file after you have created subsets. (Section 3.7)

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.

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.

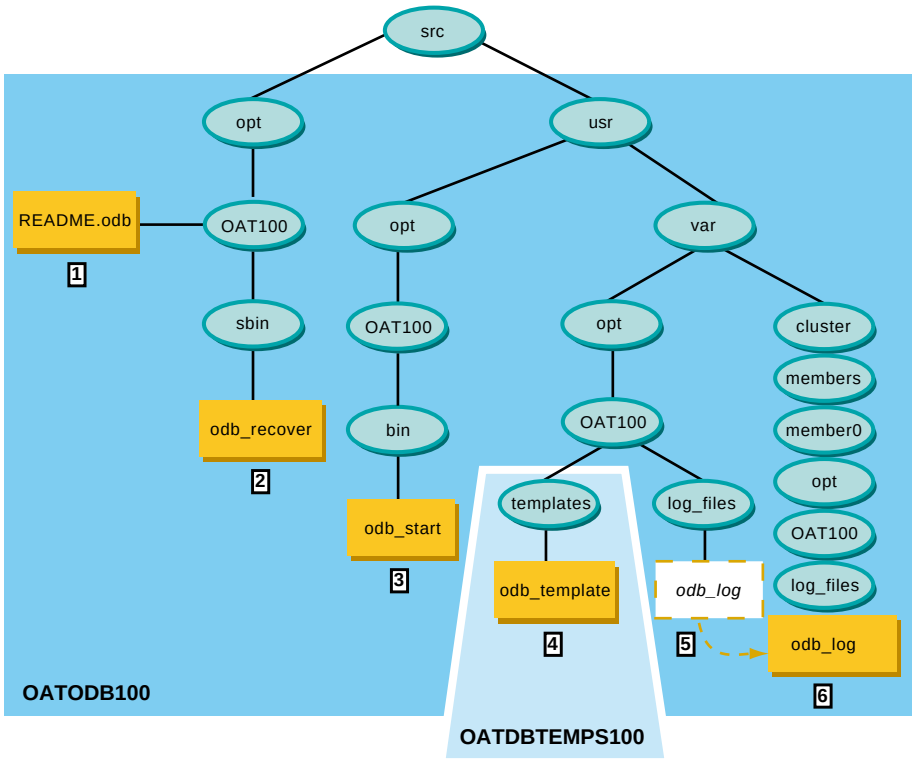
The fictitious ODB user product requires two subsets:

- `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 Kit Subsets and Files



ZK-1216U-AI

3.2 Creating the Master Inventory File

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

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

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

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

Be extremely careful when you edit the master inventory file. Separate fields in this file with single Tab characters, 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.

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

Caution

The first time that you run the `newinv` utility, 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.

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 during 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`.

Table 3–1: Master Inventory File (cont.)

Field	Description
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.

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.
```

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

1 In Example 3–1, the ./opt, ./usr, ./usr/opt, ./usr/var, and ./usr/var/opt directories use the RESERVED subset identifier. This indicates that the setld utility should not allow the directory to be overwritten if it already exists on the customer's system. Use the RESERVED subset identifier for any file or directory that is shipped with another product and already may be present on the customer's system.

The Flags field is set to 0 (zero), indicating that this directory cannot be changed.

2 The ./usr/var/opt/OAT100/log_files/odb_log file has the OATODB100 subset identifier, indicating that the file belongs in that subset.

Set the Flags field to 2, indicating that the file may change on the customer's system.

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.

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

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
#
%%
OATODB100 . 0 'Document Builder Tools' 5
OATODBTEMPS100 OATODB100|OSFDCMT??? 2 'Document Builder Templates' 6
```

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 (=). You can include comment lines, which begin with a number sign (#).

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. User and kernel product kit subsets may be compressed, but hardware product kit subsets must be 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. You cannot include comments in this section of the key file. Table 3–3 describes the subset descriptor fields.

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

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

6 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.

The Subset Description field must be enclosed in single quotes.

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 mail to `Product@DSSR.enet.dec.com` to request a product code.
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 illustrated 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 is 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). Do not compress subsets on hardware product kits. Compressed files require less space on the distribution media (sometimes as little as 40% 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|OSFDCMT505`), 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. 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 character (`%`) is reserved in this field and must not be used for layered products.

3.4 Creating Subset Control Programs

This section describes common tasks required to write subset control programs for product kits. The following topics are discussed:

- Creating SCP source files (Section 3.4.1)
- Setting up initial SCP processing (Section 3.4.2)
- Working in a cluster environment (Section 3.4.3)
- Working in a DMS environment (Section 3.4.4)
- Associating SCP tasks with `setld` processing phases (Section 3.4.5)
- Stopping the installation (Section 3.4.6)
- Creating SCPs for different product types (Section 3.4.7)

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 need to write a subset control program:

- Some of your kit's files have to be customized before the product will work properly.
- You want to offer the user the option to install some of the files in a nonstandard location.
- You want to register and statically or dynamically configure a device driver.
- Your kit depends on the presence of other products.
- 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

Be aware of the following restrictions:

- Use the *subset-id*`.lk` files for subset dependencies. Use the library routines described in Section 3.4.2.1 to determine if a subset is installed.
- Effective with this version of the operating system, *subset-id*`.dw` files are no longer created. Use the library routines described in Section 3.4.2.1 to determine if a subset is corrupt.

3.4.1 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.

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 create the subsets as described in Section 3.5, 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.

3.4.2 Setting Up Initial SCP Processing

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

- Including library routines (Section 3.4.2.1)
- Setting global variables (Section 3.4.2.2)

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

3.4.2.1 Including Library Routines

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:

```
. /usr/share/lib/shell/lib_name
```

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

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

Table 3–4: Available SCP Library Routines

Purpose	Library Routine
Architecture checking	STL_ArchAssert
Dependency locking	STL_LockInit[a]
	STL_DepLock[a]
	STL_DepUnLock[a]
Dataless environment checking	STL_IsDataless
	STL_NoDataless
Forward symbolic linking	STL_LinkCreate
	STL_LinkRemove
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.

Table 3–5 lists the library installation routines available in the `libinstall` shell script.

Table 3–5: Available Library Installation Routines

Purpose	Library Routine
Cluster member identification	INST_GetMemberID

Table 3–6 lists the library routines available in the `libswdb` shell script.

Table 3–6: Available Installed Software Database Library Routines

Purpose	Library Routine
Installed subsets list	`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_IsCorrupt`
	`SWDB_IsInstalled`
Subset dependencies	`SWDB_IsLocked`
	`SWDB_ListLockingSubsets`

3.4.2.2 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 at the beginning of all phases except the `M` phase to initialize global variables. The control file is not read before the `M` phase.

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

Table 3–7 lists global variables that the subset control program can use to access information about the current subset.

Table 3–7: 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`

Table 3–7: STL_ScpInit Global Variables (cont.)

Variable	Description
_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
_OOPS	The NULL string, for dependency checking[a]

[a] 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.

3.4.3 Working in a Cluster Environment

A cluster is a loosely-coupled collection of servers that share storage and other resources, providing high availability of applications and data. 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, setld utility phases can run on each cluster member. Any changes made by the SCP must first check to see 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 member-specific file and would cause cluster-wide problems.
- 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 3–8 describes how setld utility phases behave when your SCP runs on a cluster.

Table 3–8: 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 the SCP is run, invoking the `setld` utility.
`PRE_L`	Runs only once for the entire cluster. If you must run member-specific operations, include SCP code that performs the operation on all cluster members. Do not decline software loading because of one cluster member.
`POST_L`	Runs only once for the entire cluster. If you must run member-specific operations, include SCP code that performs the operation on all cluster members. Do not decline software configuration because of one cluster member.
`C INSTALL`	Runs once on each cluster member. If your SCP needs to access member-specific files, perform those operations during the `C INSTALL` phase.
`C DELETE`	Runs 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.[a]
`PRE_D`	Runs only once for the entire cluster. If you must run member-specific operations, include SCP code that performs the operation on all cluster members. 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.[a]
`POST_D`	Runs only once for the entire cluster. If you must run member-specific operations, include SCP code that performs the operation on all cluster members. 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.[a]
`V`	No cluster-specific restrictions.

[a] Refer to the *Installation Guide* and the `setld`(8) reference page for recovery information.

3.4.4 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. Refer to *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. You must also 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`

Checks to see if a subset is being installed into a dataless environment.

`STL_NoDataless`

Declines installation of a subset into a dataless environment.

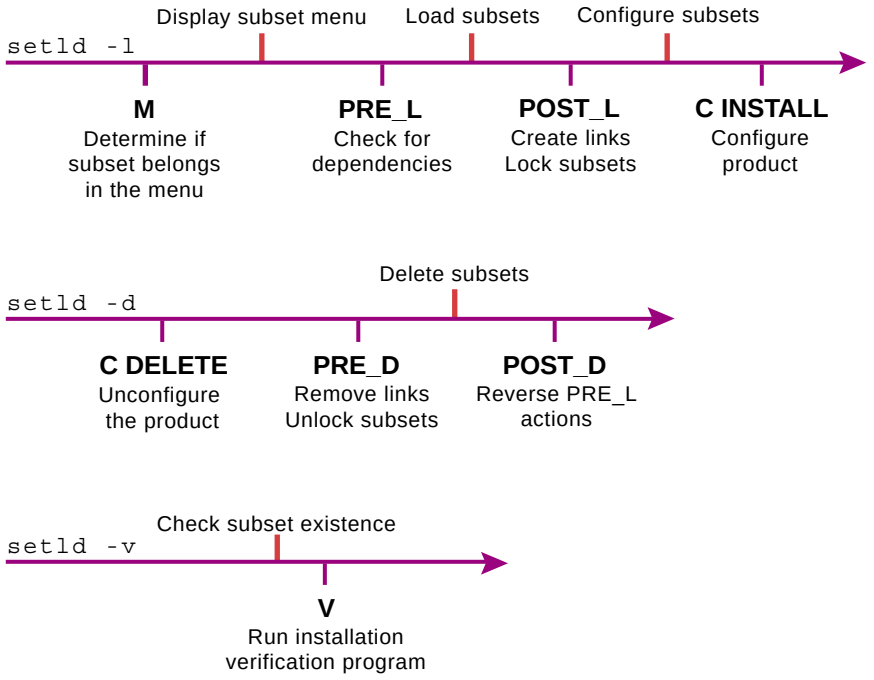
3.4.5 Associating SCP Tasks with setld Utility Phases

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 performing dependency checks or creating links, should be performed only during specific phases.

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

Figure 3–2 shows `setld` utility time lines for the `–l`, `–d`, and `–v` options.

Figure 3–2: Time Lines for setld Utility Phases



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.

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 checks the value of the `ACT` environment variable and any command line arguments to determine the required action.

Caution

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 3.4.5.1)
- Before loading the subset: `PRE_L` phase (Section 3.4.5.2)
- After loading the subset: `POST_L` phase (Section 3.4.5.3)
- After securing the subset: `C INSTALL` phase (Section 3.4.5.4)
- Before deleting a subset: `C DELETE` phase (Section 3.4.5.5)
- Before deleting a subset: `PRE_D` phase (Section 3.4.5.6)
- After deleting a subset: `POST_D` phase (Section 3.4.5.7)
- Verifying the subset: `V` phase (Section 3.4.5.8)

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.

Refer to the `setld`(8) and `stl_scp`(4) reference pages for more information about the `setld` utility and conventions for subset control programs.

3.4.5.1 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 is run only on the member where the setld utility is invoked.

Example 3–3 shows a sample setld installation menu, listing the subsets available for installation.

Example 3–3: 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, check 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 3–4, the subset control program is checking to determine the type of processor on which it is running. In this example, there is no special code for the RIS extract case.

Example 3–4: 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
#
```

Example 3–4: Sample Test for Alpha Processor During M Phase (cont.)

```
# 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 the previous example, the SCP returns the following codes to the `setld` utility:

`0` - offer the subset on the menu
`1` - do not offer the subset on 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. Refer to Section 3.4.4 and *Sharing Software on a Local Area Network* for more information about DMS.

3.4.5.2 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 in the PRE_L phase, include code in your SCP that performs the operation on all cluster members.

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 was shipped as part of the old version of the operating system with the version of the file currently 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 3.4.5.7.

In Example 3–5, the subset control program checks 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 3–5: 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.
#
```

Example 3–5: Sample Backup of Existing Files During PRE_L Phase (cont.)

```
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 extention.
  #
  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.

The PRE_L processing returns the following codes to the setld utility:

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

3.4.5.3 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 and perform dependency locking 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 in the POST_L phase, include code in your SCP that performs the operation on all cluster members.

Do not decline software configuration because of one cluster member.

Perform member-specific operations in the C INSTALL phase, not in the POST_L phase.

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.

Example 3–6 uses STL_LinkBack in the POST_L phase to create a link named /opt/OAT100/sbin/ls that refers to the real file /sbin/ls, and removes the link in the PRE_D phase.

Example 3–6: Sample Backward Link Creation During POST_L Phase

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

[1] The STL_LinkBack routine creates a backward link in the product-specific area. If you used the STL_LinkCreate routine, it would create an unacceptable link in the OSF name space. 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. The STL_LinkRemove routine is used only to remove links created by the STL_LinkCreate routine.

In the previous example, 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 installs a subset. After successful installation, that subset is then available for dependency checks 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 checks 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.

3.4.5.4 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 is run on every 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 3–7 shows the C INSTALL portion of the SCP that issues a message upon successful subset installation.

