COMPAQ

Guide to Preparing Product Kits

Compaq Tru64™ UNIX®

Tru64 UNIX

Guide to Preparing Product Kits

Part Number: AA-QYW7C-TE

April 1999

Product Version: Tru64 UNIX Version 4.0F or higher

This book describes the procedures for creating, delivering, and installing layered product kits for use on Compaq Tru64 UNIX (formerly DIGITAL UNIX) operating systems.

Compaq Computer Corporation
Houston, Texas

Contents

Figures

Tables

About This Manual

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

Audience

This book is intended for software developers who are responsible for creating product kits. They are expected to be moderately experienced with UNIX based operating systems and should have experience performing system administration tasks.

New and Changed Features

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

- The definition of a kit at the beginning of Chapter 1 has been expanded and clarified.
- Notes have been added to Section 1.3 and Section 3.1.6 to explain subset control program requirements for DMS compliance.
- The definition of the `DEPS` subset control file field has been changed in Section 4.4.3.
- The sections referring to foreign device kits have been removed. This information has been superseded by Chapter 5 which describes how to create, test, and deliver hardware product kits.
- Appendix A has been added to provide instructions on how to create and build a consolidated CD-ROM. A consolidated CD-ROM lets you upgrade your processor firmware at the same time that you install the operating system.

Organization

This manual is organized as follows:

Chapter 1	Introduction Presents an introduction to the kit-building process.
Chapter 2	Creating the Kit Directory Structure Describes how to create kit directories and build product kits.
Chapter 3	Creating Subset Control Programs Describes how to write subset control programs (SCPs) to install and manage software subsets.
Chapter 4	Building Subsets and Control Files Describes how to create subsets and subset control files with the `newinv` and `kits` utilities.
Chapter 5	Hardware Product Kits Describes how to create, test, and deliver hardware product kits for new or existing hardware.
Chapter 6	Producing Distribution Media for User and Kernel Product Kits Describes how to produce a user or kernel product kit on the distribution media.
Chapter 7	Testing a User or Kernel Product Kit Describes how to test user and kernel product kits on the target system.
Appendix A	Creating a Consolidated CD-ROM Dscribes how to create a consolidated CD-ROM.
Appendix B	Standard Directory Structure Describes the standard directory hierarchy.
Glossary	Defines terms used in this manual.

Related Documents

The printed version of the Tru64 UNIX documentation set is color coded to help specific audiences quickly find the books that meet their needs. (You can order the printed documentation from Compaq.) This color coding is reinforced with the use of an icon on the spines of books. The following list describes this convention:

Audience	Icon	Color Code
General users	G	Blue
System and network administrators	S	Red
Programmers	P	Purple
Device driver writers	D	Orange
Reference page users	R	Green

Some books in the documentation set 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.

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: Tutorial*

  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 Installation or Full Installation of the operating system on all supported processors and single-board computers. It explains how to prepare your system for installation, boot the processor, and perform the installation procedures.

- *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 03062-9987

  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

Overview of Product Kits

This guide is intended to provide product and kit developers with the proper method to create a product **kit** for a UNIX based operating system environment. A product kit is the collection of files and directories that represent a new or upgraded product 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 by the `setld` utility to install the product on the system. The 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 in Direct CD-ROM format.

All product kits consist of two types of files: product kit files and kit support files (such as subset control files). The subset control files are instrumental in telling the `setld` utility where and how the files should be installed onto the target system. Product kit files are the actual files that compose the product kit being delivered.

If you are preparing a kit to support new or additional hardware to be installed (or an upgrade kit for existing hardware) on the customer's system, you need to provide hardware product files along with the subset control files and the product kit files for the operating system to properly install the hardware product kit.

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.

After you determine the kind of kit you are building, you can refer to individual chapters of the book for detailed steps for building your particular kit.

Note

Many processes in this document refer to the **osf_boot** utility. This is a common term referring to the `boot` command for your hardware's console subsystem that supports the operating system. For additional information, refer to your system's hardware documentation. At the >>> console prompt, you can enter **`help boot`** to view online help for this command.

1.1 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. Layered products can fall into the following categories:

- User product

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

- Kernel product

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

1.2 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 the group is required or optional for the installation procedure. You can copy the product files onto the distribution media in one of the following formats:

- `tar` format

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

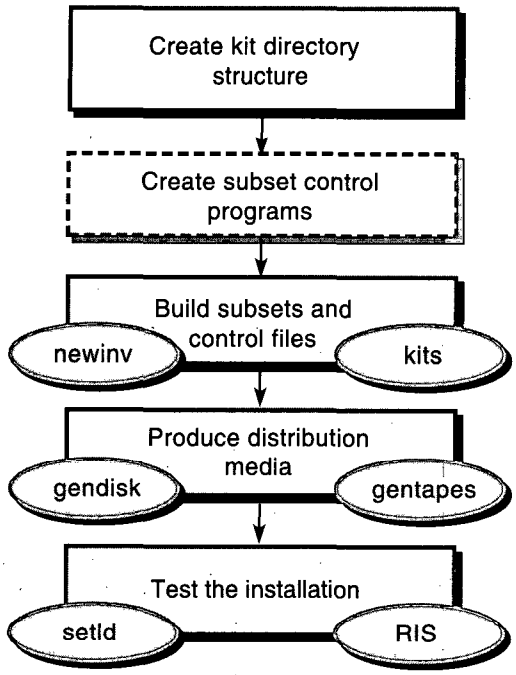
- Direct CD–ROM (DCD) format

  In **direct CD-ROM 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.

1.3 Kit-Building Process

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

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



The kit-building process consists of the following steps:

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

   On the development system, you 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.

   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.

2. Optionally create subset control programs

   The `setld` utility can call a **subset control program** (SCP) to perform installation steps specific to your kit. The SCP is optional. You supply it on your kit only if the product requires special installation steps. 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. Example 3–5 in Chapter 3 shows a subset control program for a kernel product that uses the `kreg` utility.

Note

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 is the default path for subset control program processing commands to be run from the server in a DMS environment:

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

Refer to Section 3.1.6 and *Sharing Software on a Local Area Network* for more information about DMS.

3. Building 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. The `kits` utility is invoked from the same directory in which the master inventory file is located.

4. Producing the 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 must also modify the kit and add files to support your system's bootstrap link. Hardware product kits must use the DCD format.

5. Testing the installation of the kit

   After you have successfully created the kit, you should test the installation of the kit. For user products and kernel products, you install the kit by running the `setld` utility. For hardware product kits, first test the kit by running the `setld` utility and then test by using your system boot utility's bootstrap link technology to bootstrap the custom kernel and test the ability of the kit to be installed successfully by the installation process. It is also recommended that you install the kit in a RIS area so that RIS clients can install it across a network.

1.4 Sample Products Used in This Book

This book uses the following fictitious products to demonstrate how to build kits for each product type:

- User Product — Orpheus Document Builder (ODB)

  The Orpheus Document Builder application is produced by the fictitious company, *Orpheus Authoring Tools, Inc.*. In the examples in this book, the Orpheus Document Builder product is used to show how to build a kit for a user product.

- Kernel Product — `/dev/none` device driver

  The `/dev/none` peripheral device driver is developed by the fictitious company, *EasyDriver, Inc.*. In the examples in this book, the `/dev/none` device driver is used to show how to build a kit for a kernel product. The *Writing Device Drivers: Tutorial* introduced this fictitious product.

- Hardware Product — `edg` graphics device driver

  The `edg` graphics device driver, which *EasyDriver, Inc.* also produces, is used in examples to show to how to build a kit for a hardware product.

2

Creating the Kit Directory Structure

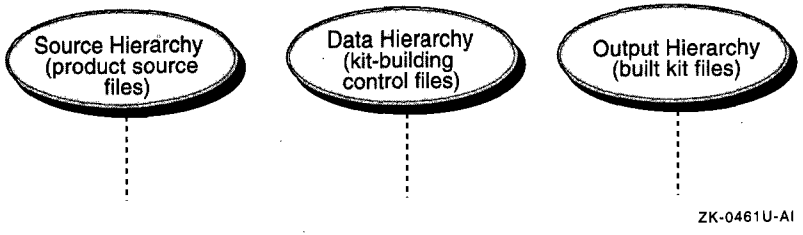
When engineers finish developing a product, they give the product files to you, the kit developer, for packaging and processing into a kit. Your first task is to organize these files by function and use, then to place them in a kit-building directory structure. When designing the kit-building directory structure, you must consider where you want to place the product files on the customer's system. You then create a kit directory structure on the development system that closely mirrors the customer's directory structure.

This chapter describes the standard directory structure and how to create a kit-building directory structure to fit within the standard directory structure for user, kernel, and hardware product kits.

2.1 Creating a Kit Building Directory Structure

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

Figure 2–1: Kit Directory Hierarchy



The following describes each directory hierarchy:

- Source hierarchy

  The **source hierarchy** is a directory structure that 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 appropriate directory in the source hierarchy. You can create the source hierarchy under any directory you choose.

- Data hierarchy

  The **data hierarchy** is a directory structure that 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.

  The `kits` utility is run from the directory in which these files are located. 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. Additional files may be required, depending on the kit type.

- Output hierarchy

  The **output hierarchy** is a directory structure that contains the subsets that are placed on the kit. The subset control files that are needed during installation are stored in the `./kit/instctrl` subdirectory. 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.

Follow these steps to create the kit-building directory structure:

1. Issue the appropriate `mkdir` commands for each of the directories and subdirectories that you need. Refer to the `mkdir`(1) reference page if you need more information.
2. Populate the `src` directory with all the files that are to be part of the finished kit.
   - Section 2.2.1 describes the directory file structure you need for a user product
   - Section 2.2.2 describes the directory file structure you need for a kernel product
   - Section 2.2.2 and Section 5.1 describe the directory file structure you need for a hardware product

   You can choose any appropriate method for populating the source hierarchy. For example, you could create a `Makefile` file for use with the `make` command.

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 the subset control programs have modified are reset to their original values in the kit's master inventory files.

2.2 Populating the Source Directory

It is possible to install the components of a kit in any directory on the customer's system. However, guidelines exist for deciding where to place kit files. The standard system directory structure is set up for efficient organization. It separates files by function and use.

You should install product files in subdirectories of `/opt`, `/usr/opt`, and `/usr/var/opt`, as follows:

- Boot files reside in `/opt`

  Files that are required at bootstrap time, such as device drivers, are installed in a product-specific subdirectory of `/opt`. This also includes any files accessed before file systems other than root (`/`) are mounted.

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

  Files that are usually read-only, such as commands, startup files (which can be modified, but not by individual users), or data files are installed in a product-specific subdirectory of `/usr/opt`.

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

  Files that users can read and write, such as lists of data that any user is allowed to change, are installed in a product-specific subdirectory of `/usr/var/opt`.

The first thing you will need to get is a **three letter product code** from Compaq Computer Corporation for your product kit name. To obtain a product code, send mail to the `Product@DSSR.enet.dec.com` electronic mail address. You use this product code and a product version number that you assign to name the product-specific subdirectories of your product.

Examples in this book use `OAT` as the prefix for the product-specific subdirectory names for the Orpheus Document Builder (ODB) product kit.

Assuming this is the first release of the product, the kit developer chose 100 as the version number. As such, directories are named /opt/OAT100, /usr/opt/OAT100, and /usr/var/opt/OAT100. OAT is the code assigned to *Orpheus Authoring Tools, Inc.*, the ficticious company who developed the ODB product, and 100 is the product version number.

Using a standard directory structure has the following advantages:

- 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.
- 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.
- Name space conflicts are avoided. 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 /opt, /usr/opt and /usr/var/opt avoids this conflict because each three letter product code is unique to a particular original equipment manufacturer (OEM).

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 normal 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 commands for the product are located in the /opt/OAT100/bin and /usr/opt/OAT100/bin directories. To be able to use ODB commands without specifying the full path on the command line, the user can add the product path to the PATH environment variable.

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

Shipping a symbolic link can be done if:

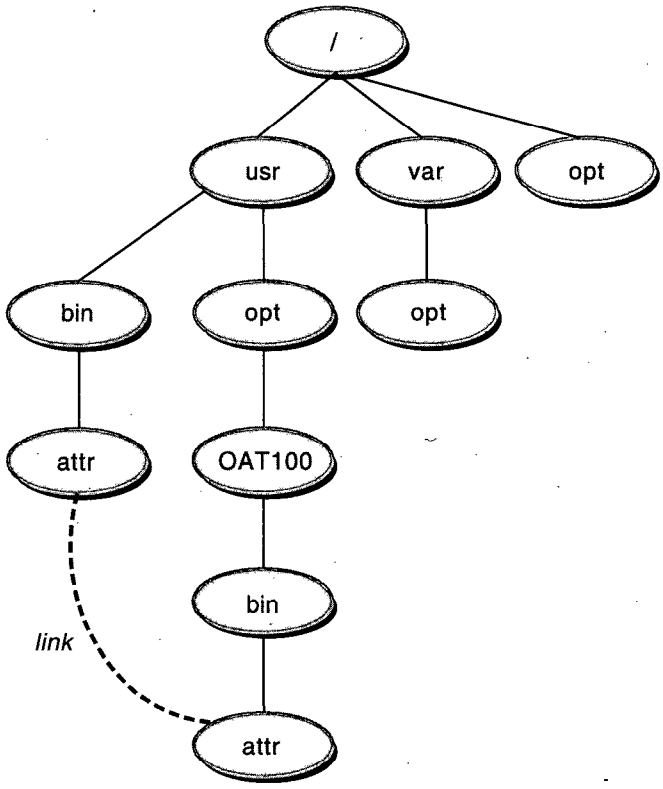
- The symbolic link does not conflict with any base operating system file. Using our example, it means that you could create the /usr/bin/attr

link only if the operating system does not already contain a /usr/bin/attr file. If the operating system did contain a /usr/bin/attr file, you could not ship the symbolic link, because installing the link would overwrite an operating system file.

- The command name does not conflict with any standard operating system command. For example, the /usr/opt/OAT100/bin/attr command is shipped in the ODB kit and as part of the standard operating system in /bin/attr. When the user enters the attr command, there would be a command name conflict and, depending upon which directory is first in the search path, /bin or /usr/bin, the user could be getting the operating system version or the symbolically linked ODB product version.

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 above the OAT* directories are the existing directories on the customer's system. All directories and files created by the layered product ship 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.*.

Figure 2–2: Layered Product Standard Directories



Caution

Shipping any file outside of the `/opt`, `/usr/opt`, and `/usr/var/opt` directories is not recommended and could cause a name space conflict with the base operating system or another layered product.

Overwriting base operating system files can cause the following problems:

- Your product will 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 with the new version of the file.
- Overwriting a base operating system file can prevent an Update Installation from completing successfully and may render the system unusable.

- The user could be forced to remove your product from the system as a part of an Update Installation process. The user would then have to reinstall your product after the Update Installation has been completed.

2.2.1 Directory Structure for a User Product Kit

The files are installed in directories under `/opt`, `/usr/opt`, and `/usr/var/opt` so that the files are centrally located and easy to find. The ODB kit contains files to be installed in the following directories:

- `/usr/opt/OAT100/bin`
- `/usr/opt/OAT100/lib/br`
- `/usr/opt/OAT100/lib/doclib/templates`

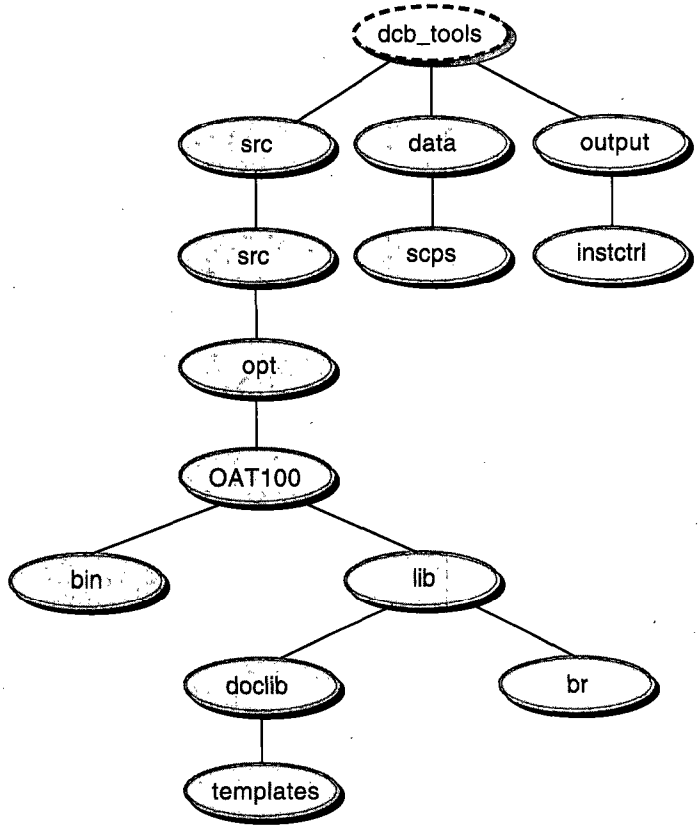
Figure 2–3 illustrates the complete directory structure for the ODB kit. In this figure, the top level directory (drawn with dashed lines), `dcb_tools`, represents an existing directory under which a kit developer created the hierarchy directories.

The `src` directory represents the `root (/)` directory on the customer's system; the `usr` directory represents `/usr` on the customer's system. All the other directories in the source hierarchy are mapped to the customer's system in the same way.

The name of the top-level product-specific directory, under the source hierarchy's `opt` directory, is made up of the product code and a three-digit version number, where the first digit identifies the major version number, the second digit identifies the minor version number, and the third digit identifies the update level. For example, the product code for the ODB kit is `OAT` and its version number is `100`, indicating major version 1, minor version 0, update 0. Version numbers cannot be lower than 100.

If the ODB kit included user-writable files, they would be placed under the `/usr/var/opt/OAT100` directory. It is recommended that this convention be used for consistency among user products.

Figure 2–3: Directory Hierarchy for the ODB Kit

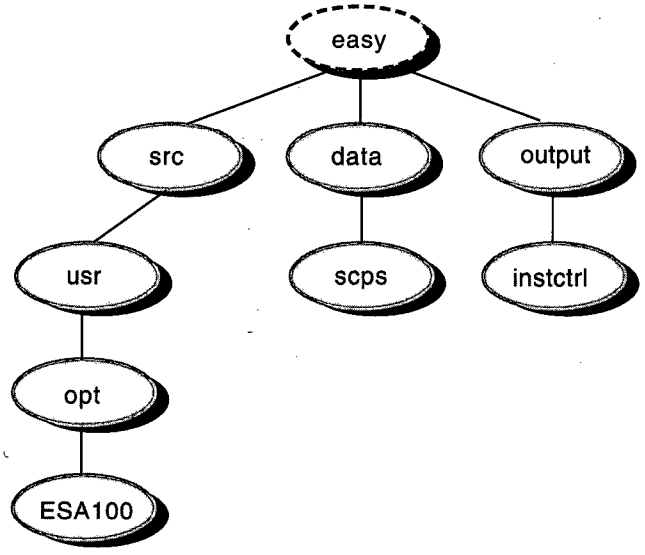


2.2.2 Directory Structure for Kernel Product and Hardware Product Kits

You set up a kit directory structure for a kernel product and hardware product in the same way as you would for a user product. You create three directory hierarchies — `src`, `data`, and `output` — and populate the source hierarchy with the product files. Under the `src` directory, create a directory structure similar to the one on the customer's system and place the product files in `/opt`, `/usr/opt`, and `/usr/var/opt` as appropriate. Unlike a user product, the kit for a kernel product or hardware product (such as a device driver) requires certain files to be present in specific directory locations.