Example 3–7: 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 read the README file
  # before using the product.
  #
    echo "
The installation of the $_DESC ($_SUB) 2
software subset is complete.
Please read the /opt/$_PVCODE/README.odb file before 3
using the $_DESC product." 2
  ;;
 .
 .
 .
 esac
 ;;
 .
 .
 .
```

1 During the C phase, the SCP checks to see 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 read the /opt/OAT100/README.odb 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 the previous example, the SCP returns the following codes to the setld utility:

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

3.4.5.5 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 is run only once for the whole cluster. If you must run member-specific operations in the C DELETE phase, include code in your SCP that performs the operation on all cluster members.

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 3–8 shows the C DELETE portion of the SCP that would reverse any changes made during the C INSTALL phase.

Example 3–8: 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.
 #
 case $1 in
 INSTALL)
  #
  # Output a message letting the user know
  # that they should read the README file
  # before using the product.
  #
    echo "
The installation of the $_DESC ($_SUB)
software subset is complete.
Please read the /opt/$_PVCODE/README.odb 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 3–7, no action is taken in the C DELETE phase.

In the previous example, the SCP returns the following codes to the setld utility:

0 - continue with the deletion
1 - terminate the deletion

3.4.5.6 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 and dependency locks.

Note

In a cluster environment, the `PRE_D` phase is run on every cluster member. You must be able to run any SCP operations in the `PRE_D` phase more than once without causing a problem.

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 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.

Perform member-specific operations in the `C INSTALL` phase, not in the `PRE_D` phase.

You can call the following routines to remove links and unlock subsets:

`STL_LinkRemove`

Removes links created by `STL_LinkCreate` and restores any original files that `STL_LinkCreate` saved. Call `STL_ScpInit` first to initialize required global variables. The `STL_LinkRemove` routine cannot remove modified links.

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

In Example 3–6, 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 3–9 shows the SCP removing this link in the `PRE_D` phase.

Example 3–9: 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. The `STL_LinkRemove` routine is used only to remove links created by the `STL_LinkCreate` routine.

In the previous example, the SCP returns the following codes to the `setld` utility:

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

3.4.5.7 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 in the `POST_D` phase, include code in your SCP that performs the operation on all cluster members.

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 3–10, the subset control program checks 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 3–10: 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
 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 3–5.

In the previous example, the SCP returns the following codes to the `setld` utility:

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

3.4.5.8 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 each subset. Any `V` phase processing included in the subset control program is executed at this time.

The `setld` utility checks for 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.

3.4.6 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 upon exiting from the particular phase for which it was called. If the subset control program returns a status of 0 (zero), the `setld` utility assumes that the subset control program is satisfied that the `setld` process should continue.

3.4.7 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 3.4.7.1)
- Creating SCPs for kernel product kits (Section 3.4.7.2)
- Creating SCPs for hardware product kits (Section 3.4.7.3)

3.4.7.1 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 3–11 shows a subset control program for the ODB user product, illustrating the types of operations that can be performed during different `setld` phases.

Example 3–11: Sample ODB User Product SCP

```
#!/sbin/sh

#
# Load all of the standard SCP library routines
#
. /usr/share/lib/shell/libscp

#
# 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 $1 in
 -l)
  #
  # Examine the machine architecture to be sure
  # that this software is being installed on an
  # alpha machine.  If it is not, exit with an
  # error status so that setld will not display
  # this subset on the menu of subsets to load.
  #
  ARCH=`./bin/machine`
  [ "$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
  #
```

Example 3–11: Sample ODB User Product SCP (cont.)

```
  # If the file to be backed up exists, create
  # a backup copy with a .OLD extention.
  #
  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
 # used on a setld delete.
 #
 case $1 in
 INSTALL)
  #
  # Output a message letting the user know
  # that they should read the README file
  # before using the product.
  #
  echo "
The installation of the $_DESC ($_SUB)
software subset is complete.

Please read the /opt/$_PVCODE/README.odb file before
```

Example 3–11: Sample ODB User Product SCP (cont.)

```
using the $_DESC product."
  ;;

 DELETE)
  ;;

 esac
 ;;

esac

exit 0
```

This program illustrates one method of using the value of the ACT environment variable to determine what actions to perform.

Note

Example 3–11 is the source for the SCP file fragments shown in Section 3.4.5.

3.4.7.2 Kernel Product Kit SCPs

In addition to the optional processing described in Section 3.4.5, a subset control program for a kernel product such as a device driver must also 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 3–12 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 3–12: Sample ODB Kernel Product SCP

```
#!/sbin/sh

#
# Load all of the standard SCP library routines
#
. /usr/share/lib/shell/libscp

#
# Load the standard Error library routines
# (location of the Error routine)
#
. /usr/share/lib/shell/Error

#
# Load the standard String library routines
# (location of the ToUpper routine)
#
. /usr/share/lib/shell/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
```

Example 3–12: Sample ODB Kernel Product SCP (cont.)

```
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"
 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
```

Example 3–12: Sample ODB Kernel Product SCP (cont.)

```
   #
   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
  # 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
```

Example 3–12: Sample ODB Kernel Product SCP (cont.)

```
  #
  # Remove the entry from the /etc/sysconfigtab file
  #
  sysconfigdb -d odb_graphics
  ;;
 esac
 ;;

esac

exit 0
```

3.4.7.3 Hardware Product Kit SCPs

In addition to the optional processing described in Section 3.4.5, a subset control program for a hardware product such as a device driver also must configure the driver statically into the kernel.

Example 3–13 shows the subset control program for the single binary module associated with the `odb_kernel` driver. The user can configure this single binary module statically into the kernel.

Example 3–13: Sample ODB Hardware Product SCP

```
#!/sbin/sh

#
# Load all of the standard SCP library routines
#
. /usr/share/lib/shell/libscp

#
# Load the standard Error library routines
#
. /usr/share/lib/shell/Error

#
# Update the specified file with the latest entries from the sysconfigtab
# file fragment shipped with the harwdare subset.  Any new sysconfigtab
# subsystem entries will be added to the specified file, while any existing
# subsystem entries will be merged.
#
update_sysconfigtab()
{
 #
 # This is the system file to be modified
 #
 MODFILE=$1

 #
 # This is the location of the sysconfigtab file fragment
 # for the hardware subset.
 #
 STANZA=./opt/$_PVCODE/etc/sysconfigtab
```

Example 3–13: Sample ODB Hardware Product SCP (cont.)

```
 #
 # Merge the sysconfigtab file fragment (STANZA) with the
 # file to be modified from the system (MODFILE)
 #
 sysconfigdb -t $MODFILE -f $STANZA -m
 STATUS="$?"
 [ "$STATUS" -eq 255 ] && STATUS=0

 return $STATUS
}


#
# This is the main loop where the proper code for each phase of
# the SCP file is executed.
#
Main()
{
 #
 # 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

 M)
  ;;

 POST_L)
  #
  # Update the ./etc/.proto..sysconfigtab file
  #
  update_sysconfigtab ./etc/.proto..sysconfigtab || {
   Error "
Error: Failed to add entries to ./etc/.proto..sysconfigtab file.\n"
   exit 1
  }

  #
  # Update the ./etc/sysconfigtab file
  #
  update_sysconfigtab ./etc/sysconfigtab || {
   Error "
Error: Failed to add entries to ./etc/sysconfigtab file.\n"
   exit 1
  }
  ;;

 C)
  case $1 in
  INSTALL)
   #
   # Register the product with the kernel using kreg
```

Example 3–13: Sample ODB Hardware Product SCP (cont.)

```
   #
   /sbin/kreg -l $_PCODE $_SUB \
    /opt/$_PVCODE/sys/BINARY
   ;;
  DELETE)
   #
   # Deregister the product using kreg
   #
   /sbin/kreg -d $_SUB
   ;;
  esac
  ;;

 esac

 exit 0
}

[ "$CHECK_SYNTAX" ] || Main $*
```

3.5 Producing Subsets

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

- The compression flag file (Section 3.5.1)
- The image data file (Section 3.5.2)
- The subset control files (Section 3.5.3)
- The subset inventory files (Section 3.5.4)

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.

Refer to the `kits`(1) reference page for more information.

Caution

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.

For example, the following commands build the subsets for the ODB product kit:

```
# 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

        *** 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.
```

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 specify the `COMPRESS` attribute in the key file.

[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`

These files are described in Table 3–9.

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

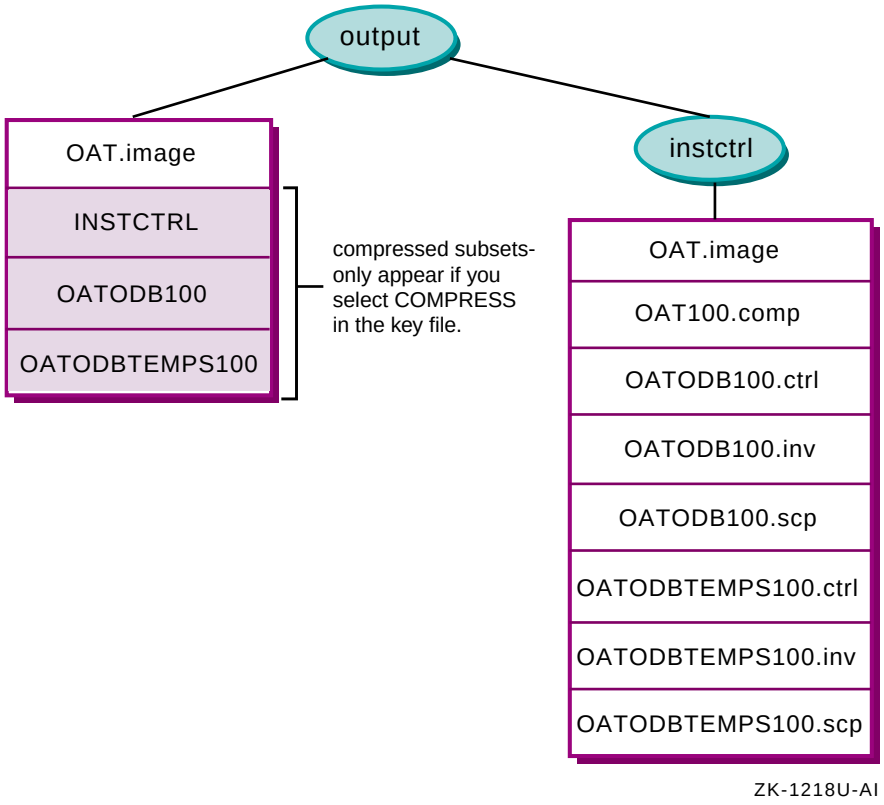
Table 3–9 shows the installation control files in the `instctrl` directory after you run the `kits` utility.

Table 3–9: Installation Control Files in the instctrl Directory

File	Description
product-id.comp	Compression flag file. This empty file is created only if you specified the COMPRESS attribute in the key file. Its presence signals to the `setld` utility that the subset files are compressed. The ODB kit's compression flag file is named `OAT100.comp`.
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`.
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`.
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`.
subset-id.scp	Subset control program. 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`.

Figure 3–3 shows the contents of the `output` directory after you run the `kits` utility.

Figure 3–3: 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 files.

3.5.1 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`. If the compression flag file exists, the `setld` utility knows that the subset files are compressed.

3.5.2 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 3–10 describes the image data file.

Table 3–10: 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] Refer to the `sum`(1) reference page.

Example 3–14 shows `OAT.image`, the image data file for the ODB kit:

Example 3–14: Sample Image Data File

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

3.5.3 Subset Control Files

The `setld` utility uses the subset control files as a source of descriptive information about subsets. Subset control file fields are described in Table 3–11.

Table 3–11: 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 3–11: 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 3–9). This value is calculated by the `kits` utility.
DEPS	Specifies either a list of subsets upon which this subset is dependent (`DEPS="OATODB100 OSFDCMT505"`), 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=uncompressed, 1=compressed). Bits 3 to 7 are reserved; bits 8 to 15 are undefined.

Example 3–15 shows OATODB100.ctrl, the control file for the ODB kit's OATODB100 subset:

Example 3–15: 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
```

3.5.4 Subset Inventory File

The subset inventory file describes each file in the subset, listing its size, checksum, permissions, and other information. The kits utility generates this information, which reflects 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 3–12 describes subset inventory file fields.

Each record of the inventory is composed of 12 fields, each separated by single Tab characters. Table 3–12 describes the contents of these fields.

Table 3–12: 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 during kit installation. 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.
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.
Type	A letter that describes the file:

Table 3–12: Subset Inventory File Field Descriptions (cont.)

Name	Description
	`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 3–16 shows the `OATODB100.inv` inventory file for the ODB kit's `OATODB100` subset.

Example 3–16: Sample Subset Inventory File

```
0 8192 00000 0 0 040755 5/13/00 100 d\
 ./opt/OAT100 none OATODB100
0 133 21616 0 0 100644 5/11/00 100 f\
 ./opt/OAT100/README.odb none OATODB100
0 8192 21616 0 0 040755 5/11/00 100 d\
 ./opt/OAT100/sbin  none OATODB100
0 151 28636 3 4 100755 5/11/00 100 f\
 ./opt/OAT100/sbin/odb_recover none OATODB100
0 8192 28636 0 0 040755 5/13/00 100 d\
 ./usr/opt/OAT100 none OATODB100
0 8192 28636 0 0 040755 5/11/00 100 d\
 ./usr/opt/OAT100/bin none OATODB100
0 75 26280 0 0 100755 5/11/00 100 f\
 ./usr/opt/OAT100/bin/odb_start none OATODB100
4  0     00000 0  0  100666   5/11/00  100  +\
   ./usr/var/cluster/members/member0/opt/OAT100/log_files/odb_log \
   none  OATODB100
0 8192 26280 0 0 040755 5/11/00 100 d\
 ./usr/var/opt/OAT100 none OATODB100
0 8192 26280 0 0 040755 5/11/00 100 d\
 ./usr/var/opt/OAT100/log_files none OATODB100
0  5  00000 0  0  100666  5/11/00  100  s\
   ./usr/var/opt/OAT100/log_files/odb_log \
   ../../../cluster/members/{memb}/opt/OAT100/log_files/odb_log
   OATODB100
```

Note

The backslashes (\) in Example 3–16 indicate line continuation and are not present in the actual file.

Fields are separated by single Tab characters.

3.6 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. The following sections describe this testing:

1. Loading all of the subsets onto a running system. (Section 3.6.1)
2. Repeating the subset loading test. (Section 3.6.2)
3. Deleting all of the subsets from a running system. (Section 3.6.3)
4. If your kit includes optional subsets, loading only the mandatory subsets onto a running system. (Section 3.6.4)
5. If your kit may run in a cluster environment, testing on a cluster. (Section 3.6.5)

Caution

It is important that you perform these tests in sequence.