Figure 2–4 shows the directory structure of a device driver product as it would appear in the kit development area. The driver shown here is the `/dev/none` driver produced by the fictitious company called *EasyDriver, Inc.* This driver is first introduced in *Writing Device Drivers: Tutorial.*

Figure 2–4: Directory Structure for the /dev/none Driver Kit



ZK-1198U-AI

The top-level directory (which is drawn with dashed lines) `easy` represents the working area for all kit development at *EasyDriver, Inc.*. The `src` directory corresponds to the customer's root directory (`/`). Directories under `src` have a one-to-one relationship to directories on the customer's system. The `ESA100` directory represents the top-level product directory for the `/dev/none` driver.

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

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

Note:

A module that is capable of being loaded dynamically also can be linked statically. The only difference is the call to configure the

driver (for more information on static or dynamic drivers, see *Writing Device Drivers: Tutorial*).

The following list describes the files that go into a kernel or hardware kit for a device driver, the directories where they reside, and the types of drivers that use them:

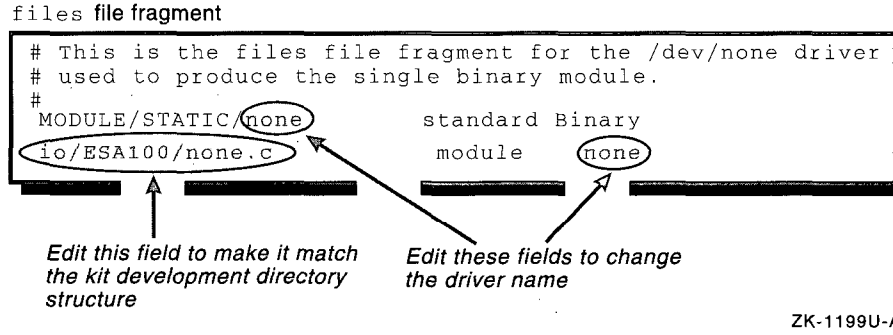
- `files` file fragment

  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. For both statically and dynamically configured drivers, place this file in a product directory, such as `/opt/ESA100/etc`. 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. Figure 2–5 shows which fields within the `files` file fragment need to change.

Note

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 2–5: Editing the files File Fragment

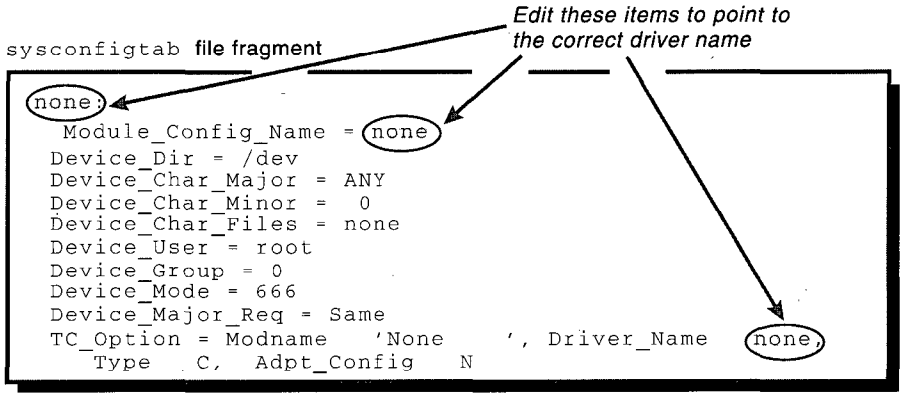


- `sysconfigtab` file fragment

  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 or hardware 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/ESA100/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 2–6. 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 2–6: Editing the sysconfigtab File Fragment



ZK-1203U-AI

- *driver*.mod **object module file**

  Contains the single binary module for both statically and dynamically configured drivers. You should include this file in a product directory, such as /opt/ESA100/sys/BINARY which must be in the root (/) file system. The kernel will not allow two kernel modules to have the same name. To avoid kernel module naming conflicts with other OEMs and the base operating system, it is recommended that you prepend your three letter product code to your module names.

  Note

  Module files for hardware products must be compressed using the objZ utility. Do not use the compress or gzip utilities to compress module files.

- *.c (source) and *.h (header) files

  Contain the source code for the device driver. You should include these files in a product directory, such as /usr/opt/ESA100, when the driver is statically configured and distributed in source form.

- *device*.mth **method files**

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

3

Creating Subset Control Programs

This chapter describes how to write subset control programs for layered products.

A subset control program (SCP) performs special tasks beyond the basic installation tasks managed by the `setld` utility. The following are 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.

3.1 Common Characteristics of a Subset Control Program

Regardless of the specific tasks they perform, all subset control programs share the following characteristics:

- They are named according to certain conventions and placed in the kit-building directory structure so that the `kits` utility can find them.
- They include library routines supplied by the operating system.
- They are invoked at various times by the `setld` utility.
- If errors occur, they must exit and return an error status to the `setld` utility.
- They can call routines to return subset information to global variables. By using these routines, you do not have to hard code subset information into the subset control program.

- They can call routines to determine whether the subset control program is running in a dataless environment.

The following sections describe the characteristics shared by all subset control programs.

3.1.1 Creating Subset Control Program Source Files

You create one subset control program for each subset that requires special handling during installation. You can write the program in any programming language, but you must take care that your subset control program is 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.

Usually subset control programs are 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.

Place all subset control programs that you write in the `scps` directory, a subdirectory of the `data` directory. The 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 `OATODBDOC100`. If both subsets required a subset control program, the source file names would be `OATODB100.scp` and `OATODBDOC100.scp`.

When you create the subsets as described in Chapter 4, the `kits` utility copies the subset control programs from the `scps` directory to the `instctrl` directory. If a subset has no subset control program, the `kits` utility creates an empty subset control program file for it in the `instctrl` directory.

3.1.2 Including Library Routines

A set of routines in the form of Bourne shell script code is provided by the operating system. These routines are located in the `/usr/share/lib/shell/libscp` file. Do not copy these routines into your subset control program. Such a design would prevent your kit from receiving the benefit of enhancements or bug fixes made in future releases. Use the shell's `source` command to call in the routines, as follows:

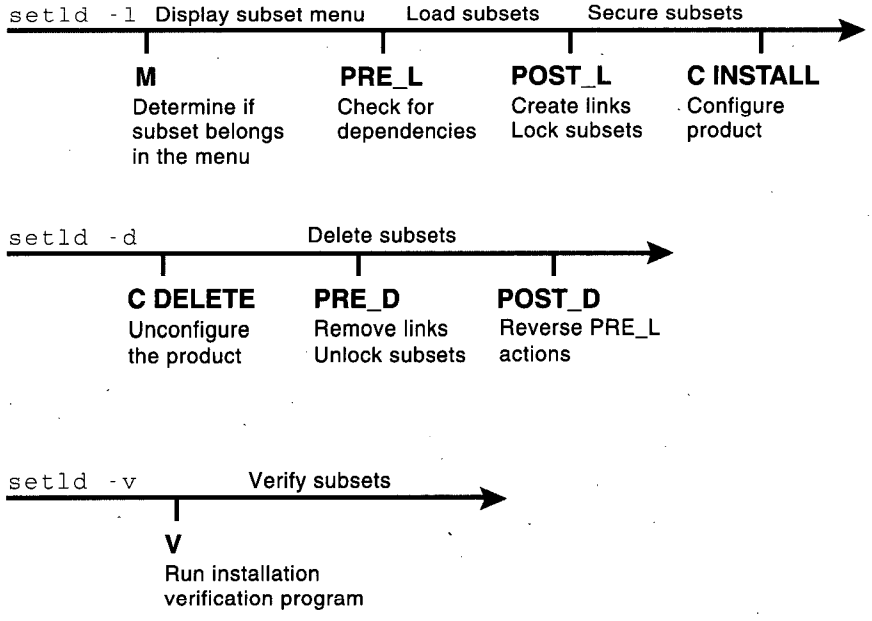
```
. /usr/share/lib/shell/libscp
```

3.1.3 Invoking Subset Control Programs

Your kit does not need to do anything to invoke its subset control program. The `setld` utility invokes it during various phases of the installation procedure. The subset control program can perform any tasks that it needs during a phase, such as creating or deleting a file or displaying messages. Certain tasks, such as performing dependency checks or creating forward and backward links, should be performed only during specific phases if the installation requires them.

Figure 3–1 shows timelines of the `setld` utility when it is invoked with the `-l`, `-d`, and `-v` options. The actions of the `setld` utility are written above the timelines; the value of the `ACT` environment variable and the actions taken by the subset control program at each phase are shown below the timelines.

Figure 3–1: Time Line of the setld Utility



ZK-1220U-AI

When it enters a new phase, the `setld` utility sets the `ACT` environment variable to a value that corresponds to the phase, then it invokes your subset control program. The subset control program checks the value of the environment variable to determine what action it needs to take. In some cases, the `setld` utility also passes arguments to the subset control program. The subset control program uses the argument values to further determine the actions it needs to take.

Do not include a wildcard 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 can be invoked. The `case` statements that choose an action simply exit with zero status in these undetected cases, and the `setld` utility continues.

3.1.4 Stopping the Program

Depending on the tests it makes, your subset control program could decide at some point to stop the installation or deletion of its subset. For example, if it checks for the existence of subsets upon which your product depends and fails to find one or more of them, 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.

3.1.5 Setting Global Variables

The subset control program can use global variables to access information about the current subset. Table 3–1 lists these variables.

Table 3–1: STL_ScpInit Global Variables

Variable	Description
_SUB	Subset identifier, for example, OATODB100
_DESC	Subset description, for example, Document Builder Tools
_PCODE	Product code, for example, OAT
_VCODE	Version code, for example, 100
_PVCODE	Concatenation of product code and version code, for example, OAT100
_PROD	Product description, for example, Orpheus Document Builder
_ROOT	The root directory of the installation
_SMDB	The location of the subset control files, ./usr/.smdb.
_INV	The inventory file, for example, OATODB100.inv
_CTRL	The subset control file, for example, OATODB100.ctrl
_OPT	The directory specifier /opt/

Table 3–1: STL_ScpInit Global Variables (cont.)

Variable	Description
_ORGEXT	File extension for files saved by the STL_LinkCreate routine, set to pre$_PVCODE
_OOPS	The NULL string, for dependency checking

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. Use STL_ScpInit in all phases except the M phase to initialize global variables. All variable names begin with an underscore (_) for easy identification.

3.1.6 Working in a DMS Environment

In a Dataless Management Environment (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. Sharing software across the network saves disk space on each of the computers in the network.

Note

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 is the default path for subset control program processing commands to be run from the server in a DMS environment:

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

Refer to *Sharing Software on a Local Area Network* for more information about DMS.

A subset control program may need to perform differently in a dataless environment or disallow installation of the subset on such a system. In particular, you should be concerned with the following issues when writing a subset control program for installing in a dataless environment:

- If the product will be installed onto a DMS server, the subset control program should not specify absolute pathnames. Otherwise, the `setld` utility will install the product into a dataless area of `/var/adm/dms/dmsx.alpha` rather than `root (/)`, as if it were installing onto the system itself.
- When running on a dataless client, the `/usr` area is not writable. Therefore, you should not let the subset control program or the product itself attempt to write to the `/usr` area during the `C INSTALL` phase.

You can use the following routines to handle 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.

3.2 SCP Tasks Associated with Installation Phases

The `setld` utility calls the subset control program at the beginning of each phase. Before calling the subset control program, the `setld` utility sets the `ACT` environment variable to a value that indicates the current phase. The subset control program uses this variable to determine what action to take. You can write the subset control program as a series of `case` statements, where each statement handles one phase.

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

The following sections describe the tasks that a subset control program may take in each phase.

3.2.1 Displaying the Subset Menu (M Phase)

At the beginning of an installation, the `setld` utility presents a menu of subsets that it can install. Before displaying the menu, it 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.

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

- `-l` indicates that the operation is a subset load.

- -x is reserved for extraction of the subset into a remote installation services (RIS)server's product area.

For example, during this phase the subset control program can issue the machine command to verify that the subset is being installed on the correct hardware platform. If the command returns a nonzero status, the subset control program exits with a nonzero status.

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 do nothing during the M phase. When loading subsets, it should make this machine test. The following Bourne shell example illustrates one way to code the M phase. In Example 3–1, the subset control program is checking to determine the type of processor on which it is running.

Example 3–1: Subset Control Program Test for Machine During M Phase

```
case $ACT in
  M)
    case $1 in
      -l)
        [ "'./bin/machine'" = alpha ] || exit 1
        ;;
    esac
    ;;
.
.
.
esac
```

Installation for a dataless client requires that the client's local copy of the machine command be used even though the installation is being performed in the dataless area on a different platform. Because the machine command is a shell script, it can be executed on any platform.

3.2.2 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 for each subset. At this time, the subset control program can take any action required to protect existing files.

Caution

Overwriting base operating system files can cause the following problems:

- Your product will 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 with the new version of the file.
- Overwriting a base operating system file can prevent an Update Installation from completing successfully and may render the system unusable.
- The user could be forced to remove your product from the system as a part of an Update Installation process. The user would then have to reinstall your product after the Update Installation has completed.

The subset control program should also check for subset dependencies at this time. A **subset dependency** is a condition under which a subset depends on the existence of one or more other subsets. Because the `setld` utility can install and remove subsets, a system administrator could attempt to remove one or more subsets on which your product depends. Because those subsets do not in turn depend on your product's subsets, the `setld` utility usually removes them without question, leaving your product unusable or disabled. You can prevent this inadvertent destruction of your product's environment by **locking** the subsets on which your subset depends. Subset locking can occur during the `POST_L` phase (see Section 3.2.3.2).

To make dependency management easier to implement, a set of routines is provided in the form of Bourne shell script code. These routines are located in the `/usr/share/lib/shell/libscp` file.

The dependency management routines use dependency expressions to examine conditions on the system. A **dependency expression** is a postfix logical expression that describes the conditions on which the subset depends. Dependency expressions are recursive left to right and are processed using conventional postfix techniques. Dependency expressions are defined in Backus-Naur form, as follows:

```
depexp ::= wc_subset_id
         | depexp not
         | depexp depexp and
         | depexp depexp or
```

Table 3–2 lists the elements of a dependency expression (*depexp*):

Table 3–2: Elements of a Dependency Expression

Element	Description
wc_subset_id	Represents a subset identifier that can contain file name expansion characters (asterisks, question marks, or bracketed sets of characters), for example, as in `OAT[RV]DOA*2??`.
and operator	Requires two dependency expressions. The dependency is satisfied if both expressions are satisfied.
or operator	Requires two dependency expressions. The dependency is satisfied if at least one of the expressions is satisfied.
not operator	Requires one dependency expression. The dependency is satisfied if the expression is not satisfied.

The following are valid dependency expressions:

```
SUBSETX??0
SUBSETY200 not
SUBSET[WX]100 SUBSETY200 and
SUBSETX100 SUBSETY200 or
SUBSETX100 SUBSETY200 and SUBSETZ300 or not
```

The last of these expressions evaluates as follows:

- The `and` operator is satisfied if both `SUBSETX100` and `SUBSETY200` are present.
- The `or` operator is satisfied if the `and` operator was satisfied or if `SUBSETZ300` is present.
- The `not` operator is satisfied only if the combination of `SUBSETX100` and `SUBSETY200` is not present and `SUBSETZ300` is not present.

You can call the following routines to perform dependency checking:

`STL_DepInit`
: Establishes objects that the `STL_DepEval` routine uses. Before you use `STL_DepEval` to check your subset's dependencies, you must execute `STL_DepInit` once. This routine has no arguments and returns no status.

`STL_DepEval` *depexp*
: Evaluates the dependency expression that you specify as an argument. You can use as many invocations of `STL_DepEval` as you need to verify that all your subset dependencies are met.

3.2.3 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, such as moving them to a different location. The subset control program should create links and perform subset dependency locking at this time.

Sometimes you may need to create links within your product's directories in the layered product areas 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 in the `ln` commands for 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, all the backward links are made in the server's kit area. Then, when that area is exported to clients, the links are already in place for the client. You do not need to create any backward links in the client area.

Note

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

3.2.3.1 Creating Backward Links

The subset control program should create backward links 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.

Use the `STL_LinkInit` and `STL_LinkBack` routines to create backward links as follows, and use the `rm` shell 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 `/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 3–2 uses STL_LinkInit and STL_LinkBack in the POST_L phase to create a link named /usr/opt/OAT100/lib/odb_users that refers to the real file /etc/odb_users, and removes the link in the PRE_D phase.

Example 3–2: Backward Link Creation

```
#! /sbin/sh

case $ACT in
:
  POST_L)
    STL_LinkInit
    STL_LinkBack odb_users ./etc ./usr/opt/OAT100/lib
    ;;

  PRE_D)
    rm -f ./usr/opt/OAT100/lib/odb_users
    ;;
esac
```

3.2.3.2 Locking Subsets

Every subset in the system's inventory has two lock files:

- A lock file named *subset-id*.lk indicates successful installation of a subset
- A lock file named *subset-id*.dw indicates failed corrupt installation of a subset

When it installs a subset, the setld utility creates one of these two lock files. At that time, the lock file is empty. Assuming successful installation, that subset is then available for dependency checks and locking performed on behalf of subsets installed later. A subset's lock file can 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 OATODB100 subset from the ODB kit, it inserts a record that contains the subset identifier OATODB100 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 OATODB100 depends on OATBASE200. Then the setld utility displays a warning message with this information and requires confirmation that the user really intends to remove the OATBASE200 subset.

If the administrator removes the OATODB100 subset, the setld utility removes the corresponding record from the OATBASE200.lk file. Thereafter, the administrator can remove OATBASE200 without causing a dependency warning.

You can call the following routines to lock subsets:

STL_LockInit
: Used in the POST_L and PRE_D phases to establish objects for the STL_DepLock and STL_DepUnLock routines. Before you use STL_DepLock or STL_DepUnLock to manipulate subset locks, you must execute STL_LockInit once. Because locking and unlocking are managed by different invocations of your subset control program, STL_LockInit must appear in both the POST_L and PRE_D phases. You should code two instances of STL_LockInit rather than calling it once before you make a decision based on the value of the ACT environment variable. This routine has no arguments and returns no status.