3.6.1 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. Refer to Section 3.4.5 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. Refer to Section 3.4.5 for discussions of SCP tasks associated with these phases.

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

3.6.2 Repeating Subset Load

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. Refer to Section 3.4.5 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.

3.6.3 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 3.6.1 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. Refer to Section 3.4.5 for discussions of SCP tasks associated with these phases.

3.6.4 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. Refer to Section 3.4.5 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.

3.6.5 Testing in a Cluster

To test your product kit in 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. The following sections describe this testing:

1. Loading all of the subsets onto a cluster. (Section 3.6.5.1)
2. Deleting all of the subsets from a cluster. (Section 3.6.5.2)

Caution

It is important that you perform these tests in sequence.

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.

3.6.5.1 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. Refer to Section 3.4.5 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.

3.6.5.2 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 3.6.1 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. Refer to Section 3.4.5 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.

3.7 Updating Inventory After Creating Subsets

You may have to update the master inventory file after you have created subsets. For example, kernel product kits and hardware product kits require additional files, some of which must be added to your kit's inventory.

After you update the kit source directory, 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 3.5 to produce updated subsets and control files.

4

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 5.
- If you want to create a hardware product kit, go to Chapter 6.

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. Create subsets as described in Chapter 3. Subset control programs are optional for user product kits (Section 3.4).
4. Create the kit distribution media as described in Section 4.2.
5. Test the distribution media as described in Section 4.3.

No additional installation files are required for user product kits.

4.1 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.

4.2 Producing Distribution Media

After you have tested the subsets as described in Section 3.6, you can produce the distribution media. Distribution media production consists of the following tasks:

1. Edit the `/etc/kitcap` file. (Section 4.2.1)
2. Build the user product kit on the distribution media:
   - Use the `gendisk` utility to build a kit on disk media (Section 4.2.2)
   - Use the `gentapes` utility to build a `tar` format kit on magnetic tape (Section 4.2.3)

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

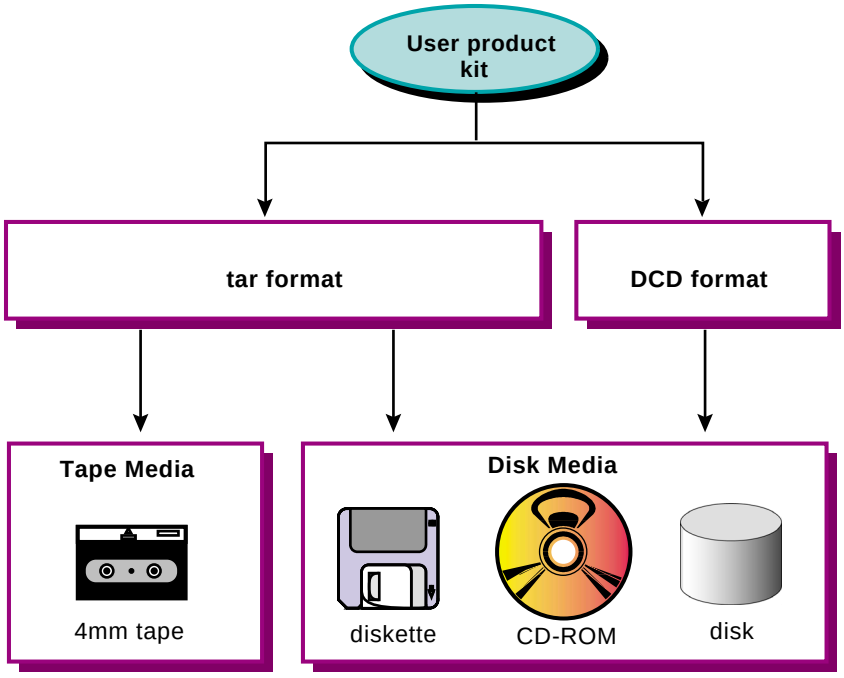
  CD-ROM media can support large kits or multiple kits on a single media. The kit is first produced on the hard disk, then written onto the CD-ROM. Use the `gendisk` utility to produce the master kit on hard disk. Follow the CD-ROM manufacturer's instructions for writing the kit onto the CD-ROM media.

- 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 4–1 shows the types of file formats and distribution media that are available for user product kits.

Figure 4–1: User Product Kit File Formats



4.2.1 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. Refer to the `gendisk`(1) reference page for additional 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. Refer to the `kitcap`(4) reference page for more information about the contents of the `/etc/kitcap` file.

4.2.1.1 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. Refer to 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.

Refer to the `kitcap`(4) reference page for more detailed information about the disk media record format.

Refer to Section 3.3 for information about the key file.

Example 4–1 shows the record to be added to the `/etc/kitcap` file to produce the ODB kit on disk media:

Example 4–1: 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 4–1, 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`.

4.2.1.2 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. Refer to 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.

Refer to the kitcap(4) reference page for more detailed information about the tape media record format.

Example 4–2 shows the record to be added to the /etc/kitcap file to produce the ODB kit on TK50 tapes:

Example 4–2: Sample Tape Media Descriptor

```
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 file in tar format and the names of the subsets that make up the kit: OATODB100 and OATODBTEMPS100.

4.2.2 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

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 -w 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. Refer to 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 4–3 creates a tar format user product kit for OAT100 on the c partition of dsk0:

Example 4–3: Sample gendisk Command

```
# gendisk OAT100 /dev/rdisk/dsk0c
```

Refer to the gendisk(1) reference page for more information about this utility.

4.2.3 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. Refer to 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 4–4 creates a tar format user product kit for OAT100 on the magnetic tape in /dev/ntape/dat:

Example 4–4: Sample gentapes Command

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

Refer to the gentapes(1) reference page for more information about this utility.

4.3 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 read the /opt/OAT100/README.odb 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.

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

5

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 4.
- If you want to create a hardware 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. Create subsets as described in Chapter 3. Kernel product kits require a subset control program (Section 3.4).
4. Create additional installation files as described in Section 5.2.
5. Rerun the `newinv` utility as described in Section 3.7 to update the master inventory file with the additional installation files.
6. Rerun the `kits` utility as described in Section 3.5 to produce updated subsets and control files.
7. Create the kit distribution media as described in Section 5.3.
8. Test the distribution media as described in Section 5.4.

5.1 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).

5.2 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:

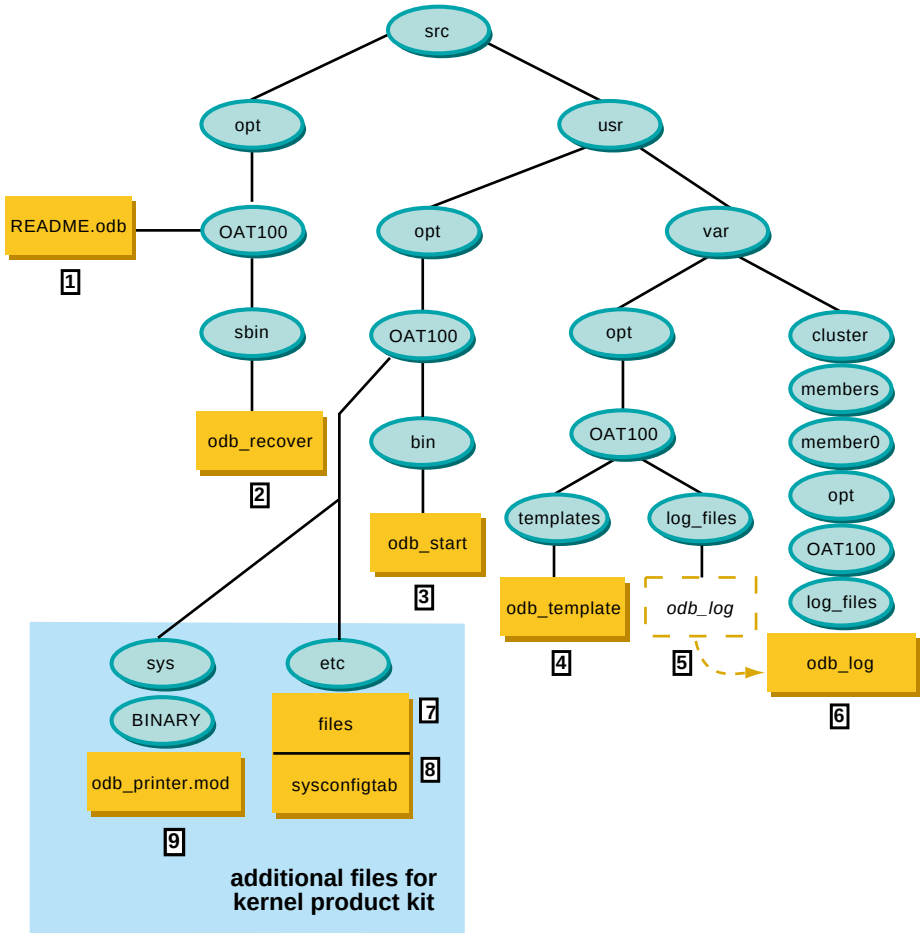
- A statically configured product is linked statically into the kernel at build (or bootlink) time and configured at boot time. A static product can be built from source files, binary objects, modules, or all three.
- A dynamically configured product is loaded into a running kernel after it has been 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, refer to *Writing Device Drivers*.

Figure 5–1 shows the files that make up the ODB product kit source directory. Additional kernel product files are highlighted.

Figure 5–1: Kernel Product Source Directory



ZK-1555U-AI

1 `/README.odb` — a text file containing general product information

This file is installed in the `root` (`/`) file system and is placed in the `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 tool 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/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.

5 /usr/var/log_files/odb_log — the context-dependent symbolic link (CDSL) for the odb_log file.

   This symbolic link is installed in the /usr/var file system and is placed in the usr/var/opt/OAT100/log_files source directory.

6 /usr/var/cluster/members/member0/log_files/odb_log — the file listing all of the ODB documents created by a user. The odb_log file can be modified by users.

   This file is installed in the /usr/var file system and is placed in the usr/var/cluster/members/member0/opt/OAT100/log_files source directory.

7 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.

8 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. The sysconfigtab file fragment gets appended to the /etc/sysconfigtab database when the kernel product kit is installed. For more information, refer to the sysconfigtab(4) reference page.

9 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 5.2.1)
- The sysconfigtab file fragment (Section 5.2.2)
- The *driver*.mod object module file (Section 5.2.3)
- The *.c (source) and *.h (header) files (Section 5.2.4)
- The *device*.mth method files (Section 5.2.5)

5.2.1 The files File Fragment

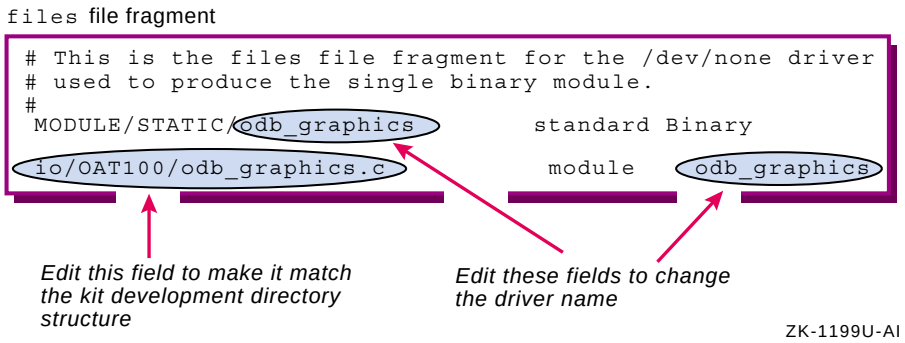
The `files` file fragment contains information about the location of the source code and modules associated with the driver, tags indicating when the driver is loaded into the kernel, and whether the source or binary form of the driver is supplied to the customer. You need to edit this file if the kit 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 5–2 shows which fields within the `files` file fragment need to change.

Figure 5–2: Editing the files File Fragment

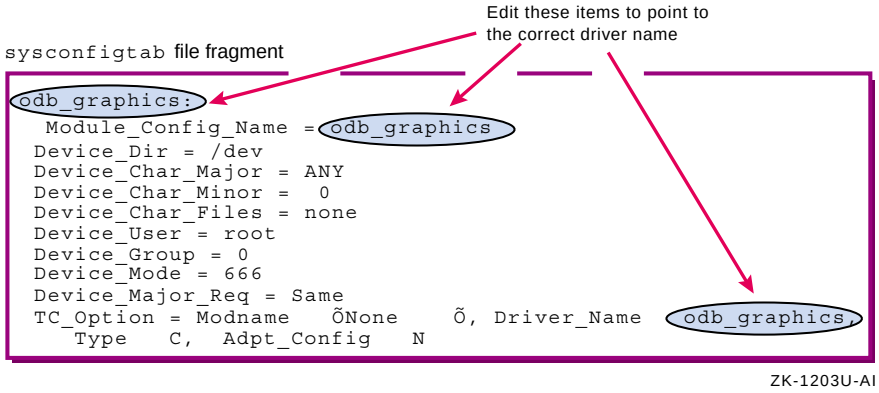


5.2.2 The sysconfigtab File Fragment

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 5–3. 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 5–3: Editing the sysconfigtab File Fragment



This file is described in the `sysconfigtab`(4) reference page.

5.2.3 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
```

5.2.4 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.

5.2.5 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.

5.3 Producing Distribution Media

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

1. Rerunning the newinv utility to update the master inventory file with the additional installation files (Section 3.7)
2. Rerunning the kits utility to produce updated subsets and control files (Section 3.5)
3. Editing the /etc/kitcap file. (Section 5.3.1)
4. Building the kernel product kit on the distribution media.
   - Use the gendisk utility to build a kit on disk media (Section 5.3.2)
   - Use the gentapes utility to build a tar format kit on magnetic tape (Section 5.3.3)

You can produce the kernel product kit in either tar format or direct CD-ROM (DCD) format.