STL_DepLock *subset depexp* ...
: Used in the POST_L phase to add the new subset's name to the lock lists for each of the subsets named as arguments. (You can use dependency expressions as arguments.) The name of the new subset is the first argument to STL_DepLock. For example, the following call to STL_DepLock places OATODB100 in the OATTOOLS100.lk and OATBASE2??.lk files:

```
STL_DepLock OATODB100 OATTOOLS100 OATBASE2??
```

3.2.4 After Securing the Subset (C INSTALL Phase)

After securing the subset, the setld utility sets the ACT environment variable to C 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 would statically or dynamically configure a device driver at this point. The subset control program cannot create a layered product's symbolic links at this time.

The setld utility enters this phase at the following times:

- When the user invokes it with the -c option

- When the user invokes it with the -l option and without the -D flag to specify an alternate root (/) directory

The setld utility does not pass through this phase if the user loads the subset and specifies an alternate root directory with the -D flag.

3.2.5 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.

The setld utility checks for the existence of the installed subset. If the user has invoked the setld utility with the -l option and the installed subset exists, the setld utility verifies the size and checksum information for each file in the subset during loading. The setld utility does not call the subset control program V Phase during the installation process.

If the subset's subset control program includes an installation verification program (IVP), the IVP is executed. However, in a kit that contains multiple subsets, the last subset control program called could execute an IVP (or a suite of IVPs) to ensure that the product works properly.

3.2.6 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 subset control program cannot remove a layered product's links at this time.

3.2.7 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, or restoring moved files to their default installation locations so that the setld utility can delete them properly. A return status of 0 (zero) allows the delete operation to continue.

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.

STL_DepUnLock *subset depexp ...*
: Removes the new subset's name from the lock lists for each of the subsets named as arguments.

3.2.8 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.

3.3 Subset Control File Flag Bits

As explained in Table 4–3, you can use bits 8 to 15 of the subset control file's flags field to specify special subset-related information. The subset control program can read these bits from the subset control file (files with a .ctrl suffix) into which this information was placed when the kit was built. During an installation, the setld utility moves the subset control file to the ./usr/.smdb. directory, where the subset control program can read the file as needed.

Not all subset control programs need to use the subset control file. It can be a convenient way to pass information between subsets, if such communication is necessary.

Caution

If you must use the subset control file, be extremely careful. Bits 0 through 7 of the flags field are reserved by the setld utility; do not use or modify these bits in any way.

To find the current settings of the flags field, the subset control program should read the subset control file, looking for a line that lists the settings. For example, the OATODBDOC100.ctrl file contains the following line:

```
FLAGS=1
```

The value of the flags field is expressed as a decimal integer. You can use the BitTest shell routine, contained in the file

/usr/share/lib/shell/BitTest, to test an individual bit. The following example tests bit 11 of the flags field for the OATODBDOC100 subset:

Example 3–3: Using the BitTest Routine to Test Bits

```
#! /sbin/sh

. /usr/share/lib/shell/BitTest

flags='sed -n '/FLAGS=/s///p' usr/.smdb./OATODBDOC100.ctrl'
BitTest $flags 11 && {
:
}
```

3.4 Creating a Subset Control Program for a User Product

Example 3–4 shows a subset control program for the ODB user product. This program illustrates one correct method to obtain the value of the ACT environment variable. It uses the value of the variable to determine what actions to perform, as follows:

- During the PRE_L phase, performs dependency checking to make sure the base tools are already installed.
- During the POST_L phase, creates symbolic links and locks subsets on which it depends.
- During the C INSTALL phase, notifies the user that installation is complete.
- During the PRE_D phase, removes symbolic links.
- During the POST_D phase, unlocks subsets.

The program does not handle the V phase or the C DELETE phase. When the setld utility invokes the program at these times, the program exits with a success status.

Example 3–4: Subset Control Program for the ODB User Product

```
#!/sbin/sh
#
# Subset Control Program for OATODB??? subset

# INCLUDE SCP LIBRARY FUNCTIONS

[ '/bin/machine' = alpha ] &&
    . /usr/share/lib/shell/libscp [1]

# BEGIN EXECUTION HERE
```

Example 3–4: Subset Control Program for the ODB User Product (cont.)

```
case $ACT in 2

M) 3
    case $1 in
    -l)
        # hardware platform check
        [ "'./bin/machine'" = alpha ] || exit 1
        ;;
    esac
    ;;

PRE_L) 4
    # dependency checking
    STL_ScpInit
    STL_DepInit

    STL_DepEval ${_PCODE}TOOLS??? ||
    {
        _OOPS="$_OOPS
    Orpheus Document Builder Tools (${_PCODE}TOOLS)"
    }

    STL_DepEval ${_PCODE}BASE[2-9]?? ||
    {
        _OOPS="$_OOPS
    Orpheus Document Builder Base Tools, Version 2.0 or later (${_PCODE}TOOLS)"
    }

    [ "$_OOPS ] &&
    {
        echo "
The $_DESC requires the existence of
the following uninstalled subset(s):
$_OOPS

Please install these subsets before retrying the installation.
" >&2
        exit 1
    }
    ;;
POST_L) 5
    # create symbolic links
    STL_ScpInit

    # dependency locking
    STL_LockInit
    STL_DepLock $_SUB ${_PCODE}TOOLS??? ${_PCODE}BASE[2-9]?? and
    ;;
C) 6
    STL_ScpInit
    case $1 in
    INSTALL)
        echo "
Installation of the $_DESC ($_SUB)
subset is complete.

Before using the tools in this subset, please read the README.odb
file located in the /usr/lib/br directory for information on the
kit's contents and for release information.
```

Example 3–4: Subset Control Program for the ODB User Product (cont.)

```
"
        ;;
    esac
    ;;
PRE_D) [7]
    # remove symbolic links
    STL_ScpInit

    # dependency unlocking
    STL_LockInit
    STL_DepUnLock $_SUB ${_PCODE}TOOLS??? ${_PCODE}BASE[2-9]?? and
    ;;

esac

exit 0  [8]
```

[1] Reads in the subset control program library routines if the installation is running on an Alpha platform.

[2] Examines the ACT environment variable to select the action the subset control program takes when called by the setld utility.

[3] For the M phase, allows the setld utility to continue if the installation is running on an Alpha platform. If not, the subset control program returns a nonzero status and exits. As a result, the setld utility does not present this subset in its menu of subsets to be installed.

[4] During the PRE_L phase, ensures that subsets on which the OATODB100 subset depends are installed. If they are not installed, the subset control program describes the missing subsets and returns a nonzero status to the setld utility, which stops the installation of this subset. If multiple subsets are being installed, each is treated individually. The $_PCODE, $_OOPS, and $_DESC variables are defined by the STL_ScpInit routine.

[5] During the POST_L phase, creates symbolic links from the subset by invoking the STL_ScpInit routine. After creating the links, the subset control program secures the subset by locking the subsets on which it depends to ensure that they are not deleted without warning the user of potential problems. The subset control program uses the $_SUB and $_PCODE global variables to define the subsets in the dependency relationship.

[6] During the C phase, checks to see if the argument passed in by the setld utility has the value of INSTALL. If so, the program displays a message indicating that the installation is complete. It uses STL_ScpInit and global variables to substitute the product description ($_DESC) and subset ID ($_SUB) within the message text.

7 During the PRE_D phase, calls the STL_ScpInit routine to remove symbolic links and calls the STL_LockInit and STL_DepUnLock routines to unlock the subsets on which OATODB100 depends. The $_SUB variable is defined by the STL_ScpInit routine.

8 Ensures that the subset control program returns a success status to the setld utility for each successful action and for all of the possible cases that the subset control program does not handle. Do not code exit 0 statements elsewhere in your subset control program.

3.5 Creating a Subset Control Program for a Kernel Product

In addition to the optional processing described in Section 3.4, a subset control program for a kernel product (or hardware product) such as a device driver must also configure the driver into the kernel. When building subset control programs for a kernel product, such as a device driver, 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 3–5 shows the subset control program for the single binary module associated with the /dev/none 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 3–5: Subset Control Program for the /dev/none Driver

```
#!/sbin/sh
#
#
#  NONE.scp - Install the files associated with the /dev/none
#  device driver.  This driver, implemented as a single binary
#  module (.mod file), can be statically or dynamically configured
#  into the kernel.
#

case "$ACT" in 1
```

Example 3–5: Subset Control Program for the /dev/none Driver (cont.)

```
C)
    case $1 in
    INSTALL)  2
        echo "***** /dev/none Product Installation Menu *****"
        echo "*****                                        *****"
        echo "1. Install the static device driver subset."
        echo "2. Install the dynamic device driver subset."

        echo" Type the number for your choice [] "

        read answer
        case ${answer} in
            1)  3
            # Register the files associated with the static
            # /dev/none device driver product.
            kreg -l EasyDriverInc ESANONESTATIC100 /usr/opt/ESA100 4

            # Merge the files associated with the statically configured
            # /dev/none device driver product to the customer's
            # /etc/sysconfigtab database
            sysconfigdb -m -f /usr/opt/ESA100/sysconfigtab none 5

            echo "The rest of the procedure will take 5-15 minutes"
            echo "to rebuild your kernel, depending on the processor"
            echo "type."
            echo ""
            echo "Starting kernel rebuild... "
            if doconfig -c $HOSTNAME 6
                then
                echo "Kernel built successfully"
            else
                1>&2 echo "Error building kernel."
                return 1
            fi
            ;;

            2)  7
            # Merge the files associated with the dynamically configured
            # /dev/none device driver product to the customer's
            # /etc/sysconfigtab database
            sysconfigdb -m -f /usr/opt/ESA100/sysconfigtab none 8

            # Copy the none.mod file to the /subsys directory. Create
            # the none.mth driver method by linking to device.mth
            # /subsys/none.mth -> /subsys/device.mth
            cp /usr/opt/ESA100/none.mod /subsys/none.mod 9
            ln -s /subsys/device.mth /subsys/none.mth 10

            # Load the /dev/none device driver and create the device
            # special files
            sysconfig -c none 11

            echo "The /dev/none device driver was added to your
            echo "/etc/sysconfigtab database." 12
            ;;
        esac
        ;;

    DELETE) 13
        echo "***** /dev/none Product Installation Menu *****"
```

Example 3–5: Subset Control Program for the /dev/none Driver (cont.)

```
            echo "*****                                    *****"
        echo "1. Delete the static /dev/none device driver subset."
        echo "2. Delete the dynamic /dev/none device driver subset."

        echo" Type the number for your choice [] "

        read answer
        case ${answer} in
            1)
            kreg -d ESANONESTATIC100 [14]

            # Delete the /dev/none device driver's entry from the
            # /etc/sysconfigtab database

            sysconfigdb -d none [15]
            echo "The rest of the procedure will take 5-15 minutes"
            echo "to rebuild your kernel, depending on the processor"
            echo "type."
            echo ""
            echo "Starting kernel rebuild... "
            if doconfig -c $HOSTNAME [16]
                then
                echo "Kernel built successfully"
            else
                1>&2 echo "Error building kernel."
                return 1
            fi
            ;;

            2)
            # Make sure the /dev/none device driver is not currently
            # loaded
            sysconfig -u none  [17]

            # Delete the /dev/none device driver's entry from the
            # /etc/sysconfigtab database
            sysconfigdb -d none [18]
             ;;
        esac
        ;;
    esac
    ;;
esac
exit 0
```

[1] Examines the ACT environment variable to select the action the subset control program should take.

[2] Displays a menu of installation options during the C INSTALL phase. The user can choose to install the driver for static configuration or dynamic configuration.

[3] Performs a static configuration, if the user chooses menu item 1.

[4] Invokes the kreg utility to register the driver files with the kernel. The kreg utility registers a device driver product by creating the

/usr/sys/conf/.product.list file on the customer's system. This file contains registration information associated with the static device driver product. The subset control program calls kreg with the following arguments:

- The -l flag

  This flag indicates that the subset was loaded, and it directs kreg to register the device driver product as a new kernel extension.

- Company name

  The company name is EasyDriverInc. The kreg utility places this name in the company name field of the customer's /usr/sys/conf/.product.list file.

- Software subset name

  The software subset name for this device driver product is ESANONESTATIC100. The subset name consists of the product code, subset mnemonic, and three-digit version code. The kreg utility extracts information from the specified subset data and loads it into the customer's /usr/sys/conf/.product.list file.

- Directory name

  The directory on the customer's system where kreg copies the files associated with this driver product is /usr/opt/ESA100. The kreg utility places this directory in the driver files path field of the customer's /usr/sys/conf/.product.list file.

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

5 Adds the sysconfigtab file fragment for the statically configured driver to the system's /etc/sysconfigtab database by calling the sysconfigdb utility with the following arguments:

- The -m flag

  This flag causes the sysconfigdb utility to merge the device driver entry to the customer's /etc/sysconfigtab database.

- The -f flag

  This flag precedes the name of the sysconfigtab file fragment whose device driver entry is to be added to the /etc/sysconfigtab database. This flag is used with the -a flag.

- The sysconfigtab file fragment

  The kit developer at EasyDriver, Inc. specifies the path /usr/opt/ESA100/sysconfigtab to indicate the location of the sysconfigtab file fragment for the /dev/none device driver.

- Device driver name

The kit developer at EasyDriver, Inc. specifies none as the name of the driver whose associated information is added to the /etc/sysconfigtab database. This name is obtained from the entry_name item of the sysconfigtab file fragment, as described in *Writing Device Drivers: Tutorial*.

[6] Runs the doconfig utility to configure the driver into the kernel. The subset control program returns an error if doconfig fails for any reason.

[7] Performs a dynamic configuration if the user chooses menu item 2.

[8] Calls the sysconfigdb utility to merge the driver's sysconfigtab file fragment to the system's /etc/sysconfigtab database.

[9] Copies the dynamically configured driver's single binary module (.mod file) to the /subsys directory.

[10] Creates a symbolic link from the /subsys/device.mth file to the driver's /subsys/none.mth file.

[11] Calls the sysconfig utility with the -c option to reconfigure the system and include the /dev/none driver. The -c option causes the sysconfig utility to dynamically configure the driver into the running system and to create device special files. The name of the driver as specified in the sysconfigtab file fragment follows the option.

[12] Displays a message notifying the user that the driver has been added to the system.

[13] Displays a menu of options for deleting subsets during the C DELETE phase. The user must tell the setld utility whether the subset to be deleted represents a statically configured driver or a dynamically configured driver. The way the driver was configured determines how the driver is deleted.

[14] Calls the kreg utility to deregister the driver with the kernel, if the user chooses menu option 1 (delete a statically configured driver). When the kreg utility is called with the -d flag, it deletes the entry for the specified layered product from the customer's /usr/sys/conf/.product.list file. In this case, the layered product is the /dev/none driver, represented by the ESANONESTATIC100 subset identifier.

[15] Calls the sysconfigdb utility with the -d flag, which deletes the static /dev/none device driver from the customer's /etc/sysconfigtab database.

[16] Runs the doconfig utility to reconfigure the kernel. The subset control program returns an error if doconfig fails for any reason.

[17] Calls the sysconfig utility with the -u flag to deconfigure the dynamically configured /dev/none device driver from the running

system if the user chooses menu item 2 (delete a dynamically configured driver).

18 Calls the `sysconfigdb` utility with the `-d` flag to delete the dynamically configured `/dev/none` device driver from the customer's `/etc/sysconfigtab` database.

4

Building Subsets and Control Files

In a kit, a subset is the smallest installable entity that is compatible with the `setld` utility. It is up to you, the kit developer, to specify how many subsets your kit has and what files each contains. A good practice is to group files by related function or interdependence.

This chapter also describes how to create the master inventory and key files for the ODB product and how to use the `kits` utility to create the subsets and subset control files. You perform the same steps when creating subsets for user products, kernel products, and hardware product kits.

The following list summarizes the steps a kit developer must follow to build subsets and associated control files:

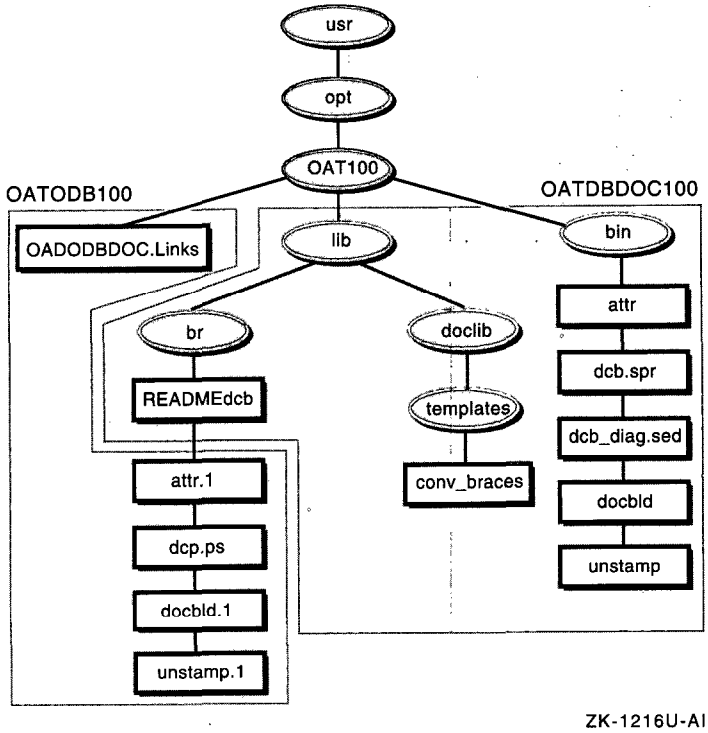
1. Organize product files into subsets.
2. Create a master inventory file which contains information about each file in the subset.
3. Create a key file to define product attributes such as the product name, product version, and subset definitions.
4. Optionally create a subset control program (SCP) to perform special tasks that are beyond the basic installation tasks managed by the `setld` utility. SCPs are documented in Chapter 3.
5. Run the `kits` utility to create the subsets and related control files.

4.1 Grouping Files into Subsets

The fictitious user product ODB requires two subsets. The subset named `OATODB100` contains the files needed to run the product. The subset named `OATODBDOC100` contains documentation and online help files. By placing the documentation in a separate subset, the customer's system administrator can choose not to install the documentation subset if space is limited on the system.

Figure 4–1 shows how the files that make up the ODB product are grouped into subsets. As the figure shows, the physical location of a file is not necessarily a factor in determining the subset to which it belongs.

Figure 4–1: Subsets and Files in the ODB kit



4.2 Creating the Master Inventory File

After deciding upon subset names and 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 /dcb_tools/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, you can use the `newinv` command. Specify the file name of the empty master inventory file and the pathname of the source hierarchy's top-level directory. For example:

```
% newinv OAT100.mi ../data
```

This command invokes `newinv` on the master inventory file for the ODB product. It specifies the pathname to the source hierarchy as a relative path from the current directory (`data`).