- 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.
- 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.

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. During installation, the `setld` utility does the following:

  1. Creates a single file in `tar` format for the product files in each subset.
  2. Compresses each `tar`–format subset file.
  3. Uncompresses each `tar`–format subset file.
  4. Writes each `tar`–format subset file onto the target system.

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

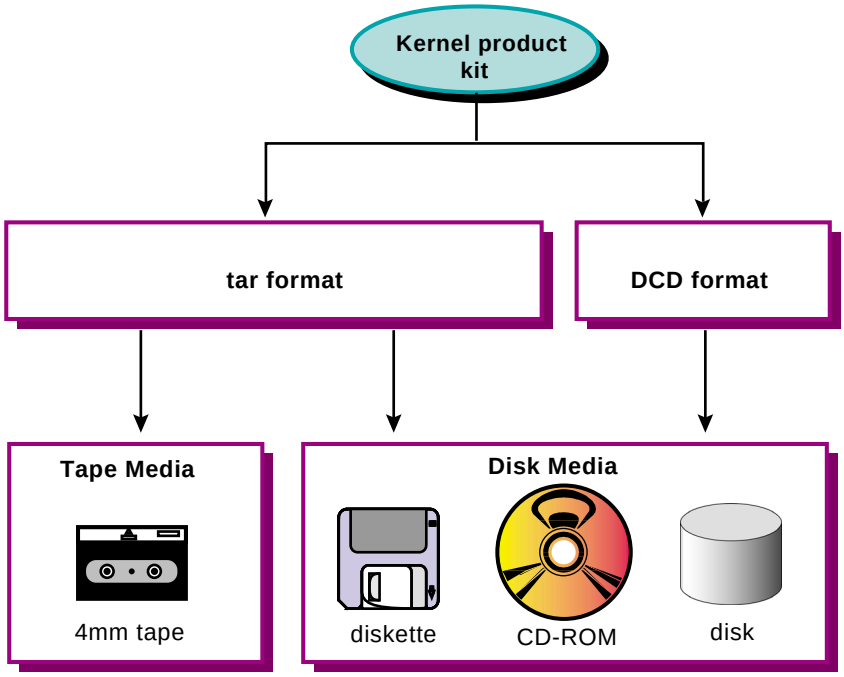
  CD-ROM media can support large kits or multiple kits on a single media. The kit is first produced on the hard disk, then written onto the CD-ROM. Use the `gendisk` utility to produce the master kit on hard disk. Follow the CD-ROM manufacturer's instructions for writing the kit onto the CD-ROM media.

- Magnetic tape

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

Figure 5–4 shows the types of file formats and distribution media that are available for kernel product kits.

Figure 5–4: Kernel Product Kit File Formats



5.3.1 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. Refer to the `gendisk`(1) reference page for additional 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. Refer to the `kitcap`(4) reference page for more information about the contents of the `/etc/kitcap` file.

5.3.1.1 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 may also 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.

Refer to the kitcap(4) reference page for more information about the disk media record format.

Refer to Section 3.3 for information about the key file.

Example 5–1 shows the record to be added to the /etc/kitcap file to produce the ODB kit on disk media:

Example 5–1: Sample Disk Media Descriptor for Kernel Product

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

Based on the information shown in Example 5–1, 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.

5.3.1.2 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.

Refer to the kitcap(4) reference page for more detailed information about the tape media record format.

Example 5–2 shows the record to be added to the /etc/kitcap file to produce the ODB kit on TK50 tapes:

Example 5–2: 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 directory and the names of the subsets that make up the kit: OATODB100, OATODBTEMPS100, and OATODBKERNEL100.

5.3.2 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. Refer to 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 illustrated in Example 5–3 creates a tar format kernel product kit for OAT100 on the c partition of dsk0:

Example 5–3: Sample gendisk Command

```
# gendisk OAT100 /dev/rdisk/dsk0c
```

Refer to the gendisk(1) reference page for more information about this utility.

5.3.3 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. Refer to 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 illustrated in Example 5–4 creates a tar format user product kit for OAT100 on the magnetic tape in /dev/ntape/dat:

Example 5–4: Sample gentapes Command

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

Refer to the gentapes(1) reference page for more information about this utility.

5.4 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 5.4.1)
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 start a Full Installation on the client system. (Section 5.4.2)

The following sections describe how to set up and perform these tests.

5.4.1 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
   ```

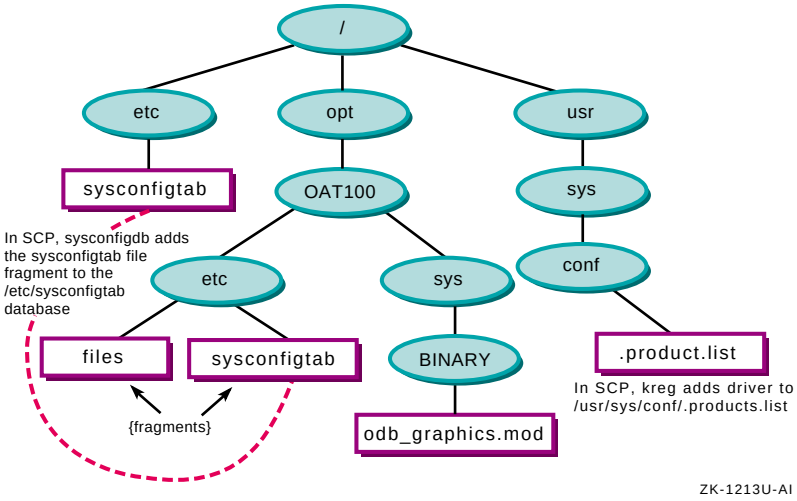
5. Use the `setld` utility to install the kernel product subsets:

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

*** 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 19 14:49:25 EDT 2000

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 5–5 shows the steps the subset control program takes to statically configure the driver.

Figure 5–5: Static Configuration of a Driver



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. You see output similar to the following:

```
# setld -d OATODB100 OATODBTEMPS100 OATODBKERNEL100

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 19 14:55:31 EDT 2000

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. Restart the system to ensure 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.

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

5.4.2 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. Refer to the *Sharing Software on a Local Area Network* manual for additional 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:

    ```
    Checking accessibility of RIS areas.... done

    *** 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:

```
Checking accessibility of RIS areas.... done

*** 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:
```

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*.

Refer to the *Installation Guide* and the `setld`(8) reference page for more information on using the `setld` utility to install layered products.

6

Producing Hardware Product Kits

This chapter describes how to produce a hardware product kit, what additional files are required, and how to test the kit installation.

Note

The information in this chapter describes how to produce hardware product kits.

- If you want to create a user product kit, go to Chapter 4.
- If you want to create a kernel product kit, go to Chapter 5.
- Files for hardware product kits are release-specific. If you want to create a hardware product for a previous release of the operating system, use the instructions in that release's *Guide to Preparing Product Kits*.

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

1. Read Chapter 1 for an overview of product kits.
2. Design the kit directory structure as described in Chapter 2.
3. Create subsets as described in Chapter 3. Hardware product kits require a subset control program (Section 3.4).
4. Create additional installation files as described in Section 6.2.
5. Rerun the `newinv` utility as described in Section 3.7 to update the master inventory file with the additional installation files.
6. Rerun the `kits` utility as described in Section 3.5 to produce updated subsets and control files.
7. Create the kit distribution media as described in Section 6.3.
8. Test the distribution media as described in Section 6.4.

6.1 Overview

A hardware product includes the kernel modules that let your operating system support new or upgraded hardware, letting you install hardware support without reinstalling or updating your base operating system. However, you must reboot your system to rebuild the kernel so that it includes the modules that support your new hardware. To support the new hardware and the hardware product kit software, the customer may need to install or update to a version of the operating system that supports hardware product kits before installing the hardware product kit software.

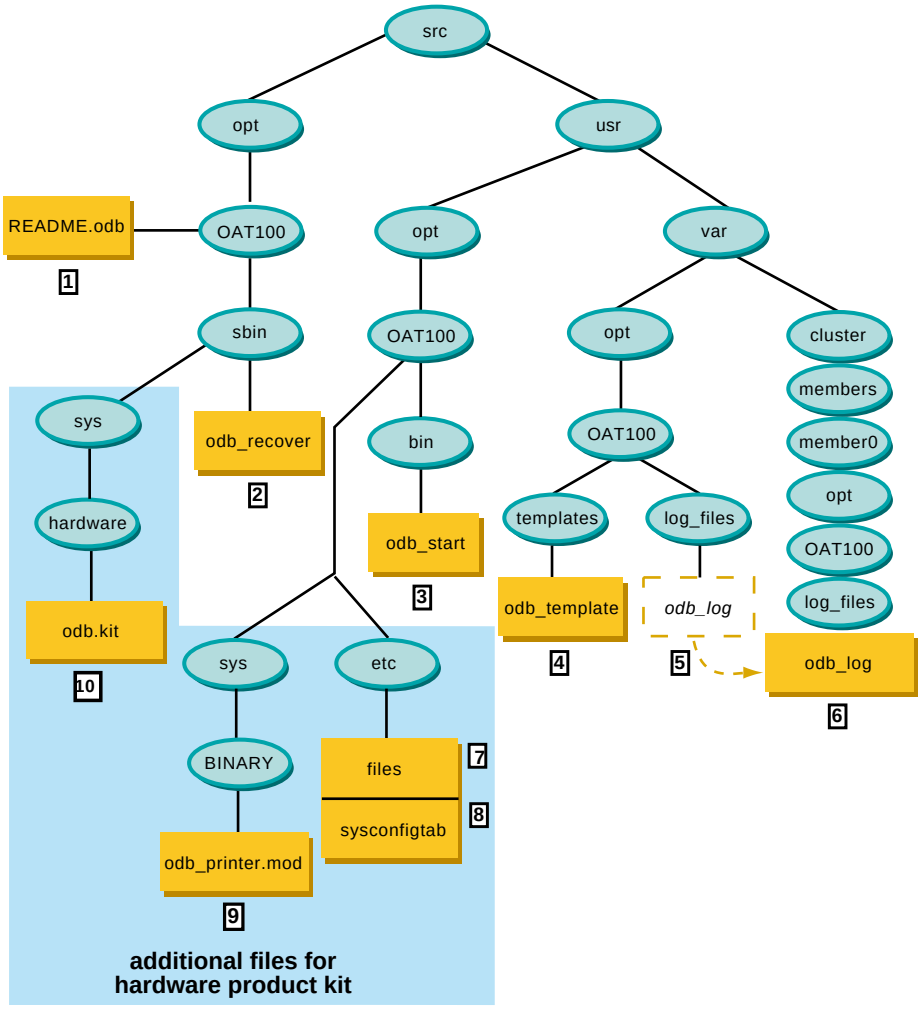
The kernel modules and the kit support files are distributed on CD-ROM as part of the hardware product kit, and can be installed either directly from the distribution media or loaded onto a Remote Installation Services (RIS) area for installation by RIS clients over a local area network (LAN).

Hardware product kits can be delivered by a process called new hardware distribution (NHD). The kit usually is provided on CD-ROM, and includes installation and testing instructions.

6.2 Creating Additional Installation Files

Figure 6–1 shows the files that make up the ODB product kit source directory. Additional hardware product files are highlighted.

Figure 6–1: Hardware Product Source Directory



ZK-1200U-AI

1 `/README.odb` — a text file containing general product information

This file is installed in the `root` (`/`) file system and is placed in the `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 tool 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/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.

5 /usr/var/log_files/odb_log — the context-dependent symbolic link (CDSL) for the odb_log file.

This symbolic link is installed in the /usr/var file system and is placed in the usr/var/opt/OAT100/log_files source directory.

6 /usr/var/cluster/members/member0/log_files/odb_log — the file listing all of the ODB documents created by a user. The odb_log file can be modified by users.

This file is installed in the /usr/var file system and is placed in the usr/var/cluster/members/member0/opt/OAT100/log_files source directory.

7 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.

8 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. The sysconfigtab file fragment gets appended to the /etc/sysconfigtab database when the kernel product kit is installed. For more information, refer to the sysconfigtab(4) reference page.

9 odb_printer.mod — the object module file containing the single binary module for the ODB kernel product.

10 The odb.kit files allow the target system's kernel to bootlink the new hardware support software modules. One stays resident on the distribution media, and one is installed onto the target system. Although their format is the same, module specifications differ within the two files. Section 6.2.6 provides more information about the *name*.kit files.

A hardware product kit requires that you include the following files on the distribution media to make the hardware product accessible during initial system installation and console-level reboots:

- The `files` file fragment (Section 6.2.1)
- The `sysconfigtab` file fragment (Section 6.2.2)
- The *driver*`.mod` object module file (Section 6.2.3)
- The `*.c` (source) and `*.h` (header) files (Section 6.2.4)
- The *device*`.mth` method files (Section 6.2.5)
- The *name*`.kit` files (Section 6.2.6)
- The `database.nhd` file (Section 6.2.7)
- The *PRODCODE*`.root` file (Section 6.2.8)
- The *kitname*`.kk` file (Section 6.2.9)

6.2.1 The files File Fragment

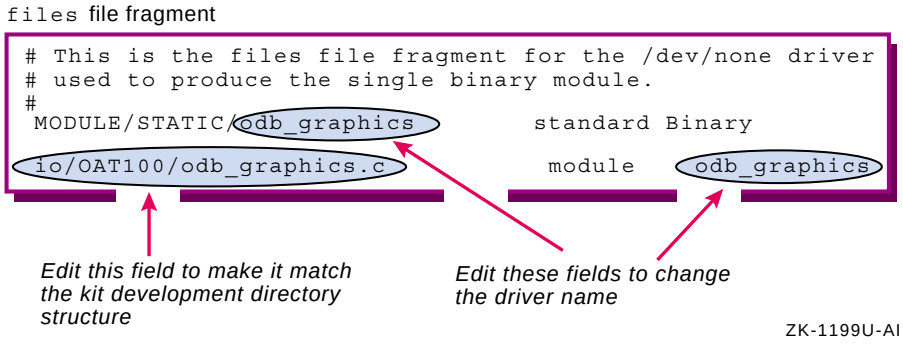
The `files` file fragment contains information about the location of the source code and modules associated with the driver, tags indicating when the driver is loaded into the kernel, and whether the source or binary form of the driver is supplied to the customer. You need to edit this file if the kit 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 6–2 shows which fields within the `files` file fragment need to change.