The `newinv` utility produces a list of files that are present in the source hierarchy and places you in the `vi` editor, or the editor specified by your `EDITOR` environment variable, so that you can make the required changes. Remove the entries for any files that should not appear on the kit, and add the flags, pathname, and subset identifier for each entry that should appear on the kit.

Note

- Use extreme care when editing the master inventory file; you must separate fields in this file with a single **tab** character, not a space. File names must not contain spaces or tabs.
- The files listed in the master inventory file are given dot-relative pathnames. The `setld` utility usually works from the system's root (`/`) directory, but the user can specify an alternate root directory with the `-D` option. For this reason, you should not use absolute pathnames in the master inventory file.

The master inventory file contains one record for each file in the kit. Each record in the master inventory file consists of three fields, described in Table 4–1.

Table 4–1: Fields in the Master Inventory File

Field	Description
Flags	A 16-bit unsigned integer. Bit 1 is the `v` (volatility) bit. When set, changes to the existing copy of the file can occur during kit installation. It usually is set for files such as `usr/spool/mqueue/syslog`. Bit 2 is the `l` (link) bit. When set, the `STL_LinkCreate` routine invoked in the subset control program (`.scp`) creates a forward link from the standard system directories to the layered product `opt` areas. The remaining bits are reserved; possible values for this field are therefore 0, 2, 4, or 6.

Table 4–1: Fields in the Master Inventory File (cont.)

Field	Description
Pathname	The dot-relative (. /) pathname of 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.

Example 4–1 shows that the ODB kit has two subsets. The OATODB100 subset contains utilities and libraries and must be installed to use the product. The OATODBDOC100 subset contains the product's documentation which is not required to run the product.

Example 4–1: Sample Master Inventory File for the ODB Kit

```
0       .       RESERVED
0       ./usr/opt       RESERVED
0       ./usr/opt/OAT100/OATODBDOC.Links        OATODBDOC100
0       ./usr/opt/OAT100/bin    OATODB100
4       ./usr/opt/OAT100/bin/attr       OATODB100
4       ./usr/opt/OAT100/bin/dcb.spr    OATODB100
4       ./usr/opt/OAT100/bin/dcb_defaults       OATODB100
4       ./usr/opt/OAT100/bin/dcb_diag.sed       OATODB100
4       ./usr/opt/OAT100/bin/docbld     OATODB100
4       ./usr/opt/OAT100/bin/unstamp    OATODB100
0       ./usr/opt/OAT100/lib    OATODB100
0       ./usr/opt/OAT100/lib/br OATODB100
4       ./usr/opt/OAT100/lib/br/README.dcb      OATODB100
4       ./usr/opt/OAT100/lib/br/attr.1  OATODBDOC100
4       ./usr/opt/OAT100/lib/br/dcb.ps  OATODBDOC100
4       ./usr/opt/OAT100/lib/br/docbld.1        OATODBDOC100
4       ./usr/opt/OAT100/lib/br/unstamp.1       OATODBDOC100
0       ./usr/opt/OAT100/lib/doclib     OATODB100
0       ./usr/opt/OAT100/lib/doclib/templates   OATODB100
4       ./usr/opt/OAT100/lib/doclib/templates/conv.braces       OATODB100
.
.
.
```

In the example, the ./usr/opt directory has the RESERVED subset identifier, indicating that the setld utility should not allow the directory to be overwritten if it exists on the customer's system. The Flags field is set to 0 (zero), indicating that this directory cannot change and that it is not linked to another directory on the customer's system. On the other hand, the /usr/opt/OAT100/bin/attr file has the OATODB100 subset identifier, indicating that the file belongs in the specified subset. The Flags

field is set to 4, indicating that the file may change and that it has a link to another file on the customer's system.

For subsequent updates to the kit, use the existing version of the master inventory file for the input file. The `newinv` utility performs the following additional steps:

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

4.3 Creating the Key File

The key file identifies the product on the product kit and includes the product name and version number and the name of the master inventory file for the kit. You create this file in the `data` directory with the text editor of you choice. The key file name must consist of the product code and version, with the letter `k` as a suffix. For example, `OAT100.k` is the key file for the ODB kit. Example 4–2 illustrates this key file.

Example 4–2: Key File for the ODB Kit

```
#       Product-level attributes
#
NAME='Orpheus Document Builder'
CODE=OAT
VERS=100
MI=OAT100.mi
COMPRESS=1
#
#       Subset definitions
#
%%
```

Example 4–2: Key File for the ODB Kit (cont.)

```
OATODB100          .          0        'Document Builder Tools'
OATODBDOC100     OATODB100|OSFDCMT440     2        'Document Builder Documentation'
```

As shown in Example 4–2, the key file is divided into two sections separated by a line that contains two percent signs (`%%`):

- 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 4–2. Each attribute name is separated from its value by an equal sign (`=`). You can include comment lines, which begin with a number sign (`#`).
- 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 4–3 describes the subset descriptor fields. In Example 4–2, the `OATODB100` subset is mandatory; its Flags field is set to 0 (zero). The `OATODBDOC100` Document Builder Documentation subset in Example 4–2 is optional; its Flags field is set to 2 (two). The `OATODBDOC100` subset is dependent on both the `OATODB100` Document Builder Tools subset, part of the `ODB` kit, and the `OSFDCMT440` Text Processing subset, which is part of the base operating system.

Table 4–2: 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 product code that consists of three characters, for example, `OAT`. The first character must be a letter. In this book, `OAT` is the three character code assigned to the *Orpheus Authoring Tools, Inc.* development company. This code cannot contain more than three characters. The product code is assigned by Compaq. Send mail to the `Product@DSSR.enet.dec.com` electronic mail address to obtain a product code. *Note*: The first three letters of a subset name must be the same as the product code.

Table 4–2: Key File Product Attributes (cont.)

Attribute	Description
	Several of these 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.
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 path to it. 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 assign 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).

Table 4–3: Key File Subset Descriptor Fields

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`). All letters in the subset identifier must be uppercase.
Dependency list	Either a list of subsets upon which this subset is dependent (`OATODB100|OSFDCMT440`), or a single period (`.`) indicating that there are no subset dependencies. If there is more than one subset dependency, separate them with the pipe character (|).

Table 4–3: Key File Subset Descriptor Fields (cont.)

Field	Description
Flags	A 16-bit unsigned integer. The operating system defines the use of the lower 8 bits. Set bit 0, the sticky bit, to indicate that the subset cannot be removed. Set bit 1 to indicate that the subset is optional. Bits 2-7 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.

4.4 Running the kits Utility

After you create the master inventory and key files, you create subsets and control files by running the `kits` utility. This command requires three arguments:

- Key file name
- Pathname for the source hierarchy
- Pathname for the output hierarchy

Note

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

For example, the following command builds the subsets for the ODB product kit:

```
% cd /dcb_tools/data
% kits OAT100.k ../data ../output
```

The `kits` utility performs the following steps and reports its progress:

1. Creates the subsets.
2. Compresses each subset, if you specify the `COMPRESS` attribute in the key file.

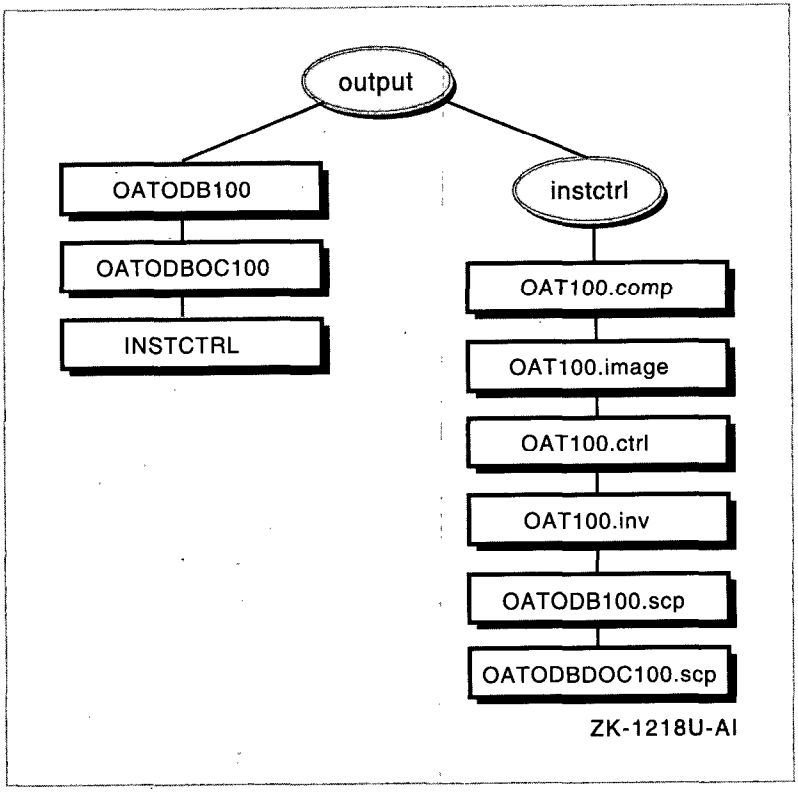
3. Creates the installation control files listed in Table 4–4 and places them in the `instctrl` directory.
4. Creates the `instctrl` file, which contains a `tar` image of all the installation control files. This file is placed in the output directory.

Table 4–4: 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.
subset-id.ctrl	Subset control file. This file contains the `setld` utility control information. There is one subset control file for each subset.
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.
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. For more information on `.scp` files, refer to Chapter 3.

Figure 4–2 shows the contents of the `output` directory after the `kits` utility has run.

Figure 4–2: Contents of the ODB output Directory



The subset files and the installation control (`instctrl`) file are constituents of the final kit. The following sections describe the contents of the installation control files in detail.

4.4.1 Compression Flag File

The `setld` utility uses the presence of the compression flag file (*product-id*`.comp`) to determine whether the subset files are compressed. The compression flag 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`.

Note

Do not compress subsets on hardware product kits.

4.4.2 Image Data File

The `setld` utility uses the image data file to verify that the subset images it loads from the installation media are uncorrupted before the actual installation process begins. The image data file name consists of the product's unique three-letter name with the string `image` for a suffix. The image data file contains one record for each subset in the kit. The following example illustrates `OAT.image`, the image data file for the ODB kit:

```
15923     70 OATODB100
24305    400 OATODBDOC100
```

Table 4–5 describes the three fields in each record.

Table 4–5: Image Data File Fields

Field	Description
Checksum	The modulo-65536 (16-bit) checksum of the subset file (after compression, if the file is compressed)
Size	The size of the subset file in kilobytes (after compression, if the file is compressed)
Subset identifier	The product code, subset mnemonic, and version number

4.4.3 Subset Control Files

The `setld` utility uses the subset control files as a source of descriptive information about subsets. A control file for each subset contains the following fields:

- `NAME`

  Specifies the product name.

- `DESC`

  Specifies a brief description of the subset.

- `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.

- `VARSIZE`

  Specifies (in bytes) the space the subset requires in the `var` file system.

- `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).

- 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 4–4).

- DEPS

  Specifies either a list of subsets upon which this subset is dependent (`DEPS="OATODB100 OSFDCMT440"`), 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 (see Example 4–3).

- FLAGS

  Specifies the value in the flags field of the subsets record in the key file. Bit 0 is the sticky bit which indicates that the subset cannot be removed. Bit 1 indicates that the subset is optional. Bits 2 to 7 are reserved; bits 8 to 16 are undefined.

The following example illustrates `OATODBDOC100.ctrl`, the control file for the ODB kit's `OATODBDOC100` subset:

Example 4–3: Sample Subset Control File

```
NAME='Orpheus Document Builder'
DESC='Document Builder Documentation'
ROOTSIZE=0
USRSIZE=522090
VARSIZE=0
NVOLS=1:2
MTLOC=1:1
DEPS="OATODB100 OSFDCMT440"
FLAGS=1
```

4.4.4 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. Example 4–4 shows the inventory file, `OATODBDOC100.inv`, for the ODB kit's `OATODBDOC100` subset.

Note

The backslashes (\) in this example indicate line continuation and are not present in the actual file.

Example 4–4: Sample Subset Inventory File

```
4       983     01851   1065    0       100644  3/21/99 100     f\
        ./usr/opt/OAT100/lib/br/attr.1  none    OATODBDOC100
4       424997  63356   1065    10      100644  4/15/99 100     f\
        ./usr/opt/OAT100/lib/br/dcb.ps  none    OATODBDOC100
4       7283    03448   1065    10      100644  4/15/99 100     f\
        ./usr/opt/OAT100/lib/br/docbld.1        none    OATODBDOC100
4       6911    37501   1065    0       100644  3/21/99 100     f\
        ./usr/opt/OAT100/lib/br/docbld.5        none    OATODBDOC100
4       985     41926   1065    0       100644  3/21/99 100     f\
        ./usr/opt/OAT100/lib/br/unstamp.1       none    OATODBDOC100
```

Each record of the inventory is composed of 12 fields, each separated by single tab characters. Table 4–6 describes the contents of these fields.

Table 4–6: Subset Inventory Field Descriptions

Field	Name	Description
1	Flags	A 16-bit unsigned integer. Bit 1 is the `v` (volatility) bit. When set, changes to the existing copy of the file can occur during kit installation. It usually is set for files such as `usr/spool/mqueue/syslog`. Bit 2 is the `l` (link) bit. When set, the `STL_LinkCreate` routine creates a forward link from the standard system directories to the layered product areas. The remaining bits are reserved; possible values for this field are therefore 0, 2, 4, or 6.
2	Size	The actual number of bytes in the file.
3	Checksum	The modulo-65536 (16-bit) checksum of the file.
4	uid	The user ID of the file's owner.
5	gid	The group ID of the file's owner.

Table 4–6: Subset Inventory Field Descriptions (cont.)

Field	Name	Description
6	Mode	The six-digit octal representation of the file's mode.
7	Date	The file's last modification date.
8	Revision	The version code of the product that includes the file.
9	Type	A letter that describes the file: `b` – Block device. `c` – Character device. `d` – Directory containing one or more files. `f` – Regular file. For regular files with a link count greater than one, see file type `l`. `l` – Hard link. Other files in the inventory have the same inode number. The first (in ASCII collating sequence) is listed in the referent field. `p` – Named pipe (FIFO). `s` – Symbolic link.
10	Pathname	The dot-relative (`./`) pathname of the file.
11	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`.
12	Subset identifier	The name of the subset that contains the file.

5

Hardware Product Kits

A hardware product kit is developed to deliver software support for hardware on a customer's system. This chapter describes how to prepare a hardware product kit, the additional files required for the kit, and how to test the installation of the kit.

A hardware product kit includes the kernel modules that let your operating system support new or upgraded hardware, and it enables you to install hardware support without reinstalling or updating the 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 Version 4.0F of the operating system before installing the hardware product kit software.

The kernel modules and the kit support files are distributed on CD-ROM as a 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).

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 structure as described in Section 2.1.
3. Populate the source directory as described in Section 2.2 and read about the file considerations for hardware product kits in Section 2.2.2.
4. Read Chapter 3 for information about creating subset control programs.
5. Read Chapter 4 to group files into subsets, create the **master inventory** and **key** files, and build files into subsets.
6. Create the additional files required for hardware product kits as described in Section 5.1.
7. Create a subset control program as described in Section 5.2.
8. Create the kit distribution media as described in Section 5.3.
9. Test the hardware product kit as described in Section 5.4.

5.1 Additional Files Required for Hardware Product Kits

A hardware product kit requires that you create the following files on the distribution media to make the hardware product accessible during initial system installation and bootstrap:

- A *name*.kit file for the distribution media

  A *name*.kit file (where *name* represents the device or product name) is provided on the hardware product kit and is located on the distribution media only. This file must ship in the ./usr/opt/*PROD_CODE*/sys/hardware directory. The *PROD_CODE* directory represents the three letter product code and product version. Using the example in this chapter, *PROD_CODE* is EDG100.

  The file is used to control the actions of the system's boot utility to allow the kernel to boot with the new hardware support software off the distribution media for the installation. The file is placed in the root directory on the hardware product kit distribution media after creating the kit media with the gendisk utility. The content of the distribution *name*.kit file specifies the location of the module files for the hardware on the distribution media. This file is not part of the installed product and does not need to be part of the product inventory. Each hardware product kit on the distribution media must have its own *name*.kit file.

  See Section 5.1.1 for more information about the format and contents of this file.

- A *name*.kit file to be installed on the target system

  A *name*.kit file (where *name* represents the device or product name) is installed with the kit software onto the target system. This file controls the boot link of the kernel from the installed target disk allowing the target system kernel to boot with the new hardware support software. The format of this file is the same as the format of the *name*.kit file on the distribution media, but the location of the module files is specified as the location of the hardware support files on the installed disk. This file is a required part of the installed product and needs to be part of the product inventory.

  See Section 5.1.1 for more information about the format and contents of this file.

- Additional installed *name*.kit files

  If there are any other hardware product kits on the same distribution media, you must create a separate *name*.kit file for each kit. A single media may contain several kits and several *name*.kit files (where *name* represents the device or product name).

- A *PROD_CODE*.root file

  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/*EasyDriver*/kit). This file is automatically created by the gendisk utility when a Direct CD-ROM (DCD) distribution media is built. Using the example in this chapter, this file would be called EDG100.root.

- A portion of the /etc/sysconfigtab file

  The contents are passed to the kernel and logically are appended to the contents of the /etc/sysconfigtab file read from the base disk. This database defines attributes of the modules configured by the boot utility. The contents of this file are described in Section 2.2.2.

- A *kitname*.kk file

  A *kitname*.kk file (where *kitname* is the product name) must be in the instctrl subdirectory so that the hardware product kit can be installed into a RIS area. The *kitname*.kk file is created automatically by the gendisk utility when the kk=true option is used in conjunction with the dd= option in the /etc/kitcap file.

  See Section 5.1.2 for more information about the *kitname*.kk file. See the kitcap(4) reference page for more information about the format of the /etc/kitcap file.

- The HW.db module database file

  The module database file describes the list of subsets that need to be installed for each kernel module. There is only one HW.db file on each piece of shipped media (CD-ROM or disk), and it contains information about all of the hardware product kits on the media. This file must be located in the root directory of the distribution media. See Section 5.1.3 for more information about the format and content this file.

- The hardware_kit.hw hardware support file

  The hardware support file contains information about each of the product kits available on the shipped media. There is only one hardware_kit.hw file on each piece of distribution media, which contains information about all the hardware products supported by the kits on the media. The hardware support file must be named hardware_kit.hw. This file must be located in the root directory of the distribution media. See Section 5.1.4 for more information about the format and content of this file.