Figure 6–2: Editing the files File Fragment

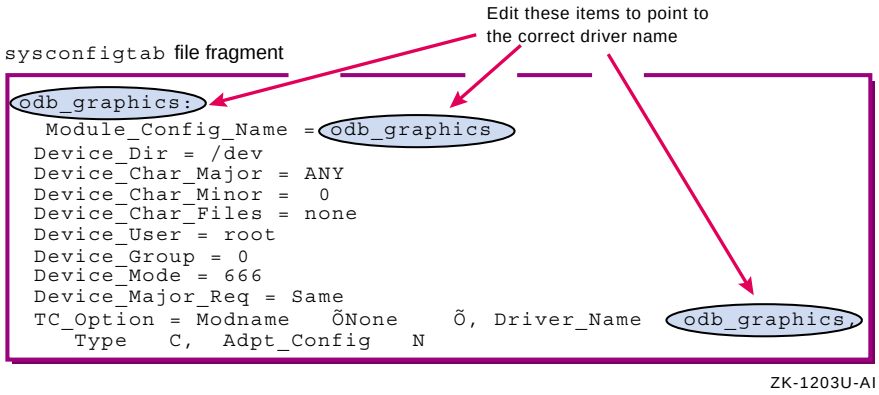


6.2.2 The sysconfigtab File Fragment

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 6–3. 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 6–3: Editing the sysconfigtab File Fragment



This file is described in the `sysconfigtab`(4) reference page.

6.2.3 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

Compressed modules cannot be linked using RIS. 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
```

6.2.4 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 configured statically and distributed in source form.

6.2.5 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.

6.2.6 The name.kit Files

The *name*.kit files allow the target system's kernel to bootlink the new hardware support software modules. The format of the distribution *name*.kit file and the installed *name*.kit files is the same, but the following list describes the differences:

- In the distribution *name*.kit file, the modules are specified by their location on the distribution media. The distribution *name*.kit file must ship in the /usr/opt/*PRODCODE*/sys/hardware directory.

  The distribution *name*.kit file is not part of the installed product and does not need to be part of the product inventory.

- In the installed *name*.kit file, the modules are specified by their location on the installed system. The installed *name*.kit file resides in the root file system and must ship under the /opt directory.

  The installed *name*.kit file is a required part of the installed product and must be part of the product inventory.

Commands in the distribution *name*.kit file direct the boot utility to modify the bootlink process and boot this kit off the distribution media. When the kernel is bootlinked, the distribution *name*.kit file controls where hardware product modules are found during the boot process. When bootlinking from a hardware product kit, the boot utility sets the default directory to the distribution media's root (/) directory. Commands in the distribution *name*.kit file indicate which modules should be added, removed, or replaced in the kernel.

Kernel modules must reside in the /opt/*PRODCODE*/sys/BINARY directory so that they can be accessed by the boot process. Kernel modules for the fictitious graphics device driver used in this chapter are located in /opt/OAT100/sys/BINARY.

Caution

Module files for hardware product kits must be compressed with the objZ utility. Do not use the compress or gzip utilities to compress module files for hardware product kits. See the objZ(1) reference page for more information.

Example 6–1 shows the format of the commands in the *name*.kit file.

Example 6–1: Sample Installed name.kit File

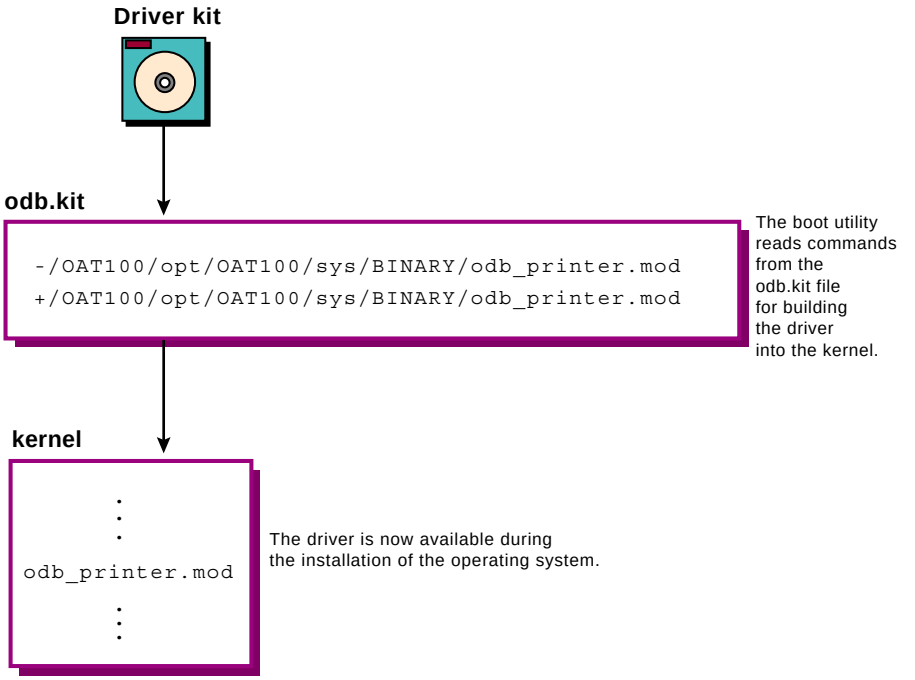
```
-/opt/OAT100/sys/BINARY/odb_printer.mod
+/opt/OAT100/sys/BINARY/odb_printer.mod
```

Each kernel module listed in the *name*.kit file first must be removed with the - operator and then added with the + operator. If kernel modules are just added and you already have an older version of the hardware product on the system, the bootlink will fail because the kernel module already exists. The + operator does not replace a module if it already exists. Therefore, to protect against this, it is recommended to first remove the module and then add it to make sure you get the latest version of the module in the kernel.

The hardware product kit for the odb_printer driver supplies a *name*.kit file, which first removes and then adds the single binary module odb_printer.mod to the kernel. In Example 6–1, the sample *name*.kit file is referencing the /opt/OAT100/sys/BINARY directory on the installed system, not on the distribution media.

Figure 6–4 shows how the distribution *name*.kit file works with boot utility software during the installation of a hardware product kit.

Figure 6–4: Using the Distribution name.kit File During Installation



In the figure, the kit contains a *name*.kit file called odb.kit. The odb.kit file instructs the system's boot utility to build and configure a temporary kernel that includes the odb_printer driver. Upon completion, this temporary kernel makes the odb_printer driver available to handle user and system requests of a specific hardware device during the installation of the operating system.

6.2.7 The database.nhd File

The database.nhd file, also known as the hardware database file, describes the hardware product kits residing on the distribution media. The installation process uses the database.nhd file to determine what software to load. This file contains one record for each hardware product kernel module. For example, if your distribution includes two hardware product kits, one with three kernel modules and another with five kernel modules, the database.nhd file would contain eight records.

There can be only one database.nhd file on each distribution media, located in the root (/) directory.

The syntax for the database.nhd file uses the following conventions:

- Lines beginning with a pound sign (#) are comments and are ignored. Blank lines are also ignored.
- Each entry must be enclosed in { } (curly braces).
- Each field in an entry must be enclosed in { } (curly braces).
- Entries and fields can span lines without the use of a line continuation character.

An entry in the database.nhd file contains eight fields:

- Vendor name
- Hardware name
- List of subsets to be loaded, separated by spaces
- List of releases under which the product is supported, separated by spaces, without V preceding the version numbers
- Kernel build flag

  Valid values are 0 for no kernel build required, and 1 for kernel build required
- Module name including the full path as specified in the *name*.kit file
- Kit directory, including the full path as located on the distribution media
- Kit file name, including the full path as located on the distribution media

Each vendor name and hardware name must form a unique combination. Each product release supported by the kit must map to at least one release of the operating system, as shown in Example 6–2. Operating system versions use the format: {*MajorNumber*}{*MinorNumber*}[*VariantLetter*].

Example 6–2: Sample database.nhd File

```
#
# New hardware support
#
{
    {Orpheus Document Builders}
    {ODB Device Driver}
    {OATBASE100 OATDOC100}
    {4.0F}
    {1}
    {/V4.0F/sys/BINARY/asynchhw_stub.mod}
    {/V4.0F/kit}
    {/v4.0F/usr/sys/hardware/base.kit}
}
```

6.2.8 The .root File

When building a Direct CD-ROM (DCD) distribution media, the `gendisk` utility also generates a *PRODCODE*`.root` file, where *PRODCODE* is the three-letter product code and the version number, for example: `OAT100.root`.

This file must ship in the `instctrl` directory and contains the path to the expanded subset files starting from the `kit` directory (for example, `/mnt/ODB/kit`).

This file is created automatically by the `gendisk` utility when a Direct CD-ROM (DCD) distribution media is built.

6.2.9 The kitname.kk File

By default, all hardware product kits should contain a hardware product indicator file (*kitname*`.kk`) in the `instctrl` directory. The file can be empty, but it must exist. This file indicates to RIS that a hardware product kit exists on the distribution media. When the `ris` utility finds this file, you are prompted for a hardware product kit name to add to the RIS area. This file is created automatically by the `gendisk` utility when it finds the `kk=true` option in the `/etc/kitcap` file.

6.3 Producing Distribution Media

After you have created the subsets and the additional installation files, you can move the subsets onto the distribution media. This consists of the following tasks:

1. Rerunning the `newinv` utility to update the master inventory file with the additional installation files. (Section 3.7)
2. Rerunning the `kits` utility to produce updated subsets and control files. (Section 3.5)
3. Editing the `/etc/kitcap` file. (Section 6.3.1)
4. Using the `gendisk` utility to build the hardware product kit on disk media. (Section 6.3.2)

6.3.1 Editing the /etc/kitcap File

The gendisk utility refers 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. Refer to the gendisk(1) reference page for additional 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 the record can depend on the product type you are delivering. For example, the kitcap record for a hardware product must contain the kk=true parameter and may require the use of the rootdd= option.

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.
  - 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 may also include these parameters:

– `kk=true`

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

– `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.

Refer to the `kitcap`(4) reference page for more information about the disk media record format.

Refer to Section 3.3 for information about the key file.

Example 6–3 shows the record to be added to the `/etc/kitcap` file to produce the ODB hardware product kit. Notice the use of the `kk=true` and `rootdd=` options.

Example 6–3: Sample Disk Media Descriptor

```
OAT100HD:c:\
  dd=/ODB/kit,kk=true,rootdd=..:Orpheus_Document_Builder:\
  /mykit/output:instctrl:OATBASE100
```

Example 6–4 shows a sample kitcap record for a CD-ROM with multiple hardware products. In Example 6–4, the rootdd=. . entry overrides the default and places the expanded subset files in the product-specific directory of /ODB. By default, expanded subset files (in DCD format) are placed at the top of the media's file system.

Example 6–4: Sample Disk Media Descriptor for Multiple Hardware Product Kits

```
OAT100HD:c:\
  dd=/OAT100/kit,kk=true,rootdd=..:ODB_Driver_V1:\
  /odb100/output:instctrl:OATBASE100:\
  dd=/OAT200/kit,kk=true,rootdd=..:ODB_Driver_V2:\
  /odb200/output:instctrl:OATBASE200:
```

Refer to the kitcap(4) reference page for more information about the disk media descriptor format.

Note

Each kit must be contained in a separate directory located in the distribution media's root directory. For example, a kit is located in /mnt/ODBprinter where /mnt is the mount point and ODBprinter is the subdirectory under which the kit files are located.

6.3.2 Building a Hardware Product Kit on Disk Media

When the product subsets are located in the output area of the kit directory structure, you produce a master copy of the hardware product kit on a hard disk and use this master disk to copy the kit onto a CD–ROM.

Follow this sample procedure to produce a master disk for the OAT100 hardware product kit:

1. Log in as root or use the su command to gain superuser privileges.
2. Make sure that you have an entry for your system's root account in the /.rhosts file. For example, if your system name is system01, your entry in the /.rhosts file would look similar to the following:

   ```
   system01 root
   ```

   If this entry is not present, the gendisk utility will fail with a Permission denied error.

3. Follow these steps to prepare the master disk for the hardware product kit:

   a. Determine the device where the master media is to be written. This sample procedure uses an RZ26 disk associated with the `/dev/disk/dsk1` device special file.

   b. Use the `disklabel` command to erase any existing label or data. For example:

      ```
      # disklabel -z /dev/disk/dsk1
      ```

   c. Use the `disklabel` command to write a new disk label. For example:

      ```
      # disklabel -wr /dev/disk/dsk1 rz26L
      ```

4. Use the `gendisk` utility to create the master disk for the hardware product kit.

   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. Refer to 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.

In this sample procedure, the system name is system01. For example:

```
# gendisk -d OAT100 /dev/disk/dsk1a
Generating OAT100 Kit from system01 on /dev/disk/dsk1a

WARNING:  this will remove any information stored in
/dev/disk/dsk1a
Are you sure you want to do this? (y/n): y
Do you want to clean the entire disk first? (y/n): n
```

Caution

When the gendisk utility asks if you want to clean the disk, always answer n or the gendisk replaces the current disk label with a default label.

The gendisk utility output is similar to the following:

```
Preparing /dev/disk/dsk1a
done.

Checking /dev/disk/dsk1a
/sbin/usf_fdck /dev/disk/dsk1a
** /dev/disk/dsk1a
File system unmounted cleanly - no fsck needed

Mounting /dev/disk/dsk1a on /usr/tmp/cd_mdt8344

Writing Images (dd=/).

 Image instctrl...done.
 Image OATBASE100...done.

Verifying Images (dd=/).

 Image instctrl...done.
 Image OATBASE100...done.

Kit OAT100 done.