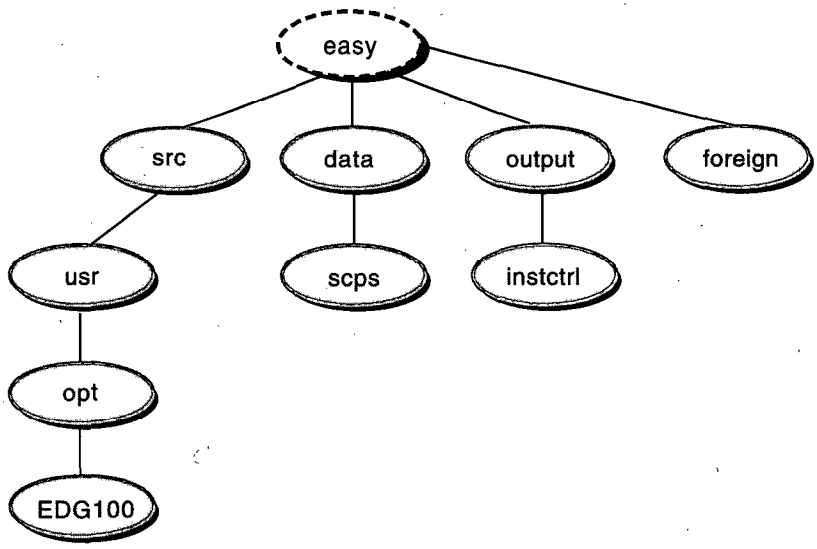
- An installed *hardwarename*.hw hardware support file

  The *hardwarename*.hw file is installed with the kit onto the target system and contains information about the hardware product supported

by the installed kit. There should be one *hardwarename*.hw file for each piece of hardware supported by the installed kit. Each file must be part of the inventory of the subset that contains the associated hardware support. This file must ship in the ./usr/opt/*PROD_CODE*/sys/hardware directory. The *PROD_CODE* directory represents the three letter product code and product version. Using the example in this chapter, *PROD_CODE* is EDG100. See Section 5.1.5 for more information about the format and contents of this file.

Figure 5–1 shows the directory structure for the EDGgraphics device driver product. In this figure, the top level directory (drawn with dashed lines), easy, is an existing directory under which the developer created the hierarchy directories. In the example, EDG is the three letter product code assigned to the sample graphics device driver produced by *EasyDriver, Inc.*.

Figure 5–1: Directory Structure for a Hardware Product Kit



The following sections describe the contents of the *name*.kit, *kitname*.kk, HW.db, hardware_kit.hw, and *hardwarename*.hw files.

5.1.1 The name.kit Files

The format of the distribution and the installed *name*.kit files is the same. The difference between the two files is:

- In the distribution *name*.kit file, the modules are specified as their location on the distribution media. This file must ship in the

./usr/opt/*PROD_CODE*/sys/hardware directory. This 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 as their location on the installed system. The installed *name*.kit file must ship under the /opt directory (that is, it must be on the root file system). This file is a required part of the installed product and must be part of the product inventory.

Commands in the distribution *name*.kit file describe how the boot utility needs to modify the bootstrap link process to boot this kit off the distribution media. When bootstrap linking the kernel, this file controls where hardware product modules are found during the boot process. When bootstrap linking from a hardware product kit, the boot utility sets the default directory to the media root directory. During a normal boot process, the default is /sys/BINARY on the system disk. Commands in the distribution *name*.kit file indicate which modules should be added, removed, or replaced in the kernel.

Note

The kernel will not allow two kernel modules to have the same name. To avoid kernel module naming conflicts with other OEMs and the base operating system, it is recommended that you prepend your three letter product code to your module names. For example, the module name for the sample hardware product in this chapter is EDGgraphics.mod.

Modules needed for the kernel must reside in the ./opt/*PROD_CODE*/sys/BINARY directory, so that they can be accessed at boot time. Kernel modules for the fictitious graphics device driver used in this chapter are located in ./opt/EDG100/sys/BINARY.

Note

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.

Table 5–1 shows the format of the commands in the *name*.kit file.

Table 5–1: Format of the name.kit File

Format of Command	Description
+[*device*:][*/path/*]*file*.mod	Adds *file*.mod from the root or the specified device. You can specify a full path or accept the default.
-[*/path/*]*file*.mod	Deletes (or subtracts) *file*.mod from the module list on the default path for *file*.mod. Each module in the kit file must be removed before it is added in case the module already exists.
file.mod=[*device*:][*/path/*]*new*.mod	Replaces *file*.mod on the default path with the module you specify.

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 EDGgraphics device driver supplies a *name*.kit file which first removes and then adds the EDGgraphics.mod single binary module to the kernel. In Example 5–1, the sample *name*.kit file is referencing the /opt/EDG100/sys/BINARY directory on the installed system, not on the distribution media.

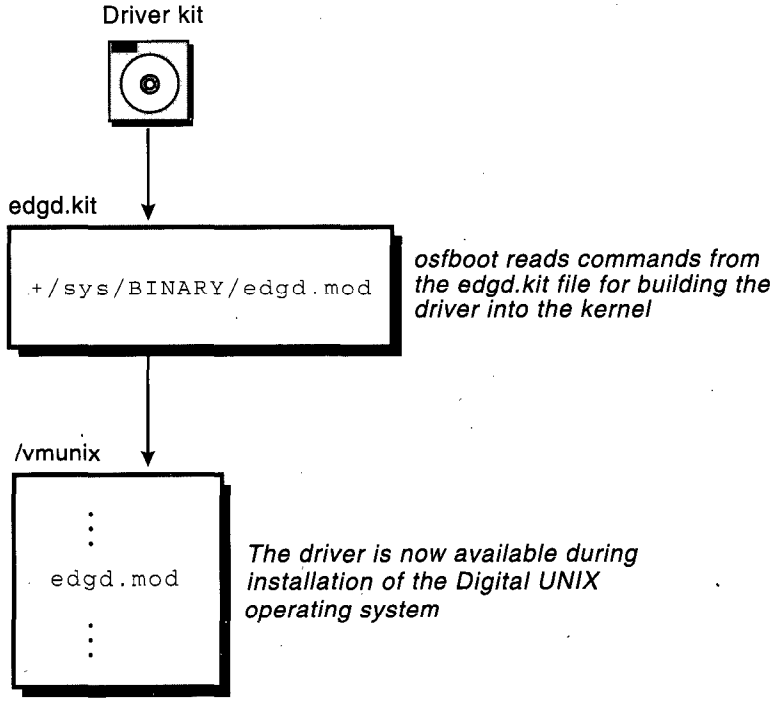
Example 5–1: Contents of an Installed name.kit File

```
-/opt/EDG100/sys/BINARY/EDGgraphics.mod
+/opt/EDG100/sys/BINARY/EDGgraphics.mod
```

Figure 5–2 shows how the distribution *name*.kit file works with boot utility software during the installation of a hardware product kit.

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

Figure 5–2: Using the Distribution name.kit File During Installation



5.1.2 The kitname.kk File

By default, all hardware product kits should contain a *kitname*.kk file 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 automatically created by the gendisk utility when it finds the kk=true option in the /etc/kitcap file.

5.1.3 The HW.db File

The HW.db file is the module database file. This file only is needed when you are creating a hardware product kit. The file describes how to map a kernel module to a list of subsets that need to be installed. The installation process uses this description to determine what subsets need to be loaded from the hardware product kit and if the kernel needs to be rebuilt after the subsets have been loaded.

The module database file must be named HW.db, and it must be located in the root directory of the hardware product media. The file must contain one entry (and only one) per kernel module. For example, if the hardware product kit has seven kernel modules, each module will have one entry in the HW.db file.

There can be only one HW.db file on each piece of distribution media you are shipping. If you are planning to ship several hardware product kits on the same CD-ROM, the module database file will contain one entry for each kernel module for each kit.

For example, if you shipped two hardware product kits, with the first kit having three kernel modules and the second kit having five kernel modules, the HW.db file would have eight entries.

The syntax for the HW.db file uses the following conventions:

- Lines beginning with the # character are comment lines. All text after the # (number or pound sign) character is ignored. Blank lines are also ignored.
- Each entry must be on a single line. Line continuation, carriage return, and line feed characters are not permitted.

Example 5–2 shows a sample HW.db file. The format for a file entry is:

- Module name
- Kernel build requirements enclosed in curly braces. Values are { 0 } or { 1 } – 0=no build; 1=build required
- List of subsets enclosed by { } (curly braces) and separated by spaces

Example 5–2: Contents of a HW.db File

```
/EasyDriver/opt/EDG100/sys/BINARY/EDGgraphics.mod
{1}
{EDGBASE100 EDGDOC100}
```

5.1.4 The hardware_kit.hw File

The hardware_kit.hw file is the hardware support file. This file only is needed when you are creating a hardware product kit. The file contains information about each of the product kits available on the distribution media and it is used by the Update Installation process.

The hardware support file must be named hardware_kit.hw, and it must be located in the root directory of the distribution media. The file must

contain one entry (and only one) per piece of hardware supported by the kit. If the hardware product kit contains support for several pieces of hardware, each piece of hardware will have one entry in the hardware support file. Even if the media only contains one subset, every piece of hardware supported on the media must have one entry in the file.

There only can be one `hardware_kit.hw` file per piece of media you are shipping. If you are planning to ship several hardware product kits on the same CD-ROM, the hardware support file will contain one entry for each piece of hardware supported by each kit on the media.

For example, if you shipped two hardware product kits, with the first kit containing support for two pieces of hardware and the second kit containing support for seven pieces of hardware, the `hardware_kit.hw` file would have nine entries.

The syntax for the `hardware_kit.hw` file uses the following conventions:

- Lines beginning with the # (pound sign or number) character are comment lines. All text after the # character is 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.

The format for an entry contains four fields:

- Vendor name
- Hardware name
- Name of kit file containing the necessary kernel modules for the hardware
- List of releases under which the product is supported (space separated). Do not include the letter `V` in front of the release number.

The vendor and hardware name must be unique when combined and must match the name to be used in the *`hardwarename`*`.hw` file, described in Section 5.1.5. Each product release supported by the kit must map to at least one release of the operating system as shown in Example 5–3. Operating system release versions use the format {*`MajorNumber`*}.{*`MinorNumber`*}[*`VariantLetter`*].

Example 5–3: Contents of a Hardware Support File

```
{
    {Easy Driver Inc.}
```

Example 5–3: Contents of a Hardware Support File (cont.)

```
    {EDG Graphics Device Driver}
    {/EasyDriver/usr/opt/EDG100/sys/hardware/edgd.kit}
    {4.0F}
}
```

5.1.5 The hardwarename.hw File

The *hardwarename*.hw file is the installed hardware support file. The file contains information about each of the product kits and it is installed onto the user's system. The file must be part of the subset inventory file and is used by the Update Installation process to determine what hardware product has been installed. Each piece of supported hardware installed on the user's system needs to have a unique *hardwarename*.hw file containing the subsets that were installed to support the hardware.

The installed hardware support file must use a unique combination of vendor and hardware identification for the file name and must use the .hw extension. For example, for the EasyDriver graphics device driver, the file might be called edgdriver.hw.

The file must be located in the ./usr/opt/*PROD_CODE*/sys/hardware directory relative to the top of the hardware product kit for each piece of supported hardware. For example, using our sample hardware scenario, the edgdriver.hw file is in the ./usr/opt/*EDG100*/sys/hardware/ directory.

Each piece of hardware in each of the hardware product kits on the media should have its own *hardwarename*.hw file. This enables the subsets to be built so that only files that contain entries for the hardware product support will be installed.

As an example, assume that the EDG graphics device driver product has two subsets, EDGBASE100 and EDGLSM100. The EDGBASE100 subset supports two pieces of hardware, ATM_V1 and ATM_V2, and the EDGLSM100 subset supports a hardware product called LSM_V1. If a user installs the EDGBASE100 subset, the installed hardware support file should contain entries for both ATM_V1 and ATM_V2, but should not contain an entry for LSM_V1. The installed hardware support file for the EDGLSM100 subset only would have an entry for the LSM_V1 product.

You could also have ATM_V1 and ATM_V2 as single entries in separate installed *hardwarename*.hw files. Each file would be listed as inventory for the EDGBASE100 subset and would be installed along with the subset. The

installed *hardwarename*.hw file can contain more than one entry, but having an installed hardware support file for each entry will provide maximum flexibility when building the subsets.

The syntax for the *hardwarename*.hw file uses the following conventions:

- Lines beginning with the # (pound or number sign) character are comment lines. All text after the # character is ignored. Blank lines are 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.

The format for an entry contains four fields:

- Vendor name
- Hardware name
- Name of kit file containing the necessary kernel modules for the hardware
- Releases under which the product is supported (space separated) are in the format {*MajorNumber*}.{*MinorNumber*}[*VariantLetter*]

As described in Section 5.1.4, the vendor and hardware name must be unique when combined and must match the name used in the hardware_kit.hw file . Each product release supported by the kit must map to at least one release of the operating system, as shown in Example 5–4.

Example 5–4: Contents of an Installed Hardware Support File

```
{
    {Easy Driver Inc.}
    {EDG Graphics Device Driver}
    {EDGBASE100 EDGDOC100}
    {4.0F}
}
```

5.2 Creating a Subset Control Program for a Hardware Product

In addition to the optional processing described in Section 3.4, a subset control program for a hardware product kit such as a device driver also must configure the driver into the kernel. When building subset control programs for a hardware product kit, such as a device driver, 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 5–5 shows the subset control program for the single binary module associated with the `EDG graphics` device driver. The user can 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 5–5: Subset Control Program for the EDGgraphics Device Driver

```
#!/sbin/sh
#
#
#  EDGBASE100.scp - Install the files associated with the EDGgraphics
#  device driver.  This driver, implemented as a single binary
#  module (EDGgraphics.mod file), can be statically or dynamically
#  configured into the kernel.
#

case "$ACT" in 1
C)
    case $1 in
    INSTALL)  2
        echo "*****  EDG Graphics Product Installation Menu *****"
        echo "*****                                        *****"
        echo "1. Install the static device driver subset."
        echo "2. Install the dynamic device driver subset."

        echo" Type the number for your choice [] "

        read answer
        case ${answer} in
            1)  3
            # Register the files associated with the static
            # EDG graphics device driver product.
            kreg -l EasyDriverInc EDGBASE100 /opt/EDG100/sys/BINARY4
```

Example 5–5: Subset Control Program for the EDGgraphics Device Driver (cont.)

```
        # Merge the files associated with the statically configured
        # EDG graphics device driver product to the customer's
        # /etc/sysconfigtab database
        sysconfigdb -m -f /opt/EDG100/etc/sysconfigtab EDGgraphics [5]

        echo "The rest of the procedure will take 5-15 minutes"
        echo "to rebuild your kernel, depending on the processor"
        echo "type."
        echo ""
        echo "Starting kernel rebuild... "
        if doconfig -c $HOSTNAME [6]
            then
            echo "Kernel built successfully"
        else
            1>&2 echo "Error building kernel."
            return 1
        fi
        ;;

        2) [7]
        # Merge the files associated with the dynamically configured
        # EDG graphics device driver product to the customer's
        # /etc/sysconfigtab database
        sysconfigdb -m -f /opt/EDG100/etc/sysconfigtab EDGgraphics [8]

        # Load the EDG graphics device driver and create the device
        # special files
        sysconfig -c EDGgraphics [9]

        echo "The EDG graphics device driver was added to your
        echo "/etc/sysconfigtab database." [10]
        ;;
    esac
    ;;

DELETE) [11]
    echo "***** EDG Graphics Product Removal Menu *****"
    echo "*****                                  *****"
    echo "1. Delete the static EDG graphics device driver subset."
    echo "2. Delete the dynamic EDG graphics device driver subset."

    echo" Type the number for your choice [] "

    read answer
    case ${answer} in
        1)
        kreg -d EDGBASE100 [12]

        # Delete the EDG graphics device driver's entry from the
        # /etc/sysconfigtab database

        sysconfigdb -d EDGgraphics [13]

        echo "The rest of the procedure will take 5-15 minutes"
        echo "to rebuild your kernel, depending on the processor"
        echo "type."
        echo ""
        echo "Starting kernel rebuild... "
```

Example 5–5: Subset Control Program for the EDGgraphics Device Driver (cont.)

```
            if doconfig -c $HOSTNAME [14]
                then
                echo "Kernel built successfully"
            else
                1>&2 echo "Error building kernel."
                return 1
            fi
            ;;

            2)
            # Make sure the EDG graphics device driver is not currently
            # loaded
            sysconfig -u EDGgraphics [15]

            # Delete the EDG graphics device driver's entry from the
            # /etc/sysconfigtab database
            sysconfigdb -d EDGgraphics [16]
             ;;
        esac
        ;;
    esac
    ;;
esac
exit 0
```

[1] Examines the ACT environment variable to select the action the subset control program should take.

[2] Displays a menu of installation options during the C INSTALL phase. The user can install the driver for static configuration or dynamic configuration.

[3] The system performs a static configuration if the user chooses menu item 1.

[4] Invokes the kreg utility to register the driver files with the kernel. The kreg utility registers a device driver product by creating the /usr/sys/conf/.product.list file on the customer's system. This file contains registration information associated with the static device driver product. The subset control program calls kreg with the following arguments:

- The -l flag

  This flag indicates that the subset was loaded, and it directs kreg to register the device driver product as a new kernel extension.

- Company name

  The company name is EasyDriverInc. The kreg utility places this name in the company name field of the customer's /usr/sys/conf/.product.list file.

- Software subset name

  The software subset name for this device driver product is EDGBASE100. The subset name consists of the product code, subset mnemonic, and three digit version code. The kreg utility extracts information from the specified subset data and loads it into the customer's /usr/sys/conf/.product.list file.

- Directory name

  The directory on the customer's system where kreg copies the files associated with this driver product is /opt/EDG100/sys/BINARY. The kreg utility places this directory in the driver files path field of the customer's /usr/sys/conf/.product.list file.

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

5 Adds the /opt/EDG100/etc/sysconfigtab file fragment for the statically configured driver to the system's /etc/sysconfigtab database by calling the sysconfigdb utility with the following arguments:

- The -m flag

  This flag causes sysconfigdb to merge the device driver entry to the customer's /etc/sysconfigtab database.

- The -f flag

  This flag precedes the name of the sysconfigtab file fragment whose device driver entry is to be added to the /etc/sysconfigtab database. This flag is used with the -a flag.

- The sysconfigtab file fragment

  The kit developer at EasyDriver, Inc. specifies the path /opt/EDG100/etc/sysconfigtab to indicate the location of the sysconfigtab file fragment for the EDGgraphics device driver.

- Device driver name

  The kit developer at EasyDriver, Inc. specifies EDGgraphics as the name of the device driver whose associated information is added to the /etc/sysconfigtab database. This name is obtained from the entry_name item of the sysconfigtab file fragment, as described in *Writing Device Drivers: Tutorial*.

6 Runs the doconfig utility to configure the driver into the kernel. The subset control program returns an error if doconfig fails for any reason.

7 Performs a dynamic configuration if the user chooses menu item 2.

8 Calls the sysconfigdb utility to add the driver's sysconfigtab file fragment to the system's /etc/sysconfigtab database.

[9] Calls the `sysconfig` utility with the `-c` option to reconfigure the system and include the `EDGgraphics` device driver. The `-c` option causes the `sysconfig` utility to dynamically configure the driver into the system that is running and to create device special files. The name of the driver as specified in the `sysconfigtab` file fragment follows the option.

[10] Displays a message notifying the user that the driver has been added to the system.

[11] Displays a menu of options for deleting subsets during the `C DELETE` phase. The user must tell the `setld` utility whether the subset to be deleted represents a statically configured driver or a dynamically configured driver. The way the driver was configured determines how the driver is deleted.

[12] Calls the `kreg` utility to deregister the driver with the kernel if the user chooses menu option 1 (delete a statically configured driver). When the `kreg` utility is called with the `-d` flag, it deletes the entry for the specified layered product from the customer's `/usr/sys/conf/.product.list` file. In this case, the layered product is the `EDGgraphics` device driver, represented by the `EDGBASE100` subset identifier.

[13] Calls the `sysconfigdb` utility with the `-d` flag, which deletes the static `EDGgraphics` kernel subsystem from the customer's `/etc/sysconfigtab` database.

[14] Runs the `doconfig` utility to reconfigure the kernel. The subset control program returns an error if `doconfig` fails for any reason.

[15] Calls the `sysconfig` utility with the `-u` flag to deconfigure the dynamically configured `EDGgraphics` kernel subsystem from the system that is running if the user chooses menu item 2 (delete a dynamically configured driver).

[16] Calls the `sysconfigdb` utility with the `-d` flag to delete the dynamically configured `EDGgraphics` kernel subsystem from the customer's `/etc/sysconfigtab` database.

5.3 Creating Distribution Media for a Hardware Product Kit

To prepare a hardware product kit, edit the `/etc/kitcap` file to describe the kit, and then run the `gendisk` utility with the `-d` option. The `gendisk` utility creates a kit in Direct CD-ROM (DCD) format as specified in the `/etc/kitcap` entry. The `/etc/kitcap` file is a database for kit descriptors. This database contains product codes, media codes, and the names of the directories, files, and subsets that make up product description.

Example 5–6 shows a sample `kitcap` record for a hardware product kit. Notice the use of the `kk=true` and `rootdd=` options.

Example 5–6: Sample /etc/kitcap Record for a Hardware Product Kit on CD-ROM

```
EDG100HD:c:/: \
  dd=/EasyDriver/kit,kk=true,rootdd=..:EasyDriver_edg_driver: \
 /easy/output:instctrl:EDGBASE100
```

Example 5–7 shows a sample `kitcap` record for a CD-ROM with multiple hardware products. In Example 5–7, the `,rootdd=..` entry overrides the default and places the expanded subset files in the product-specific directory of `/EasyDriver`. By default, expanded subset files (in DCD format) are placed at the top of the media's file system.

Example 5–7: Sample /etc/kitcap Record for a CD-ROM with Multiple Hardware Kits

```
EDG100HD:c:/: \
  dd=/EDG100/kit,kk=true,rootdd=..:EasyDriver_edg_driver_V1.0: \
 /easy100/output:instctrl:EDGBASE100: \
  dd=/EDG200/kit,kk=true,rootdd=..:EasyDriver_edg_driver_V2.0: \
 /easy200/output:instctrl:EDGBASE200:
```

Refer to the `kitcap`(4) reference page for more information about the format of the `/etc/kitcap` file.

Note

Each kit must be contained in a subdirectory of the root media which only can be one level deep. For example, a kit is located in `/mnt/EasyDriver` where `/mnt` is the mount point and `EasyDriver` is the subdirectory under which the kit files are located.

The following instructions describe how to assemble the files to create the distribution media. The hard disk serves as the master for the kit. You can then burn the kit onto a CD–ROM by following the instructions that came with your CD–ROM burner.

Note

When testing a DCD kit, be sure to reference the kit media at its mount point. For instance, if you decide to use a spare disk for creating a media master area, you must reference your kit to the mount point of the device.

Perform the following steps to create a kit on a hard disk for the EDG100 product:

1. Determine the device where the master media is located. The following example uses the disk located at rz1.
2. Erase any existing label on the disk (this destroys the data on the disk):

```
# disklabel -z rz1
```

3. Write a disk label to the disk:

```
# disklabel -wr rz1 rz26L
```

Note

Before running the gendisk utility in the next step, make sure you have added the name of the system you are using to the root (/) .rhosts file. You will need an entry for the host name and root account, otherwise the gendisk utility fails with a Permission denied error.

4. Run the gendisk utility to move the kit onto the disk. In this example, the system name is visier.

```
# gendisk -d EDG100 /dev/rz1a
Generating EDG100 Kit from visier on /dev/rz1a

WARNING:  this will remove any information stored in
/dev/rz1a
Are you sure you want to do this? (y/n): y
Do you want to clean the entire disk first? (y/n): n
```

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

The output of the gendisk utility is similar to the following:

```
Preparing /dev/rz1a
done.

Checking /dev/rz1a
```

```
/sbin/usf_fdck /dev/rz1a
** /dev/rz1a
File system unmounted cleanly - no fsck needed

Mounting /dev/rz1a on /usr/tmp/cd_mdt8344

Writing Images (dd=/).

 Image instctrl...done.
 Image EDGBASE100...done.

Verifying Images (dd=/).

 Image instctrl...done.
 Image EDGBASE100...done.

Kit EDG100 done.

Cleaning up working directories.
Unmounting /dev/rz1a
```

5. Mount the disk in preparation for making hardware product kit modifications:

```
# mount /dev/rz1a /mnt
# cd /mnt
```

6. Verify that the *kitname*.kk and *PROD_CODE*.root files exist in the instctrl directory:

```
# cd ./EasyDriver/kit/instctrl
# ls *.kk *.root
```

7. Copy the *name*.kit, HW.db, and *hardware_kit*.hw files from the kit-building area to the disk:

Note

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

```
# cp /kit_area/edgd.kit \
  /mnt/EasyDriver/usr/opt/EDG100/sys/hardware/edgd.kit
# cp /kit_area/HW.db /mnt/HW.db
# cp /kit_area/hardware_kit.hw /mnt/hardware_kit.hw
```

8. Unmount the disk:

```
# umount /mnt
```

5.4 Testing a Hardware Product Kit

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

There are four separate tests that you should run to test the completeness of a hardware product kit:

- 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.
- 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 boot linked kernel.
- Use the `hw_check` utility to verify that the `HW.db`, *`hardware_kit`*`.hw`, and *`hardwarename`*`.hw` files are in the correct locations and that they contain the correct information.
- 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. Then, test the area by registering a client system to it and starting a Full Installation of the client system.

The following sections describe how to set up and perform these four test cases.

5.4.1 Using setld to Test a Hardware Product Kit

To test a hardware product kit using the `setld` utility, the system on which the test is run must be running the same version of the operating system for which the hardware product kit was built. That is, if the hardware product kit was built for Version 4.0F of the operating system, the target system must be running Version 4.0F.

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 as the user `root` from multiuser mode on the system to be tested.
2. Mount the hardware product kit using the `mount` command:

   ```
   # mount -r /dev/rz3a /mnt
   ```

3. Use the following `setld` command syntax to install each subset on the distribution media:

```
# setld -l location subset_name
```

In the command syntax *location* is the path to the ./kit directory on the hardware product kit (for example, /mnt/EasyDriver/kit), and *subset_name* is the name of the subset (for example, EDGBASE100). If you want to install more than one subset, separate each subset name with a space. Using these examples, the command line looks like this:

```
# setld -l /mnt/EasyDriver/kit EDGBASE100
```

Successful output of the setld utility is similar to the following:

```
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)...

EDG Graphics Device Driver Files
  Copying from /mnt/EasyDriver/kit (disk)
  Verifying

1 of 1 subset(s) installed successfully.

Configuring "EDG Graphics Device Driver Files " (EDGBASE100)
```

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.

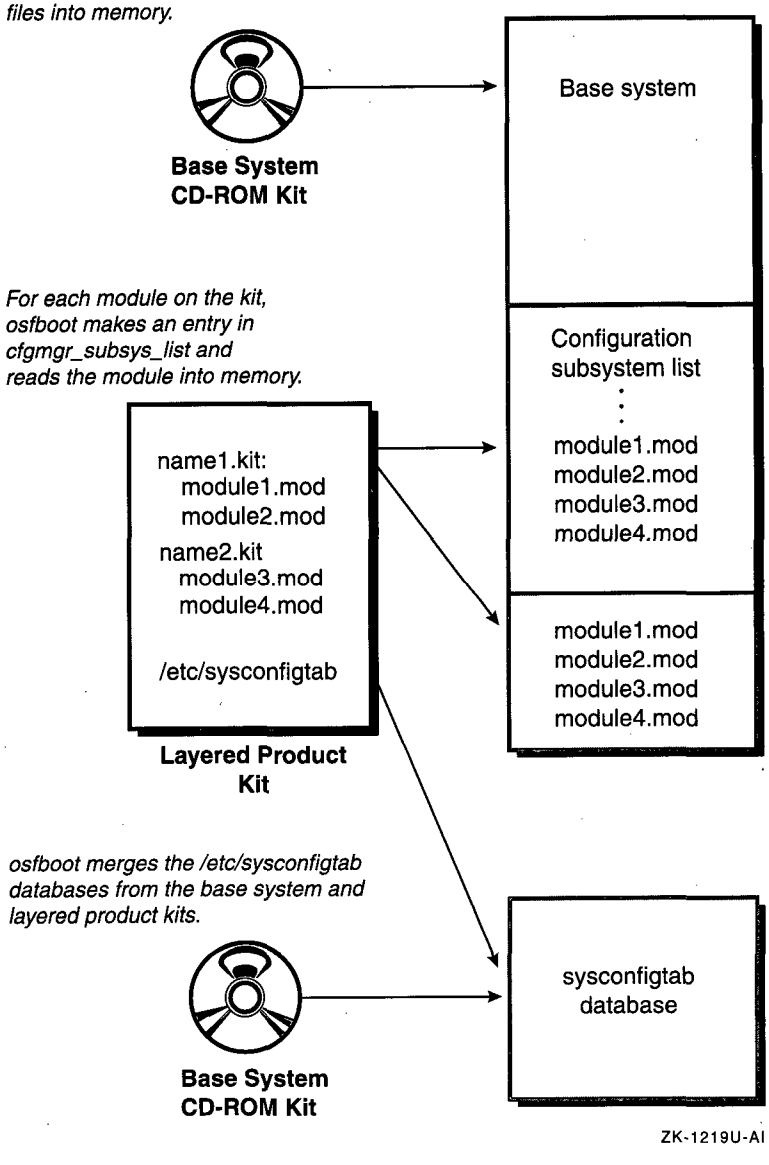
4. After the setld utility has finished installing and configuring each subset, verify that the files for each subset (including any *hardwarename*.hw files) have been installed in the locations you intended them to be in.

5.4.2 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 Version 4.0F of the operating system.

Figure 5–3 shows how the boot utility builds a kernel to include a hardware product kit.

Figure 5–3: Bootstrap Linking 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 Version 4.0F of the operating system:

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, install the new hardware, and power up your system.

6. Enter the following command at the console prompt:

```
>>> boot -fl fa -fi "/GENERIC" sys_disk
```

In the previous example:

- The `-fl fa` defines the boot flags: `f` for a hardware product kit and `a` for multiuser mode.
- The `-fi "/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.

7. Enter the CD-ROM console device name, for example, `DKA500`, at the following prompt:

```
Enter Device Name:
```

8. Enter the hardware product kit name at the following prompt:

```
Enter Kit Name:
```

Enter the name of the hardware product kit that you want to install and press Return. This is the full path and file name of the `name.kit` file.

9. Insert the CD-ROM containing hardware product kit into the CD-ROM drive when you see the following prompt:

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

In this example, *device:hw_kit_name* is the device name you entered in Step 7 and the hardware product kit name you entered in Step 8 .

When you press the Return key, the `boot` utility reads the selected hardware product kit file into memory.

10. Do one of the following at the `Enter Kit Name:` prompt:

- If you are installing another hardware product kit from the same CD-ROM, enter the kit name, press Return, and return to Step 9.

- If you are not installing another hardware product kit just press Return and continue.

11. Do one of the following at the `Enter Device Name:` prompt:
    - If you are installing another hardware product kit, enter the CD-ROM device name, press Return, and go back to Step 8.
    - If you are not installing another hardware product kit, just press Return and continue.
12. Because you are adding hardware support to a running system, and the system disk is your boot media, just press Return at the following prompt:

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

    The generic kernel modules are read so that the bootlink process can build the kernel in memory in the next step.
13. The `boot` utility links the kernel objects, and issues the following prompt. Insert the CD-ROM into the drive and press Return.

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

    In this prompt *dev_name* is the CD-ROM device name you entered in Step 7 and *hw_kit_name* is the hardware product kit name that you entered in Step 8.

    This step is performed for every device and every kit name entered in Step 7 through Step 11.
14. Because the boot media is still your installed system disk, press Return at the following prompt:

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

    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.

15. Enter the name of the kernel configuration file at the following prompt:

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

In the previous prompt, *SYS_NAME* is the name of your existing kernel configuration file, usually your system name in upper case characters. Two things may happen here:

- 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 *SYS_NAME*.bck.
- If the boot utility prompts you to rebuild the kernel:

  a. 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.

  b. After selecting kernel options, 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]:
```

Respond as follows:

- 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 14a and select kernel options again.

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

16. 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.
17. Log in as the user `root` and use the `setld -i` command to verify that your hardware product kit is installed
18. Check to make sure that the installed files are where you want them to be and that the hardware product is operational.
19. 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 module_name.mod
```

In the previous example, *module_name*`.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.

5.4.3 Using the hw_check Utility to Test a Hardware Product Kit

When you perform an Update Installation, the process automatically checks for any installed hardware product kits and lets you know:

- 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 in which to test the validity of a hardware product kit during an Update Installation.

Follow these steps to test the format of the `*.hw` files:

Note

The following test assumes that you have performed the bootlink test as described in Section 5.4.2.

1. Log in as the user `root` to the same system where you performed the boot link test.
2. Mount the Version 4.0F operating system CD-ROM from multiuser mode:

   ```
   # mount -r /dev/rz9a /mnt
   ```

3. Run the `hw_check` utility:

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

   A successful test shows output similar to the following:

   ```
   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.

           EDG Graphics Device Driver

   Press RETURN to continue...
   ```

Note

If you do not see this message, the most likely cause of the error is an incorrect value in the supported release field of the *hardwarename*`.hw` file (for example, `./usr/opt/EDG100/sys/hardware/edgd.hw`). See Section 5.1.5 for information about the format of this file.

Next, follow these steps to run the `hw_check` utility which tests the kit just 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 have already tested the kit with the bootlink utility as described in Section 5.4.2.

1. Log in as the user 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/opt/*PROD_CODE*/sys/hardware/*hardwarename*.hw file and change the value in the supported release field to XXX. For example, assume this is the original installed *hardwarename*.hw file:

```
{
    {Easy Driver Inc.}
    {EDG Graphics Device Driver}
    {EDGBASE100 EDGDOC100}
    {4.0F}
}
```

Change the supported release field as shown in this example:

```
{
    {Easy Driver Inc.}
    {EDG Graphics Device Driver}
    {EDGBASE100 EDGDOC100}
    {XXX}
}
```

Changing the support 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
```

The output of the hw_check utility is 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 (V4.0F).  In order for the update to
complete successfully, you must provide the distribution
media that contains the V4.0F version of the hardware
product listed below.  The update install process will
verify that the media you provide contains
the correct software.

      Easy Driver Inc. EDG Graphics Device Driver

Enter kit location (e.g.: /dev/rz3c 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 V4.0F.  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.

  Easy Driver Inc. EDG Graphics Device Driver
 (Kit File: /EasyDriver/usr/opt/EDG100/sys/hardware/edgd.kit)

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

5. The test is successful when you see this message. Otherwise, the `hw_check` utility displays an error message to describe the problem. You must correct the problem and start the test over.

5.4.4 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, and start the RIS utility:

```
# /usr/sbin/ris
```

2. Choose `INSTALL software products` from 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: i
```

3. **Choose `Install software into a new area` from 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: 1
You have chosen to establish new remote installation
environment.
```

4. **Enter the location of the base operating system distribution media. In this example, the location is `/mnt/ALPHA/BASE`.**

```
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: /mnt/ALPHA/BASE
```

5. **Select the Boot-Link method to create the base product:**

```
Select the type of DIGITAL 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:  2
```

6. **The base operating system is extracted into the new RIS area. The `ris` utility displays a list of subset extraction messages and the base product that has just been extracted to the RIS area.**

```
        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    'DIGITAL UNIX V4.0F Operating System ( Rev nnn )'
```

7. **Choose `INSTALL software products` from the RIS Utility Main Menu to extract the hardware product kit into the RIS area you have just created:**

```
*** 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: 1
```

8. **Choose** `Add software into an existing area` **from 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: 2
```

The `ris` **utility displays the names of any existing RIS environments.**

9. **Enter the number that represents the new RIS environment where you just extracted the base operating system.**

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

Select the remote installation environment:

    1) /usr/var/adm/ris/ris0.alpha
         'DIGITAL UNIX V4.0F Operating System ( Rev nnn )

Enter your choice or press RETURN to quit: 1
```

10. **Enter the location of the hardware product kit that you want to install. In this example, the location is** `/mnt/kit`**. The location you enter is the name of the top-level directory of the kit followed by the** `kit` **directory.**

```
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: /mnt/kit
```

The `ris` **utility searches the distribution media for the** *kitname*`.kk` **file, which indicates that the distribution media contains a hardware product kit. If this file is found, you will see the message displayed in the next step.**

11. **Select the option to** `Integrate with Base product and include product` **to integrate the kit with the base product.**

```
The kit you have specified has been identified as a DIGITAL
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:1
```

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, then, start this test over.

12. Select the version of the base operating system into which you want to integrate the kit:

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

    1   'DIGITAL UNIX V4.0F Operating System ( Rev nnn )'

 Enter your selection or <return> to quit: 1
```

13. Decide whether you want to either extract the software onto the system or just create symbolic links to the software. In this example, the option to `Extract software` is selected.

```
Choose one of the following options:

    1)  Extract software from   /mnt/EDG100/kit
    2)  Create symbolic link to /mnt/EDG100/kit

 Enter your choice: 1
```

The `ris` utility lists all the subsets associated with the software kit.

14. Select the subsets you want to install. In this example, the `EDG100` kit contains only one subset.

```
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) EDG Graphics 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): 1
```

15. Confirm your software selection:

```
You are installing the following optional subsets:

        EDG Graphics Device Driver Version 1

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

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

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

File system space checked OK.
Extracting EDGBASE100...
Media extraction complete.


The following software now exists in the RIS product area
/var/adm/ris/ris0.alpha:
1    'DIGITAL UNIX V4.0F Operating System ( Rev nnn )' with
     'EDG Graphics Device Driver Version 1'

2    'EDG Graphics Device Driver Version 1'
```

The hardware product has been extracted into the RIS area and installed 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 will see that there are now two products in the new RIS area: the current version of the operating system with support for the `EDGgraphics` device driver and the `EDGBASE100` hardware product kit.