Cleaning up working directories.
Unmounting /dev/disk/dsk1a
```

5. Follow these steps to copy the additional installation files to the master disk:

   a. Mount the master copy's hard disk. For example:

      ```
      # mount /dev/disk/dsk1a /mnt
      # cd /mnt
      ```

   b. Verify that the .kk and .root files exist in the master disk's instctrl directory. For example:

      ```
      # cd ./ODB/kit/instctrl
      # ls *.kk *.root
      OAT100.kk
      OAT100.root
      ```

c. Copy the `database.nhd` file to the master disk's root area. For example:

```
# cp /mykit/database.nhd /mnt
```

d. If it was not included in the inventory, copy the distribution *name*`.kit` file to the proper directory on the master disk. For example:

```
# cp /mykit/odb.kit \
    /mnt/ODB/usr/opt/OAT100/sys/hardware/odb.kit
```

e. Copy the distribution *name*`.kit` file and the `database.nhd` file from the kit-building area to the master disk. For example:

```
# cp /kitarea/odb.kit \
    /mnt/ODB/usr/opt/OAT100/sys/hardware/odb.kit
# cp /kitarea/database.nhd /mnt/database.nhd
```

Note

The backslash (\) in this example represents line continuation. Do not include it in the command line.

f. Unmount the disk. For example:

```
# umount /mnt
```

6. Use this master disk to copy the hardware product kit onto a CD–ROM by following the instructions that came with your CD–ROM burner.

6.4 Testing the Distribution Media

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

Run the following tests, in sequence, to test a hardware 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 6.4.1)
2. Use the bootlink process to test that all of the modules are in the correct locations to allow the system to add support to the bootlinked kernel. (Section 6.4.2)

3. Use the `hw_check` utility to verify that the `database.nhd` file is in the correct location and that it contains the correct information. (Section 6.4.3)
4. Use the `ris` utility to add a hardware product kit into a RIS area to verify that the correct files are present on the kit. (Section 6.4.4)
5. Register a client for the RIS area containing the hardware product kit and start a Full Installation on the client system. (Section 6.4.5)

When testing a hardware product kit in DCD format, reference the kit media at its mount point. For example, if you decide to use a spare disk for creating a media master area, you must reference your kit to that devices's mount point.

6.4.1 Testing a Hardware Product Kit with the setld Utility

To test a hardware product kit using the `setld` utility, the system where you conduct the test must be running the same version of the operating system for which the hardware product kit was built.

Follow these steps to test the hardware product kit using the `setld` utility. The purpose of this test is so that you can test the location of the installed product files as you defined them in the master inventory file.

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

   # **mount -r /dev/disk/cdrom0c /mnt**
5. Use the following `setld` command syntax to install each subset on the distribution media:

   # **setld -l /mnt/OAT100/kit OATBASE100**

```
Checking file system space required to install specified subsets:

File system space checked OK.

1 subset(s) will be installed.

Loading 1 of 1 subset(s)...

ODB Device Driver Files
  Copying from /mnt/ODB/kit (disk)
  Verifying
```

```
1 of 1 subset(s) installed successfully.

Configuring "ODB Device Driver Files " (OATBASE100)
```

If there is an error during subset installation, the `setld` utility displays a message explaining the error. Details for verification errors are in the `/var/adm/smlogs/fverify.log` file.

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

   ```
   # umount /mnt
   ```

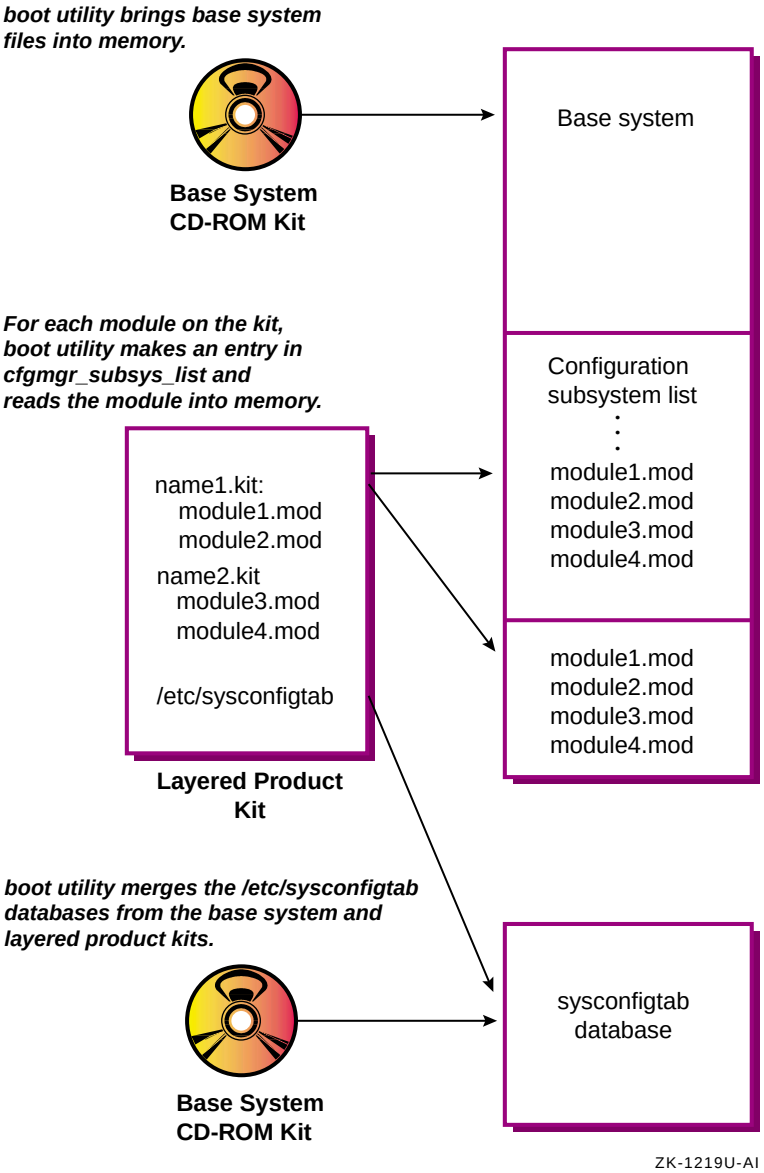
7. After the `setld` utility has finished installing and configuring each subset, verify that the files for each subset (including the `/usr/sys/hardware/database.nhd` files) have been installed in the correct locations.

6.4.2 Testing a Hardware Product Kit on a Running System

This section contains procedures for testing the installation of a hardware product kit on a system that already is running a version of the operating system that supports this feature.

Figure 6–5 shows how the `boot` utility builds a kernel to include a hardware product kit.

Figure 6–5: Bootlinking with a Hardware Product Kit



Follow these steps to use the bootlink process to install a hardware product kit onto a system that is running a version of the operating system that supports this feature:

1. Log in as the user `root` or use the `su` command to gain superuser privileges.
2. Back up the operating system.
3. Use the `shutdown` command to halt your system and bring it down to console mode:

   ```
   # shutdown -h now
   ```

4. Enter the following command to turn off automatic reboots at the console prompt:

   ```
   >>> set auto_action halt
   ```

5. Power down your system.
6. Install the new hardware.
7. Power up your system. You see the console prompt:

   ```
   >>>
   ```

8. Enter the following `boot` command:

   ```
   >>> boot -flag fa -file "/GENERIC" sys_disk
   ```

   In the previous example:

   - The `-flag fa` defines the boot flags: `f` for a hardware product kit and `a` for multiuser mode.
   - The `-file "/GENERIC"` tells the kernel to bootlink using the file `/GENERIC`.
   - The optional *sys_disk* argument is the console device name of the system disk of the system that is running. You only need this argument if your `bootdef_dev` console variable is not set to your running system disk.

   You see the following prompt:

   ```
   Enter Device Name:
   ```

9. Enter the CD-ROM console device name, for example, DKA500. You see a prompt similar to the following:

   ```
   Insert media for kit 'device:hwkitname', press Return when ready:
   ```

10. Insert the hardware product kit CD-ROM and press Return. You see the following prompt:

    ```
    Enter Kit Name:
    ```

11. Enter the name of the hardware product kit you want to install, using the full pathname of the distribution *name*.kit file on the distribution media. You see the following prompt:

    ```
    Insert media for kit 'device:hwkitname', press Return when ready:
    ```

    In this example, *device*:*hwkitname* is the device name you entered in Step 9 and the hardware product kit name you entered in Step 11

12. Insert the CD-ROM containing the hardware product kit into the CD-ROM drive and press Return. The boot utility reads the selected hardware product kit file into memory, and you see the following prompt:

    ```
    Enter Kit Name:
    ```

13. Take one of the following actions:

    - If you are installing another hardware product kit from the same CD-ROM, enter the kit name, press Return, and return to Step 12.
    - If you are not installing another hardware product kit from the same CD-ROM just press Return and continue.

      You see the following prompt:

      ```
      Enter Device Name:
      ```

14. Take one of the following actions:

    - If you are installing another hardware product kit from a different CD-ROM, enter the CD-ROM device name, press Return, and go back to Step 11.
    - If you are not installing another hardware product kit, just press Return and continue.

      You see the following prompt:

      ```
      Insert boot media, hit <return> when ready:
      ```

15. Press Return, because you are adding hardware support to a running system and the system disk is your boot media.

    The `boot` utility reads the generic kernel modules and links the kernel objects, and you see the following prompt:

    ```
    Insert media for kit 'dev_name:hwkitname', press Return when ready:
    ```

    In the previous prompt, *dev_name* is the CD-ROM device name you entered in Step 9 and *hwkitname* is the hardware product kit name that you entered in Step 11.

16. Insert the CD-ROM into the drive and press Return.

    Note

    Do this step for every device and every kit name entered in Step 9 through Step 14.

    You see the following prompt:

    ```
    Insert boot media, press Return when ready:
    ```

    Press Return, because you are adding hardware support to a running system and the system disk is your boot media. The `boot` utility loads and configures the hardware product kit.

    Note

    If you installed more than one hardware product kit from different media on the same device, the `boot` utility may prompt you for the location of some of the hardware support subsets. If you see this prompt, load the appropriate CD-ROM into the CD-ROM drive and enter the appropriate console device name.

    You see the following prompt:

    ```
    Enter a name for the kernel configuration file. [SYSNAME]:
    ```

    SYSNAME is the name of your existing kernel configuration file, usually your system name in upper case characters.

17. Enter the name of the kernel configuration file or press Return to accept the default.

    You may see one or more of the following sequences:

    - If you select the existing kernel configuration file name, you are asked to confirm your selection. If you confirm your selection of the existing file name, the old kernel configuration file is backed up to *SYSNAME*.bck.
    - If the boot utility prompts you to rebuild the kernel, you see a prompt similar to the following:

```
*** KERNEL OPTION SELECTION ***

    Selection   Kernel Option
-------------------------------------------------------------
        1       System V Devices
        2       Logical Volume Manager (LVM)
        3       NTP V3 Kernel Phase Lock Loop (NTP_TIME)
        4       Kernel Breakpoint Debugger (KDEBUG)
        5       Packetfilter driver (PACKETFILTER)
        6       Point-to-Point Protocol (PPP)
        7       STREAMS pckt module (PCKT)
        8       Data Link Bridge (DLPI V2.0 Service Class 1)
        9       X/Open Transport Interface (XTISO, TIMOD, TIRDWR)
        10      ISO 9660 Compact Disc File System (CDFS)
        11      Audit Subsystem
        12      ACL Subsystem
        13      Logical Storage Manager (LSM)
        14      Advanced File System (ADVFS)
        15      All of the above
        16      None of the above
        17      Help
        18      Display all options again
-------------------------------------------------------------

Enter the selection number for each kernel option you want.
For example, 1 3 [16]:
```

    The options you see depend upon the software subsets that you have installed. See the *Installation Guide* for information about selecting kernel options and the doconfig(8) reference page for information about the kernel build process.

    a. Select kernel options from the list and press Return. You see a prompt similar to the following:

```
You selected the following kernel options:

 System V Devices
 Logical Volume Manager (LVM)
 NTP V3 Kernel Phase Lock Loop (NTP_TIME)
 Kernel Breakpoint Debugger (KDEBUG)
 Packetfilter driver (PACKETFILTER)
 Point-to-Point Protocol (PPP)
 STREAMS pckt module (PCKT)
 Data Link Bridge (DLPI V2.0 Service Class 1)
 X/Open Transport Interface (XTISO, TIMOD, TIRDWR)
 ISO 9660 Compact Disc File System (CDFS)
 Audit Subsystem
```

```
ACL Subsystem
Logical Storage Manager (LSM)
Advanced File System (ADVFS)

Is that correct? (y/n) [y]:
```

b. Do one of the following:

   – If the list is correct, enter **y** and continue to the next step.

   – If the list is not correct, enter **n** to return to Step 17a and select kernel options again.

   The `boot` utility asks if you want to edit the `/usr/sys/conf/`*SYSNAME* kernel configuration file. For information about editing this file, refer to the *Installation Guide*. Usually, there is no reason to edit this file.

The `boot` utility rebuilds your operating system kernel and reboots with the new kernel. After a successful reboot, you see the operating system login window.

18. Log in as `root`.
19. Use the `setld -i` command to verify that your hardware product kit is installed.
20. Check to make sure that the installed files are where you want them to be and that the hardware product is operational.
21. Check to make sure that the `/GENERIC` file was rebuilt correctly by issuing the following command for each module file that was loaded:

    # **cat /GENERIC | grep -e *modname*.mod**

    In the previous example, *modname*`.mod` represents the name of the module file that was loaded. If the `.mod` file was supplied in the `/opt` directory, the full path and file name should be in the `/GENERIC` file.

6.4.3 Using the hw_check Utility to Test a Hardware Product Kit

When you perform an Update Installation, the process checks for any installed hardware product kits automatically and reports the following information:

- If an existing hardware product kit will continue to operate correctly with the new version of the operating system.
- If an existing hardware product kit is integrated into the new version of the operating system. If so, the hardware product kit will be removed as part of the Update Installation and will be replaced by the functionality shipped with the operating system.
- If an existing kit is not supported in the new version of the operating system.

This section describes two test cases to test the validity of a hardware product kit during an Update Installation.

Note

The following test assumes that you already have done the bootlink test described in Section 6.4.2.

Follow these steps to test the format of the database.nhd files:

1. Log in as root to the same system where you performed the boot link test.
2. Mount the operating system CD-ROM from multiuser mode:

   ```
   # mount -r /dev/disk/dsk9a /mnt
   ```

3. Run the hw_check utility:

   ```
   # /usr/lbin/hw_check /mnt
   ```

   You see output similar to the following for a successful test:

   ```
   Checking for installed supplemental hardware support...

   The following hardware was installed using a supplemental
   hardware kit and will continue to work under the new
   operating system without any modifications.

           ODB Device Driver

   Press RETURN to continue...
   ```

   If you do not see a message similar to this, the most likely cause of the error is an incorrect value in the supported release field of the /usr/sys/hardware/database.nhd file on the system. This file is generated or updated from the database.nhd file on the distribution media. Refer to Section 6.2.7 for information about the format of this file.

Next, follow these steps to run the hw_check utility, testing the kit as if you were performing an Update Installation. This test sets up a scenario where an existing kit is not supported in the new release of the operating system. This procedure assumes that you already have tested the kit with the bootlink utility as described in Section 6.4.2.

1. Log in as root to the same system where you tested the kit with the setld utility.
2. Use the editor of your choice to modify the installed system's /usr/sys/hardware/database.nhd file and change the value in the supported release field to XXX. For example, if this is the original installed database.nhd file:

```
{
    {Orpheus Document Builders}
    {ODB Device Driver}
    {OATBASE100 OATDOC100}
    {4.0F}
    {1}
    {/V4.0F/sys/BINARY/asynchhw_stub.mod}
    {/V4.0F/kit}
    {/V4.0F/usr/sys/hardware/base.kit}
}
```

   Change the supported release field as shown in this example:

```
{
    {Orpheus Document Builders}
    {ODB Device Driver}
    {OATBASE100 OATDOC100}
    {XXX}
    {1}
    {/V4.0F/sys/BINARY/asynchhw_stub.mod}
    {/V4.0F/kit}
    {/V4.0F/usr/sys/hardware/base.kit}
}
```

   Changing the supported release field to XXX indicates that the hardware product kit is supported only under operating system version XXX. Changing this field should trigger the appropriate warning logic in the Update Installation process because XXX is not a valid release format.

3. Run the hw_check utility to start the test:

```
# /usr/lbin/hw_check /mnt
```

You see output similar to the following:

```
Checking for installed hardware product kits...

The update installation has detected that the hardware
product kit listed below is loaded on your system and
is not supported in the new release of the operating
system (V5.0A). In order for the update to
complete successfully, you must provide the distribution
media that contains the V5.0A version of the hardware
product listed below.  The update install process will
verify that the media you provide contains
the correct software.

      Orpheus Document Builders ODB Device Driver

Enter kit location (e.g.: /dev/disk/dsk3c or /mnt)
```

4. Enter the kit location, for example **/mnt**, and press the Return key.

The output is similar to the following:

```
The kit located at /mnt contains support for hardware
that is currently installed on your system and is not
supported under the new version of the operating
system V5.0A.  In order for your hardware
to continue to function properly you will be asked
to supply the following kit file names when the update
installation reboots the system for the first time.
Be sure to record these file names for future use
within the update install process.  Each kit file
will only need to be entered once for all of the
associated hardware support to be loaded.

  Orpheus Document Builders ODB Device Driver
 (Kit File: /ODB/usr/opt/OAT100/sys/hardware/odb.kit)

Press <RETURN> to continue...
```

If you see this message, the hw_check test is successful. Otherwise, the hw_check utility displays an error message to describe the problem. You must correct the problem and run the test again.

6.4.4 Testing a Hardware Product Kit in a RIS Area

To install a hardware product kit into a RIS area, you must extract the base operating system into the RIS area first. Follow these steps to extract the base operating system into a new RIS area and then add the hardware product kit to the new area:

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 **`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:
```

4. Enter **`1`** to select `Install software into a new area`. You see the following prompt:

```
You have chosen to establish new remote installation
environment.

Enter the device special file name or the path of the
directory where the software is located (for example,
/mnt/ALPHA/BASE) or press <Return> to exit:
```

5. Enter the location of the base operating system distribution media, for example: **/mnt/ALPHA/BASE**. You see the following prompt:

```
Select the type of Tru64 UNIX base product to create.
If the software you are offering supports add-on hardware that is needed
to boot the client system, select "boot-link" as the type of RIS area to
create. Otherwise, select "standard". If you select "boot-link", the
software will be extracted (or copied) to the RIS area, because symbolically
linked RIS areas do not support this feature.

Choose one of the following options:
    1)  Standard boot method
    2)  Boot-Link method

Enter your choice:
```

6. Enter **2** to select `Boot-Link method`. You see messages similar to the following:

```
       Media extraction complete
The new environment is in /var/adm/ris/ris0.alpha

Building Network Bootable Kernel........Done

The following software now exists in RIS product area
/var/adm/ris/ris0.alpha:
     1   'Tru64 UNIX V5.0A Operating System ( Rev nnn )'
```

You see the RIS Utility Main Menu:

```
*** RIS Utility Main Menu ***

    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:
```

7. 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:
```

8. Enter **2** to select `Add software into an existing area`. You see a prompt similar to the following:

```
You have chosen to add a product to an existing environment.

Select the remote installation environment:

    1) /usr/var/adm/ris/ris0.alpha
         'Tru64 UNIX V5.0A Operating System ( Rev nnn )

Enter your choice or press RETURN to quit:
```

9. Enter **1** (in this example) to select the new RIS environment where you just extracted the base operating system. You see a prompt similar to the following:

```
Enter the device special file name or the path of the
directory where the software is located (for example,
/mnt/ALPHA/BASE) or press <Return> to exit:
```

10. Enter the location of the hardware product kit that you want to install, for example: **/mnt/kit**. This is the `kit` directory under the top-level directory of the hardware product kit on the distribution media.

    The `ris` utility searches the distribution media for the *kitname*`.kk` file that indicates that the distribution media contains a hardware product kit.

    You see a prompt similar to the following:

```
The kit you have specified has been identified as a Tru64 UNIX
kernel kit. This type of kit may contain software which is
needed during the booting of the kernel for the installation, due
to required hardware support. If you need to add this kit to the
base, select the option to integrate the kit. You may otherwise
choose to add this kit to the RIS area as a separate product.

1)  Integrate with Base product and include product
2)  Include as separate product
3)  Return to Main Menu

Enter your choice:
```

Note

If you do not see this message, the hardware product kit is not structured correctly. The most likely cause is a missing `./kit/instctrl/*.kk` file or specifying an invalid location in the previous step. Make sure all kit files are located in the appropriate directories and restart this test.

11. Enter **1** to select `Integrate with Base product and include product`. You see a prompt similar to the following:

```
Please select one of the following products to add the kit to.

    1   'Tru64 UNIX V5.0A Operating System ( Rev nnn )'

 Enter your selection or <return> to quit:
```

12. Enter **1** (in this example) to select the version of the base operating system where you want to integrate the kit. You see a prompt similar to the following:

```
Choose one of the following options:

    1)  Extract software from   /mnt/OAT100/kit
    2)  Create symbolic link to /mnt/OAT100/kit

 Enter your choice:
```

13. Decide whether you want to either extract the software onto the system or just create symbolic links to the software. In this example, enter **1** to select `Extract software`.

    The `ris` utility lists all the subsets associated with the software kit. In this example, the `OAT100` kit contains only one subset, and you see a prompt similar to the following:

```
The subsets listed below are optional:

     There may be more optional subsets than can be presented
     on a single screen. If this is the case, you can choose
     subsets screen by screen or all at once on the last
     screen. All of the choices you make will be collected
     for your confirmation before any subsets are extracted.

     1) ODB Device Driver Version 1

 Or you may choose one of the following options:

     2) ALL of the above
     3) CANCEL selections and redisplay menus
     4) EXIT without extracting any subsets

Enter your choices or press RETURN to redisplay menus.

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

14. Enter **1** (in this example) to select `ODB Device Driver Version 1`, or you can enter **2** to select `ALL of the above`. You see a prompt similar to the following:

```
You are installing the following optional subsets:

        ODB Device Driver Version 1

 Is this correct? (y/n):
```

15. Enter **y** to confirm your software selection.

    The `ris` utility integrates the hardware product kit software and the base system. No user interaction is needed at this time.

    You see messages similar to the following:

```
Checking file system space required to extract selected
subsets:

File system space checked OK.
Extracting OATBASE100...
Media extraction complete.

The following software now exists in the RIS product area
/var/adm/ris/ris0.alpha:
1   'Tru64 UNIX V5.0A Operating System ( Rev nnn )' with
    'ODB Device Driver Version 1'

2   'ODB Device Driver Version 1'
```

The hardware product has been extracted into the RIS area and integrated into a new version of the base operating system.

If you choose `SHOW software products in remote installation environments` from the RIS Utility Main Menu, you see that there are now two products in the new RIS area: the current version of the operating system with support for the `ODB` device driver and the `OATBASE100` hardware product kit.

6.4.5 Registering a RIS Client and Installing a Hardware Product Kit

After adding the base operating system and hardware product kit to a RIS area, the hardware product subsets are now available. However, before a client can perform a Full Installation from this RIS area, you must register the client. You need to know the RIS client's hardware network address.

If you do not know your RIS client's hardware network address, do one of the following:

- Log in to the RIS client as `root` or use the `su` command to gain superuser privileges, then shut down the system to the console prompt (`>>>`).

  Use the `show dev` command to show all devices, and look for the hardware address of your network interface in the form *xx-xx-xx-xx-xx-xx*. For example:

```
>>> show dev
        .
        .
        .
ewa0.0.0.0.1000.0 EWA0 xx-xx-xx-xx-xx-xx
        .
        .
        .
```

- Log in to the RIS client as `root` or use the `su` command to gain superuser privileges.

  Use the `uerf -r 300` command and look for the string `hardware address` in the ouput. Either that line or the next one contains the hardware network address. For example:

```
# uerf -r 300 | grep -i "hardware address" | uniq
_hardware address: xx-xx-xx-xx-xx-xx
```

  If the hardware address is not on the line that contains the string `hardware address`, search the output from the `uerf` command to find the correct hardware address. For example:

```
# uerf -r 300 | more
        .
        .
        .
   _Interface, hardware address:
   _xx-xx-xx-xx-xx-xx
        .
        .
        .
```

- Log in to the RIS server as `root` or use the `su` command to gain superuser privileges.

  Use the `ping` and `arp` commands to determine the hardware address of a running client from the RIS server. For example, to determine the hardware address of the RIS client `atlanta`, enter a command similar to the following example:

```
# /usr/sbin/ping -q -c1 atlanta ; arp atlanta
PING atlanta.cities.xsamplex.com (nn.nn.nnn.nnn): 56 data bytes
----atlanta.cities.xsamplex.com PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0/0/0 ms
atlanta (nn.nn.nnn.nnn) at xx-xx-xx-xx-xx-xx
```

Follow these steps to register your RIS client:

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`. You see the following messages:

```
You have chosen to add a client for remote installation
services.

The following conditions must be met to add a client:

        1. You must know the client processor's hostname
        2. The client's hostname must be in your system's
           host database(s).
        3. You must know whether the client is on an
           Ethernet, FDDI, or Token Ring network.
        4. You must know the client's hardware Ethernet,
           FDDI, or Token Ring address if the client is
           registering to install operating system software.
        5. If the client and the server reside on different
           subnets, you will need the address of the
           gateway(s) that the client can use to
           communicate with the server.

Do you want to continue? (y/n) [y]:
```

4. Enter **y** to continue if you have the information you need. You see the following prompt:

```
Enter the client processor's hostname or press RETURN
to quit:
```

5. Enter the client's host name. In this example, **client02**. You see a prompt similar to the following:

```
The existing environment is /usr/var/adm/ris/ris0.alpha

Select one or more products for the client to install
from /usr/var/adm/ris/ris0.alpha:

Product    Description
    1    'Tru64 UNIX V5.0A Operating System ( Rev nnn )' w/
         'ODB Device Driver Version 1'
    2    'ODB Device Driver Version 1'

Enter one or more choices as a space-separated list
(for example, 1 2 3):
```

6. Enter **1 2** (in this example) to select the products you want the client system to be able to install. This includes the base system that includes kernel support for the ODB device driver as well as the hardware product kit. You see a prompt similar to the following:

```
You chose the following products:

   1    'Tru64 UNIX V5.0A Operating System ( Rev nnn )' w/
        'ODB Device Driver Version 1'
   2    'ODB Device Driver Version 1'

Is that correct? (y/n) [y]:
```

7. Enter **y** to confirm your selection. You see the following prompt:

```
Network type:
        1) Ethernet or FDDI
        2) Token Ring

Enter your choice:
```

8. Enter the network type. You see the following prompt:

```
Enter the client processor's hardware network address.
For example, 08-00-2b-02-67-e1: xx-xx-xx-xx-xx-xx
```

9. Enter the RIS client's hardware network address. You see the RIS Utility Main Menu:

```
*** RIS Utility Main Menu ***

    a) ADD a client
    d) DELETE software products
    i) INSTALL software products
    l) LIST registered clients
    m) MODIFY a client
    r) REMOVE a client
    s) SHOW software products in remote installation
       environments
    x) EXIT

Enter your choice:
```

10. Enter **x** to exit from the `ris` utility.

The client system can now boot over the network from the RIS area with the kernel that contains the hardware product subsets. For example:

```
>>> boot -flag fa boot_device
```

In the previous example:

- The `-flag fa` defines boot flags `f` for a hardware product kit and `a` for multiuser mode.
- The *boot_device* represents the console network boot device name. The device name depends on the processor type, but it usually is `ewa0`. Use the `show dev` command at the console mode prompt to determine the boot device for your processor.

The boot procedure installs the kernel from the RIS area and then performs a Full Installation by loading the operating system subsets and the subsets from the hardware product kit.

For more information about RIS, see the guide to *Sharing Software on a Local Area Network*. For more information about booting a system over the network, see the *Installation Guide*.

A

Creating a Consolidated Firmware CD-ROM

A consolidated firmware CD-ROM lets you upgrade your processor firmware at the same time that you install the operating system. This appendix describes how to create a consolidated firmware CD-ROM, which requires the operating system CD-ROM in UFS format and the required *Alpha Systems Firmware CD-ROM*. The following information is included in this appendix:

- How to prepare for the build (Section A.1.1)
- How to build the consolidated firmware CD-ROM (Section A.1.2)
- Sample sessions for both the build preparation (Section A.2.1) and the actual building (Section A.2.2) of the consolidated firmware CD-ROM

This release includes the documentation and utilities that you need to build a consolidated firmware CD-ROM in ISO 9660–compliant format. This documentation includes the `disklabel`(8) and `mkisofs`(8) reference pages.

A.1 Build Instructions

The following sections describe how to prepare and build a consolidated firmware CD-ROM.

Note

The examples in this appendix use the C shell.

A.1.1 Prepare for the Build Session

After you receive a new kit, follow these steps to move the necessary files from the CD-ROM to working directories on the build machine:

1. Log in as `root` or use the `su` command to gain superuser privileges.
2. Use the `disklabel` utility to set up a 635 Mb partition on a spare disk, starting at block 0, with a size of 1300480 512–byte blocks and a file system type of `unused`. Create a mount point for this partition (such as `/cdimage`) to use later.

For example, to set partition a of dsk6 starting at offset 0 with a size of 1300480, and create mount point /cdimage:

```
% disklabel -F -r -e /dev/disk/dsk6
write new label [y] y
% mkdir /cdimage
```

3. Mount the operating system CD-ROM to a temporary mount point (such as /mnt) and use the tar command to copy the contents of the CD-ROM onto a suitably large directory on the system (at least 1.5 Gb). After this is done, unmount the CD-ROM.

Note

This step may take as long as 60 minutes to complete.

For example, using /spare as the target directory and /dev/disk/cdrom4 as the CD-ROM drive:

```
% mount -r /dev/disk/cdrom4a /mnt
% cd /mnt
% tar cf /spare/os_copy.tar .
% cd /
% umount /mnt
```

A.1.2 Build the Consolidated Firmware CD-ROM

After you have completed the steps in Section A.1.1, follow these steps to consolidate the necessary data to a single CD-ROM in ISO9660–compliant format:

1. Log in as root or use the su command to gain superuser privileges.
2. Use the newfs command to initialize a file system on the partition reserved in Step 2 of Section A.1.1 and mount it to the mount point /cdimage. If you are prompted for confirmation, enter **y**. Use the tar utility to copy the base operating system image created in Step 3 of Section A.1.1 to /cdimage.

Note

This step may take as long as 60 minutes to complete.

For example, using /spare as the source and dsk6c as the target partition:

```
% newfs /dev/disk/dsk6c
% mount /dev/disk/dsk6c /cdimage
% cd /cdimage
% tar xpf /spare/os_copy.tar
% cd /
```

3. Optionally use the following multiple step operation to copy the firmware images to the target directory:

   a. Mount the *Alpha Systems Firmware CD-ROM* to a temporary mount point such as `/mnt`. For example, using `/dev/disk/cdrom4a` as the CD-ROM drive:

      ```
      % mount -t cdfs -r /dev/disk/cdrom4a /mnt
      ```

   b. Copy the System Marketing Model (SMM) table from the *Alpha Systems Firmware CD-ROM* to the target directory. The SSM table maps system models to firmware image files.

      ```
      % cp /mnt/SMMTABLE.TXT\;1 /cdimage/smmtable.txt
      ```

      Caution

      The target file name must be in lower case with the `;1` removed from the end.

   c. Look in the SMM table to find the name and locations of the firmware images to be copied by entering the following command:

      ```
      % more /cdimage/smmtable.txt
      ```

      As an example, the entry for the EV5 AlphaServer 1000A platform is similar to the following example (the actual table entry is on one line):

      ```
      27    5    1270,1311,1558,1580-1581\
      [ALPHA1000A]AS1000A_E5_V5_1.EXE;1\
      6    5.1    !    AlphaServer 1000A 5/xxx
      ```

      In this example, the firmware file on the CD-ROM is `AS1000A_E5_V5_1.EXE;1`.

   d. Create the required firmware directories in the target directory, and copy each of the platform firmware images that you want from the *Alpha Systems Firmware CD-ROM*.

      Caution

      The target file name must be in lower case with the "`;1`" removed from the end. Otherwise, the `fwupgrade` program cannot locate the firmware images. If the source file is `AS1000A_E5_V5.1.EXE;1`, the target file is `as1000a_e5_v5_1.exe`.

For example, using the file names on the *Alpha Systems Firmware CD-ROM*:

```
% mkdir /cdimage/alpha800
% mkdir /cdimage/alpha1000a
% mkdir /cdimage/as4x00
% cp /mnt/ALPHA800/AS800_V5_1.EXE\;1 \
     /cdimage/alpha800/as800_v5_1.exe
% cp /mnt/ALPHA1000A/AS1000A_E5_V5_1.EXE\;1 \
     /cdimage/alpha1000a/as1000a_e5_v5_1.exe
% cp /mnt/AS4X00/AS4X00_IMAGE.EXE\;1 \
     /cdimage/as4x00/as4x00_image.exe
```

e. Unmount and remove the firmware CD-ROM.

```
% umount /mnt
```

Note

You cannot repackage firmware or software unless you have a specific licensing agreement with the manufacturer that allows you to do so.

4. Use the `mkisofs` program to build the target CDFS file image of the directory structure in `/cdimage`. For example, using `/spare` as the target location for the image:

```
% /usr/sbin/mkisofs -D -R -a -d -o \
 /spare/cons_oper_sys.cdfs /cdimage/
```

Refer to the `mkisofs`(8) reference page for more information.

5. Use the `disklabel` command to insert a label into the file generated in Step 4.

```
% disklabel -r -w -t cdfs -f \
 /spare/cons_oper_sys.cdfs \
 /mdec/rzboot.cdfs /mdec/bootrz.cdfs
```

Refer to the `disklabel`(8) reference page for more information.

The CD image file `/spare/cons_oper_sys.cdfs` is ready to be written to a CD-ROM.

A.2 Sample Build Session

The following assumptions are made for the examples in this section:

- The target partition is on /dev/disk/dsk6c.
- The /spare directory has at least 1.5 Gb of free space.
- The CD-ROM drive is /dev/disk/cdrom4.

Note

The examples in this appendix use the C shell.

A.2.1 Preparing for the Build Session

Follow these steps to prepare for the CD-ROM build session:

1. Log in as root or use the su command to gain superuser privileges.
2. Enter the following commands:

```
% cd /
% disklabel -F -r -e /dev/disk/dsk6
write new label? [y] y
% mkdir /cdimage
```

3. Place the operating system CD-ROM into the CD-ROM drive, and enter the following commands:

```
% mount -r /dev/disk/cdrom4a /mnt
% cd /mnt
% tar cf /spare/os_copy.tar .
% cd /
% umount /mnt
```

4. Remove the operating system CD-ROM from the drive. The preparatory steps are complete.

A.2.2 Building the Consolidated Firmware CD-ROM

Follow these steps to build a consolidated firmware CD-ROM:

1. Log in as root or use the su command to gain superuser privileges.
2. Enter the following commands:

```
% cd /
% newfs /dev/disk/dsk6c
% mount /dev/disk/dsk6c /cdimage
% cd /cdimage
% tar xpf /spare/os_copy.tar
% cd /
```

3. Place the *Alpha Systems Firmware CD-ROM* into the CD-ROM drive, and enter the following command:

```
% mount -t cdfs -r /dev/disk/cdrom4a /mnt
% cp /mnt/SMMTABLE.TXT\;1 /cdimage/smmtable.txt
% more /cdimage/smmtable.txt
```

4. Review the output to determine the required directory and file names for the firmware images that you want.

   The following example uses the same firmware images as Step 3d of Section A.1.2:

```
% mkdir /cdimage/alpha800
% mkdir /cdimage/alpha1000a
% mkdir /cdimage/as4x00
% cp /mnt/ALPHA800/AS800_V5_1.EXE\;1 \
     /cdimage/alpha800/as800_v5_1.exe
% cp /mnt/ALPHA1000A/AS1000A_E5_V5_1.EXE\;1 \
     /cdimage/alpha1000a/as1000a_e5_v5_1.exe
% cp /mnt/AS4X00/AS4X00_IMAGE.EXE\;1 \
     /cdimage/as4x00/as4x00_image.exe
% umount /mnt
```

5. Remove the *Alpha Systems Firmware CD-ROM* and enter the following commands:

```
% /usr/sbin/mkisofs -D -R -a -d -o \
 /spare/cons_oper_sys.cdfs /cdimage/
```

   Output is similar to the following:

```
Using OSFMANWO.000;1 for \
/cdimage/ALPHA/BASE/instctrl/OSFMANWOS505.scp \
(OSFMANWOP505.scp)
 Using OSFMANWO.001;1 for \
/cdimage/ALPHA/BASE/instctrl/OSFMANWOS505.inv \
 (OSFMANWOP505.inv)
 Using OSFMANWO.002;1 for \
 /cdimage/ALPHA/BASE/instctrl/OSFMANWOS505.ctrl \
 (OSFMANWOP505.ctrl)
                .
                .
                .
 Using PROCFS_V.000;1 for \
 /cdimage/usr/sys/procfs/procfs_vnops_stubs.c \
(procfs_vfsops_stubs.c)
   3.92% done, estimate finish Fri May 19 15:36:59
   5.87% done, estimate finish Fri May 19 15:39:24
                .
                .
                .
  99.74% done, estimate finish Fri May 19 15:41:52
 Total extents actually written = 255673
 Total translation table size: 0
 Total rockridge attributes bytes: 2066594
 Total directory bytes: 4239360
 Path table size(bytes): 10130
 Max brk space used b9ec60
 255673 extents written (499 Mb)
```

Note

The backslashes (\) in the previous example indicate line continuation and are not present in the output.

6. Enter the following commands:

```
% disklabel -r -w -t cdfs -f \
 /spare/cons_oper_sys.cdfs \
 /mdec/rzboot.cdfs /mdec/bootrz.cdfs
```

The information is consolidated, and the file can be written onto a CD-ROM.

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. Refer to your hardware documentation for information about valid parameters for the `boot` utility on your system.

C

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 loosely-coupled collection of servers that share storage and other resources, providing high availability of applications and data. A cluster consists of communications media, member systems, peripheral devices, and applications. One system can form a single-member cluster.

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 connnected to a private physical bus for intracluster communications and 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. Hardware products can only be distributed on CD-ROM in Direct CD-ROM format.

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. Refer to the `gendisk`(1) reference page.

See also *kitcap database*

gentapes utility

The `gentapes` utility is used to produce magnetic tape distribution media for a product kit. Refer to the `gentapes`(1) reference page.

See also *kitcap database*

H

hardware product

A hardware product includes kernel modules to support hardware devices. A hardware product kit, such as a device driver, can be installed during the initial installation and bootstrap linking of the operating system.

See also *kernel product*, *layered product*, *NHD*

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*

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 are usually 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 *hardware product*, *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 *hardware product*, *kernel product*, *user product*

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*

member-specific file

A file 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.

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 distribution (NHD) provides delivery of support for new hardware without providing a new release of the operating system. The kit is usually provided on CD-ROM, and includes installation and testing instructions.

See also *hardware product*

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 know 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.enet.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. Refer to the `setld`(8) reference page.

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 contains 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*

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. Refer to the `tar`(1) reference page for information about using the `tar` command.

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 *hardware product*, *kernel product*, *layered product*

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

How to Order Tru64 UNIX Documentation

You can order documentation for the Tru64 UNIX operating system and related products at the following Web site:

http://www.businesslink.digital.com/

If you need help deciding which documentation best meets your needs, see the Tru64 UNIX *Documentation Overview* or call **800-344-4825** in the United States and Canada. In Puerto Rico, call **787-781-0505**. In other countries, contact your local Compaq subsidiary.

If you have access to Compaq's intranet, you can place an order at the following Web site:

http://asmorder.nqo.dec.com/

The following table provides the order numbers for the Tru64 UNIX operating system documentation kits. For additional information about ordering this and related documentation, see the *Documentation Overview* or contact Compaq.

Name	Order Number
Tru64 UNIX Documentation CD-ROM	QA-6ADAA-G8
Tru64 UNIX Documentation Kit	QA-6ADAA-GZ
End User Documentation Kit	QA-6ADAB-GZ
Startup Documentation Kit	QA-6ADAC-GZ
General User Documentation Kit	QA-6ADAD-GZ
System and Network Management Documentation Kit	QA-6ADAE-GZ
Developer's Documentation Kit	QA-6ADAG-GZ
Reference Pages Documentation Kit	QA-6ADAF-GZ

Reader's Comments

Tru64 UNIX
Guide to Preparing Product Kits
AA-RH9SB-TE

Compaq welcomes your comments and suggestions on this manual. Your input will help us to write documentation that meets your needs. Please send your suggestions using one of the following methods:

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

If you are not using this form, please be sure you include the name of the document, the page number, and the product name and version.

Please rate this manual:

	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Usability (ability to access information quickly)	☐	☐	☐	☐

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____
_____
_____
_____

What version of the software described by this manual are you using? _____

Name, title, department _____
Mailing address _____
Electronic mail _____
Telephone _____
Date _____

Do Not Cut or Tear - Fold Here and Tape

COMPAQ

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPAQ COMPUTER CORPORATION
UBPG PUBLICATIONS MANAGER
ZKO3-3/Y32
110 SPIT BROOK RD
NASHUA NH 03062-2698

Do Not Cut or Tear - Fold Here

Cut on This Line