5.4.4.1 Registering a Client for a RIS Area Containing 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 an installation from this RIS area, you must register the client, as described in the following steps:

1. Log in to the RIS server.
2. Start the ris utility:

```
# /usr/sbin/ris
```

3. Choose `Add a client` from 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: a
```

Through a series of prompts, the `ris` utility lets you know what information you need to enter and gives you the opportunity to exit from the procedure.

4. Enter **y** to continue if you have the information you need:

```
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]: y
```

5. Enter the client's host name. In this example, the host name is `aruba`:

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

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

6. Enter the products you want the client system to be able to install. In this example, the base system that includes kernel support for the `EDGgraphics` device driver and the `EDG` product itself are selected:

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

Product    Description
    1    'DIGITAL UNIX V4.0F Operating System ( Rev nnn )' w/
         'EDG Graphics Device Driver Version 1'
    2    'EDG graphics Device Driver Version 1'

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

7. Confirm your selection at the following prompt:

```
You chose the following products:

   1    'DIGITAL UNIX V4.0F Operating System ( Rev nnn )' w/
        'EDG Graphics Device Driver Version 1'
   2    'EDG Graphics Device Driver Version 1'

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

8. Enter the network type and the client processor's hardware network address:

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

Enter your choice: 1

Enter the client processor's hardware network address.
For example, 08-00-2b-02-67-e1: 01-47-2b-e2-3a-43
```

9. Client registration is complete. You may exit from the `ris` utility.

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

```
>>> boot -fl fa boot_device
```

In the previous example:

- The `-fl 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 is usually `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*.

6

Producing Distribution Media for User and Kernel Product Kits

After you have gathered product files into subsets, you can move the subsets onto the distribution media.

Note

Procedures for creating distribution media for hardware product kits is documented in Section 5.3.

You can create the kit in either `tar` or direct CD–ROM (DCD) format. If your product kit does not access kernel modules during boot, you can use the `tar` format to compress your kit and save space on the media. If your product kit does access kernel modules during boot, you must use the DCD format.

- User and kernel product kits can be distributed in `tar` format

  In `tar` format, the product files belonging to the same subset are dumped to the distribution media as a single file. During installation, the `setld` utility uncompresses the files, then moves them onto the target system, preserving the files' original directory structure.

- Hardware product kits must be distributed in direct CD–ROM (DCD) format

  In DCD format, the files are written to the distribution media as a UNIX file system. Subsets distributed in DCD format cannot be compressed.

You can distribute user and kernel product kits on tape, diskette, or CD–ROM, as follows:

- Magnetic tape

  You can distribute kits for user and kernel products on magnetic tape. You cannot distribute hardware product kits on magnetic tape because this media does not support DCD format. Use the `gentapes` utility to produce kits for magnetic tape media.

- Diskette

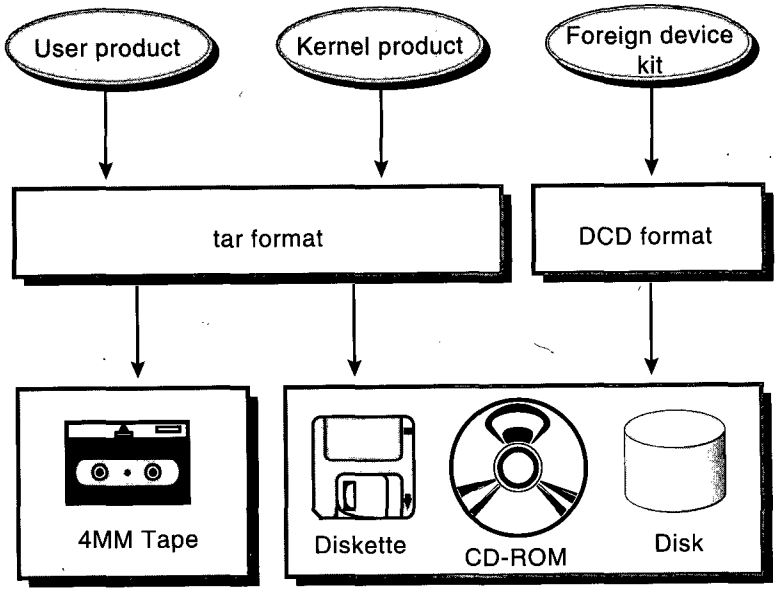
Diskettes are a good media for testing purposes or for small products, such as device drivers. However, the product must fit on a single diskette; it cannot span multiple diskettes. Use the `gendisk` utility to produce kits for diskette media.

- 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 burned onto the CD–ROM. Use the `gendisk` utility to produce the master kit on hard disk. Follow the CD–ROM manufacturer's instructions for burning the kit onto the CD–ROM media.

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

Figure 6–1: File Formats for Layered Product Kits



ZK-1215U-AI

The `gentapes` and `gendisk` utilities refer to a file called `/etc/kitcap`, a database of kit descriptors. This database contains 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.

This chapter describes how to edit the `/etc/kitcap` file and how to use the `gentapes` and `gendisk` utilities to produce kits for each type of media.

6.1 Editing the /etc/kitcap File

Before you can build your kit, you must add a record to the `/etc/kitcap` database to describe your kit. When you add a record to the file, use the following conventions:

- Separate fields with colons (:).
- Indicate a continuation line with a backslash (\) at the end of the line.
- Begin a comment line with a number sign (#). The comment ends at the end of the line.
- Delimit comments within a `kitcap` record with an opening number sign (#) and a closing colon (:).

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

The contents of the record also depends on the product type you are delivering. For example, the `kitcap` record for a kernel product must contain the `kk=true` flag and might require the use of the `rootdd=` option. It is recommended that you refer to the `kitcap`(4) reference page for more information about the contents of the `/etc/kitcap` file.

6.1.1 Tape Media kitcap Record Format

The `kitcap` record for tape media contains the following elements:

- Name of the product, which consists of the product code and version number specified in the `CODE` and `VERS` fields of the key file (the key file is the file with the `.k` suffix).
- A code that indicates the media type, either `TK` for TK50 tapes or `MT` for 9-track magnetic tapes.
- Product description. This entry usually is taken from the `NAME` field of the key file.
- Name of the kit's output directory, where the `gentapes` utility can find the subsets.
- Three `SPACE` files, which are empty files used 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:

```
# touch space
# tar -cf SPACE space
```

- The `instctrl` directory, relative to the output directory specification.

- The names of the subsets that make up the kit. Each subset listed must be stored in one of the specified directories.
- An optional volume identifier. Multiple tapes are supported.

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

Example 6–1 shows the record to be added to the /etc/kitcap file to produce the ODB kit on TK50 tapes:

Example 6–1: Sample /etc/kitcap Record for Magnetic Tape

```
OAT100TK | Orpheus Document Builder: \
   /dcb_tools/output:SPACE:SPACE:SPACE: \
   INSTCTRL:OATODB100:OATODBDOC100
```

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 /dcb_tools/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 two subsets that make up the kit — OATODB100 and OATODBDOC100.

6.1.2 Disk Media kitcap Record Format

You create a disk media kitcap record when producing kits for distribution on diskette or CD-ROM. The kitcap record for disk media contains the following elements:

- Name of the product, which consists of the product code and version number specified in the CODE and VERS fields of the key file.
- The code HD, which indicates disk media.
- 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.
- Product description, which must use underscores (_) in place of spaces. This entry is usually taken from the NAME field of the key file.
- The destination directory for the subsets on the disk media. 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.
- Name of the kit's output directory, 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.

Example 6–2 shows the `kitcap` record for the `/dev/none` driver:

Example 6–2: Sample /etc/kitcap Record for CD-ROM or Diskette

```
ESA100HD:c:/: \
  dd=/kit:EasyDriver_none_driver: \
  /easy/output:instctrl:ESANONESTATIC100
```

Based on the information shown in Example 6–2, the `gendisk` utility places the kit on the `c` partition, in the `/` (`root`) directory of the disk media. The product description is `"EasyDriver none driver"`, the kit output directory is named `/easy/output`, and subset control information is in the `instctrl` directory. The kit consists of one subset, named `ESANONESTATIC100`.

6.2 Building a User or Kernel Product Kit on Magnetic Tape Media in tar Format

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. The syntax of the `gentapes` command is as follows:

gentapes [-w -v] [*hostname:*] *product-code special*

- The `-w` option specifies that `gentapes` writes to the tape without verifying it; the `-v` option specifies that the command verifies a tape without writing to it first. If you specify neither option, `gentapes` writes the tape, rewinds it, and verifies its contents.
- The optional *`hostname`* argument is the name of a remote TCP/IP network machine that contains the `/etc/kitcap` file. The `gentapes` utility searches the `/etc/kitcap` file on the remote machine for the *`product-code`* 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 *`product-code`*. If you do not specify a *`hostname`*, `gentapes` looks on your own system. You can use network file system (NFS) file sharing to mount the kit files remotely on a system with the required tape drive.

- The *product-code* is a user-defined code that describes the product. It should match the product name specified in the kitcap record, which is usually a concatenation of the NAME and VERS fields of the key file.
- The *special* argument is the name of the device special file for the tape device, such as /dev/ntape.

The following command produces a kit for the ODB product on a magnetic tape:

```
% gentapes OAT100 /dev/ntape
```

6.3 Building a User or 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 support creation of 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.

The syntax of the gendisk command is as follows:

gendisk [-w -v] [-d] [*hostname:*] *product-code special*

The –w option specifies that gendisk writes to the disk without verifying it; the –v option specifies that the command verifies a disk without writing to it first. If you specify neither option, gendisk writes the disk and verifies its contents.

You can use the gendisk utility to produce kits in either tar or DCD format, depending on whether or not you use the –d option.

The optional *hostname* argument is the name of a remote TCP/IP network machine that contains the /etc/kitcap file. The gendisk utility searches /etc/kitcap on the remote machine for the *product-code* 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 *product-code*. If you do not specify a *hostname*, gendisk looks on your own system. You can use NFS file sharing to mount the kit files remotely on a system with the required disk drive.

The *product-code* is a user-defined code that describes the product. It should match the product name specified in the kitcap record, which is usually a concatenation of the NAME and VERS fields of the key file.

The *special* argument is the name of the device special file for the disk device, such as /dev/rrzla.

6.3.1 Preparing a User or Kernel Product Kit in tar Format

To prepare a kit on disk for a user or kernel product, you use the gendisk utility without the -d option. You specify the product name and the device special file name. For example, the following command creates a kit in tar format for the /dev/none driver on the c partition of the disk named rz0:

```
% gendisk ESA100 /dev/rz0c
```

7

Testing a User or Kernel Product Kit

Before shipping a user or kernel product kit to customers, you should test the installation of the kit by using the same procedures that your customers will use. You should run these tests on hardware configurations that resemble your customers' systems. When you know that the installation procedure works correctly, you should document it and ship it as part of the product kit.

Note

Procedures for testing hardware product kits are documented in Section 5.4.

There are several ways to test a user or kernel product kit:

- The `setld` utility can install a kit either during system installation or after the system is running.
- During system installation, the `setld` utility installs the kit so that it is available for subsequent reboots of the system.
- The `ris` utility integrates a user or kernel product kit into a RIS environment. Client systems can then install the kit from the RIS area by calling the `setld` utility.

This chapter describes how to test the installation of a user product or kernel product kit and how to install a kit in a RIS environment.

7.1 Testing a User Product Kit

To test a user product kit, log in to the system as superuser or `root` and run the `setld` utility. For example, the ODB product could be tested as follows. In this example, the kit is distributed on CD–ROM.

1. Place the CD–ROM in the drive.
2. Create a directory to be the mount point for the CD–ROM, such as `/cdrom`:

```
# mkdir /cdrom
```

3. Mount the CD–ROM on `/cdrom`. For example, if the CD–ROM device were located on the `c` partition of `rz4`, you would enter the following command:

```
# mount -r /dev/rz4c /cdrom
```

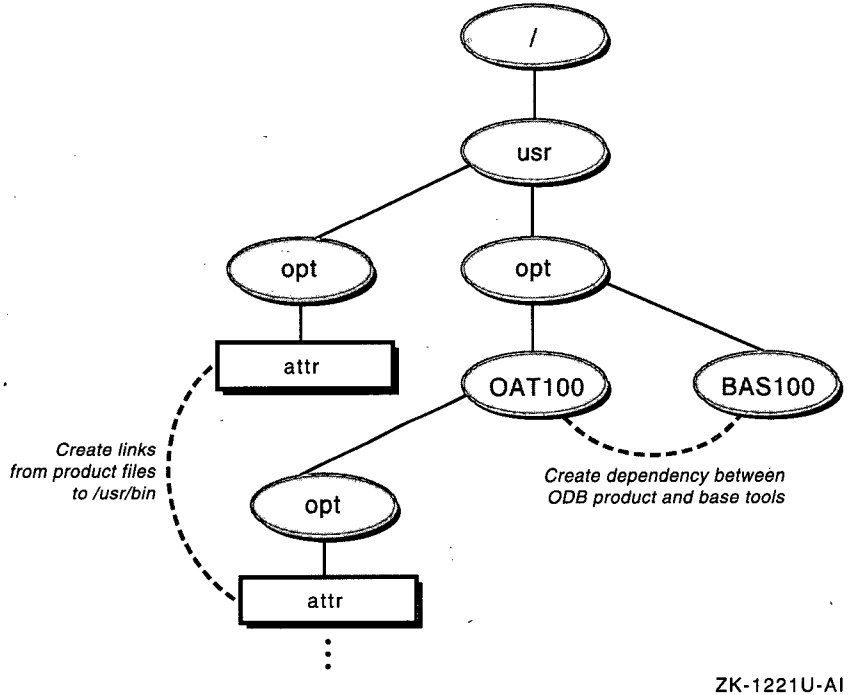
After mounting the CD–ROM, you can change to the `/cdrom` directory and view the directories on the CD–ROM.

4. Install the user product subsets:

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

The `setld` utility displays prompts and messages to guide you through the process of selecting the subsets you want to install. After it loads the subsets, the `setld` utility calls the subset control program for each subset. Figure 7–1 shows the links and dependencies that the ODB subset control program creates.

Figure 7–1: Defining Links and Dependencies for the ODB User Product



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

```
# umount /cdrom
```

See the *Installation Guide* for more information on using the `setld` utility to install layered products.

7.2 Testing a Kernel Product Kit

To test a kernel product kit, log in to the system as superuser or `root` and run the `setld` utility. If the driver is statically configured, you must also reconfigure the kernel to incorporate the driver into the system.

For example, the `edg` driver would be installed as follows, if the kit were distributed on CD–ROM:

1. Insert the CD–ROM in the drive.
2. Create a directory to be the mount point for the CD–ROM, such as `/cdrom`:

   ```
   # mkdir /cdrom
   ```

3. Mount the CD–ROM on `/cdrom`. For example, if the CD–ROM device were located on the `c` partition of `rz4`, you would enter the following command:

   ```
   # mount -r /dev/rz4c /cdrom
   ```

4. Install the device driver subsets:

   ```
   # setld -l /cdrom/ESA100/kit
   ```

   The `setld` utility displays prompts and messages to guide you through the process of selecting the subsets you want to install. After it loads the subsets onto the system, `setld` invokes the subset control program to statically or dynamically configure the driver. Figure 7–2 shows the steps the subset control program takes to statically configure the driver; Figure 7–3 shows the steps the subset control program takes to dynamically configure the driver.

5. Unmount the CD–ROM when the installation is complete:

   ```
   # umount /cdrom
   ```

6. Restart the system with the new kernel:

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

   When the system starts up, the `edg` device driver is available on the system.

Figure 7–2: Statically Configuring a Driver

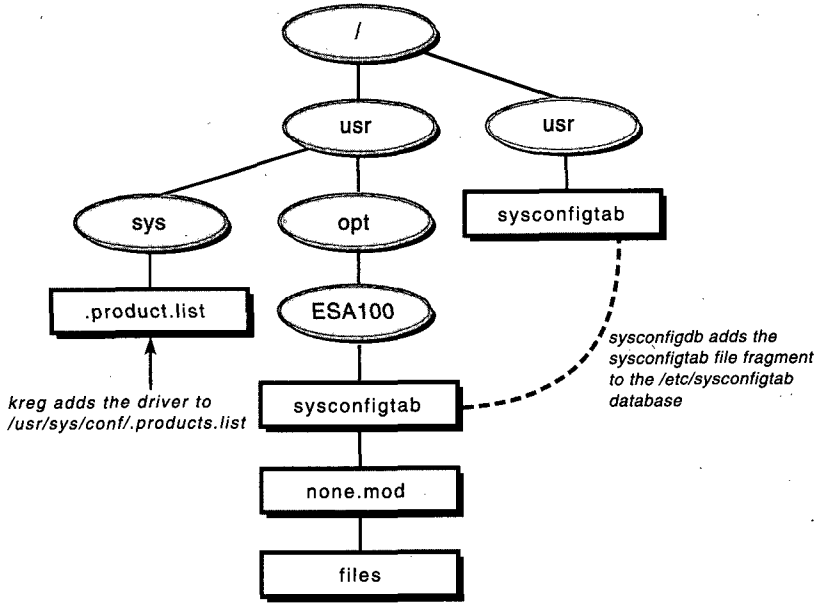
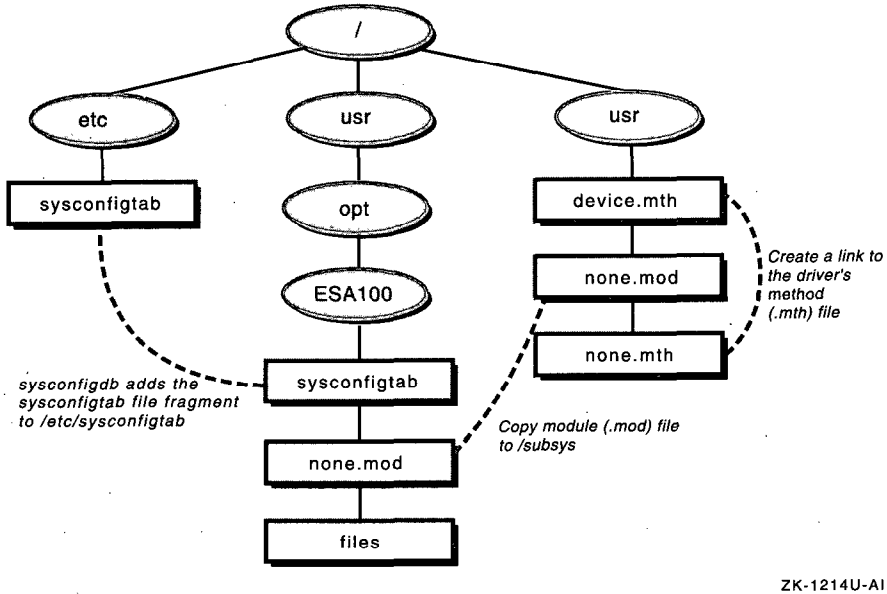


Figure 7–3: Dynamically Configuring a Driver



See the *Installation Guide* for more information on using the `setld` utility to install layered products.

7.3 Testing a User or Kernel Product in a RIS Area

You can use the `ris` utility to test a kernel product kit on a RIS server to be used by RIS client installations.

To install the product in the RIS area on the server, run the `ris` utility as follows:

1. Log onto the server as `root` and invoke the `ris` utility:

```
# /usr/sbin/ris
```

2. Choose `INSTALL software products` from the RIS Utility Main Menu by entering `i` at the prompt:

```
*** 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: i
```

The RIS Software Installation Menu is displayed.

3. Enter option 1, `Install software into a new area` or option 2, `Add software into an existing area`:

```
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:
```

To install the product kit from the RIS server onto the client system, register the client system with the RIS server, then use the `setld` utility, as follows:

1. Run the `ris` utility on the server, and choose `ADD a client` from the main menu:

```
# /usr/sbin/ris

*** 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: a
```

2. Enter the client information requested by the prompts as described in *Sharing Software on a Local Area Network*.
3. When the client is registered to the RIS server, log in to the client as superuser or `root`.
4. Use the `setld` utility to install the product subsets from the RIS area. For example, if the RIS server was named `visier`, you would enter the following command:

   ```
   # setld -l visier:
   ```

   The `setld` utility displays prompts and messages to guide you through the installation process. See the *Installation Guide* for more information on using the `setld` utility to install layered products.

A

Creating a Consolidated CD-ROM

A consolidated 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 CD-ROM.

This release includes the documentation and utilities that you need to build a consolidated CD-ROM in ISO9660–compliant Rock Ridge format. This documentation includes the `disklabel`(8) and `mkisofs`(8) reference pages.

To build a consolidated CD-ROM, you need the following:

1. The operating system CD-ROM in UFS format
2. The appropriate Alpha System firmware update CD-ROM

The following information is included in this appendix:

- How to prepare for the build
- How to build the consolidated CD-ROM
- Sample sessions for both the build preparation and actual building of the consolidated CD-ROM

A.1 Build Instructions

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

A.1.1 How to Prepare for the Build

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.

Note

The examples in this appendix use the C shell.

1. 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 c of rz6 starting at offset 0 with a size of 1300480, and create mount point /cdimage:

```
% disklabel -F -r -e rz6
write new label [y] y
% mkdir /cdimage
```

2. 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 1.5Gb). 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 rz4 as the CD-ROM drive:

```
% mount -r /dev/rz4c /mnt
% cd /mnt
% tar cf /spare/digital_unix_v40f.tar .
% cd /
% umount /mnt
```

A.1.2 How to Build a Consolidated 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. Use the newfs command to initialize a file system on the partition reserved in Step 1 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 2 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 rz6c as the target partition:

```
% newfs /dev/rz6c
% mount /dev/rz6c /cdimage
```

```
% cd /cdimage
% tar xpf /spare/digital_unix_v40f.tar
% cd /
```

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

Note

You cannot repackage firmware or software unless you have a specific licensing agreement with Compaq Computer Corporation which allows you to do so.

a. Mount the appropriate Alpha System firmware update CD-ROM to a temporary mount point such as `/mnt`. For example, using `/dev/rz4c` as the CD-ROM drive:

```
% mount -t cdfs -r /dev/rz4c /mnt
```

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

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

Note

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 /mnt/SMMTABLE.TXT\;1
```

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 appropriate firmware directories in the target directory, and copy each of the platform firmware images that you want from the appropriate Alpha System firmware update 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 appropriate Alpha System firmware update 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
```

3. 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/bin/mkisofs -D -R -a -d -o \
 /spare/consolidate_digital_unix.cdfs /cdimage/
```

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

4. Use the disklabelcommand to insert a label into the file generated in Step 3.

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

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

The CD image file /spare/consolidate_digital_unix.cdfs is ready to be loaded onto 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/rz6c.
- The /spare directory has at least 1.5 Gb of free space.

- The CD-ROM drive is `/dev/rz4`.

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`, and enter the following commands:

```
% cd /
% disklabel -F -r -e rz6
write new label? [y] y
% mkdir /cdimage
```

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

```
% mount -r /dev/rz4c /mnt
% cd /mnt
% tar cf /spare/digital_unix_v40f.tar .
% cd /
% umount /mnt
```

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

A.2.2 Building a Consolidated CD-ROM

Follow these steps to build a consolidated CD-ROM:

1. Log in as `root`, and enter the following commands:

```
% cd /
% newfs /dev/rz6c
% mount /dev/rz6c /cdimage
% cd /cdimage
% tar xpf /spare/digital_unix_v40f.tar
% cd /
```

2. Place the appropriate Alpha System firmware update CD-ROM into the CD-ROM drive, and enter the following:

```
% mount -t cdfs -r /dev/rz4a /mnt
% cp /mnt/SMMTABLE.TXT\;1 /cdimage/smmtable.txt
% more /cdimage/smmtable.txt
```

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

Note

This example uses the same firmware images as Step 2d 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
```

4. Remove the appropriate Alpha System firmware update CD-ROM, and enter the following commands:

```
% /usr/bin/mkisofs -D -R -a -d -o
 /spare/consolidate_digital_unix.cdfs /cdimage/
```

Output is similar to the following:

Note

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

```
Using OSFMANWO.000;1 for \
/cdimage/ALPHA/BASE/instctrl/OSFMANWOS440.scp \
(OSFMANWOP440.scp)
 Using OSFMANWO.001;1 for \
/cdimage/ALPHA/BASE/instctrl/OSFMANWOS440.inv \
 (OSFMANWOP440.inv)
 Using OSFMANWO.002;1 for \
 /cdimage/ALPHA/BASE/instctrl/OSFMANWOS440.ctrl \
 (OSFMANWOP440.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 Oct 22 15:36:59
   5.87% done, estimate finish Fri Oct 22 15:39:24
                .
                .
                .
  99.74% done, estimate finish Fri Oct 22 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)
```

5. Enter the following commands:

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

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

B

Standard Directory Structure

It is recommended that you install files for layered products in the `/opt`, `/usr/opt`, and `/usr/var/opt` directories. Most UNIX based systems use the standard directory structure shown in this appendix. Placing your product within this standard directory structure can help to ensure that your product installs successfully on most customer systems.

Figure B–1 and Figure B–2 show the directories in the standard directory structure. These are the directories that you should use to ensure that your product is portable to other systems.

Note

Some of the illustrated directories are actually symbolic links.

Figure B–1: Base System Directory Structure

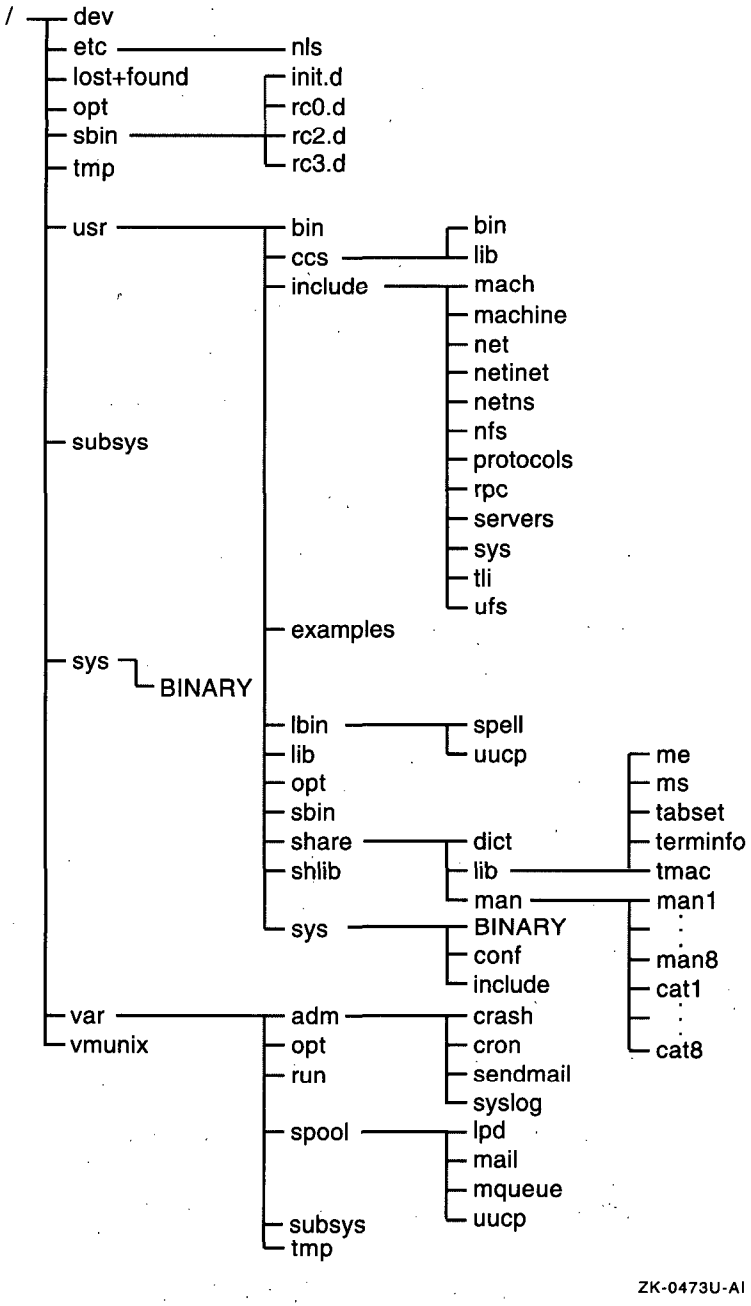


Table B–1 describes the contents and purpose of the directories shown in Figure B–1.

Table B–1: Contents and Purpose of Base System Directories

Directory	Description
/	The root directory of the file system
/dev/	Block and character device special files
/etc/	System configuration files and databases; nonexecutable files
nls/	National language support databases
/lost+found/	Files located by `fsck`
/opt/	Optional for layered products, such as applications and device drivers
/sbin/	Commands essential to boot the system (most of these commands depend on shared libraries or the loader and have other versions in `/usr/bin` or `/usr/sbin`)
init.d/	System state rc files
rc0.d/	The rc files executed for system-state 0
rc2.d/	The rc files executed for system-state 2
rc3.d/	The rc files executed for system-state 3
/subsys/	Dynamically configured kernel modules required in single-user mode
/tmp/	System-generated temporary files, usually not preserved across a system reboot.
/usr/	Most user utilities and applications
bin/	Common utilities and applications
ccs/	C compilation system; tools and libraries used to generate C programs
bin/	Development binaries such as `cc`, `ld`, and `make`
lib/	Development libraries and back ends
include/	Program header (include) files; not all subdirectories are listed in this appendix
mach/	Mach-specific C include files
machine/	Machine-specific C include files
net/	Miscellaneous network C include files
netinet/	C include files for Internet standard protocols
netns/	C include files for XNS standard protocols
nfs/	C include files for Network File System
protocols/	C include files for Berkeley service protocols

Table B–1: Contents and Purpose of Base System Directories (cont.)

Directory	Description
rpc/	C include files for remote procedure calls
servers/	C include files for servers
sys/	System C include files (kernel data structures)
tli/	C include files for Transport Layer Interface
ufs/	C include files for UNIX File System
examples/	Subdirectories of programming examples
lbin/	Back-end executable files
spell/	Spell back-end
uucp/	UNIX-to-UNIX Copy (UUCP) programs
lib/	Links to libraries located elsewhere (/usr/ccs/lib), (/usr/libin), (/usr/share/lib), (/X11/lib); included for compatibility
opt/	Optional layered products, such as applications and device drivers
sbin/	System administration utilities and system utilities
share/	Architecture-independent ASCII text files
dict/	Word lists
lib/	Various libraries
me/	Macros for use with the me macro package
ms/	Macros for use with the ms macro package
tabset/	Tab description files for a variety of terminals; used in /etc/termcap
terminfo/	Terminal information database
tmac/	Text-processing macros
man/	Online reference pages
man1/	Source for user command reference pages
man2/	Source for system call reference pages
man3/	Source for library routine reference pages
man4/	Source for file format reference pages
man5/	Source for miscellaneous reference pages
man7/	Source for device reference pages

Table B–1: Contents and Purpose of Base System Directories (cont.)

Directory	Description
man8/	Source for administrator command reference pages
cat1-cat8	Formatted versions of files in man1 – man8
shlib/	Binary-loadable shared libraries; shared versions of libraries in /usr/ccs/lib
sys/	System configuration files
BINARY	Object files
conf/	Kernel configuration control files
include/	Header files
/var/	Multipurpose log, temporary, varying, and spool files
adm/	Common administrative files and databases
crash/	For saving kernel crash dumps
cron/	Files used by cron
sendmail/	Configuration and database files for sendmail
syslog/	Files generated by syslog
opt/	Optional layered products, such as applications and device drivers
run/	Files created when daemons are running
spool/	Miscellaneous printer and mail-system spooling directories
lpd/	Line printer spooling directories
mail/	Incoming mail messages
mqueue/	Undelivered mail queue
uucp/	UUCP spool directory
subsys/	Loadable kernel modules required in multiuser mode

Table B–1: Contents and Purpose of Base System Directories (cont.)

Directory	Description
tmp/	Application-generated temporary files that are kept between system reboots
/vmunix	Pure kernel executable (the operating system loaded into memory at boot time)

Figure B–2: X Directory Structure

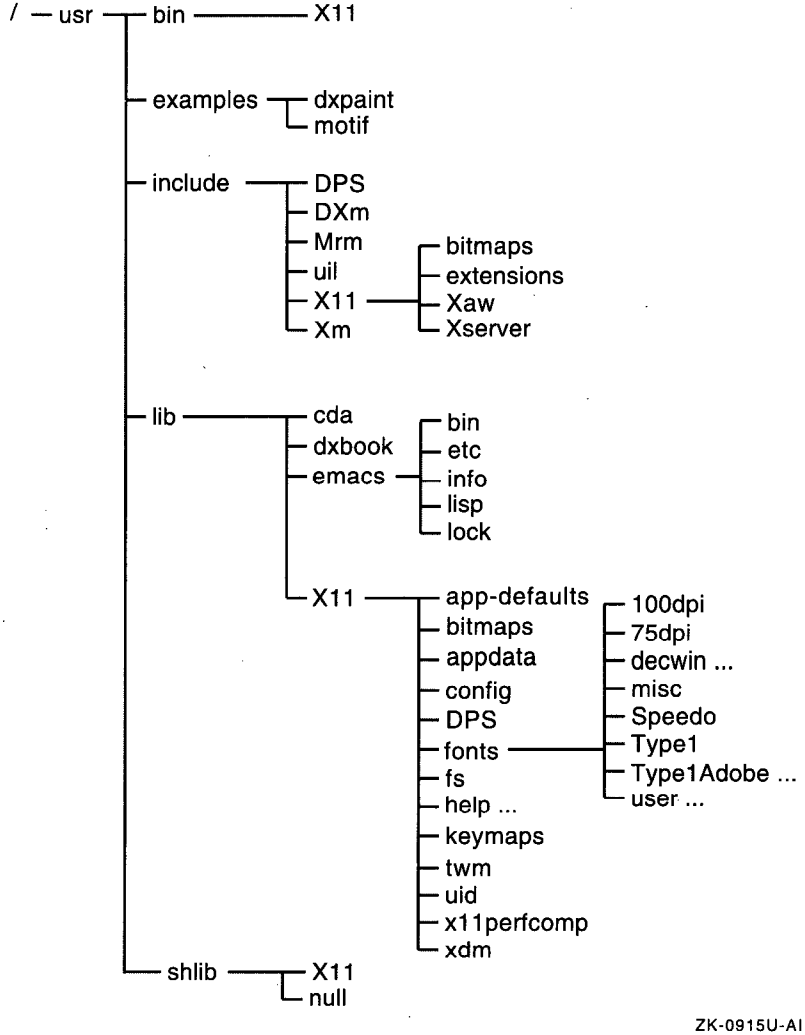


Table B–2 describes the contents and purpose of the directories shown in Figure B–2.

Table B–2: Contents and Purpose of X Directories

Directory	Description
/usr/	Most user utilities and applications
bin/	Common utilities and applications
X11/	X applications
demos/	Miscellaneous demo programs
examples/	Example programs
dxpaint/	Sample Paint image
motif/	Motif example programs
include/	Header files
DPS/	Files for DPS
DXm/	Files for libDXm
Mrm/	Files for libMrm
uil/	UIL header files
X11/	X C header files
bitmaps/	X bitmaps
extensions/	Header files for use with X extensions
Xaw/	Files for libXaw
Xserver/	Header files used for loadable X server libraries
Xm/	Header files for libXm
lib/	Static archive X libraries
cda/	CDA style guides
dxbook/	Default Bookreader bookshelf
emacs/	Emacs directory base
X11	
app-defaults/	System-wide resource files for X client applications
bitmaps/	Program-specific bitmaps
appdata/	Generic program-specific data
config/	Imake configuration files
DPS/	Display Postscript files

Table B–2: Contents and Purpose of X Directories (cont.)

Directory	Description
fonts/	Font files
100dpi/	100 dpi fonts from X Consortium
75dpi/	75 dpi fonts from X Consortium
decwin/	DECwindows fonts
100dpi/	100 dpi fonts
75dpi/	75 dpi fonts
misc/	Fonts from X Consortium
Speedo/	Speedo scalable fonts
Type1/	Type1 scalable fonts
Type1Adobe/	Adobe Type1 scalable fonts
afm/	Adobe font metrics
user	Fonts from layered products and local installations
100dpi/	100 dpi fonts
75dpi/	75 dpi fonts
misc/	Other fonts
fs/	Fontserver config and error log files
help/	Help files for X client applications; subdirectories as applicable
keymaps/	Keymaps for various keyboards
twm/	Default configuration for `twm` window manager
uid/	User Interface Definitions for X client applications
x11perfcomp/	Scripts for analyzing `x11perf` output
xdm/	X Display Manager configuration and resource files, and error log
shlib/	Shareable libraries
X11/	Shareable libraries loaded by X server

Glossary

This glossary defines terms used in this book.

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

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.

C

control files
The collection of files that the `kits` utility places in the `instctrl` directory are referred to as control files. These files include the compression flag file, image data file, subset control file, subset inventory file, and subset control programs.

D

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.

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* **locking** *and* **subset dependency**.

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` database

The `sysconfigtab` database contains information about the attributes of subsystems, such as device drivers. Device drivers supply attributes in `sysconfigtab` file fragments, which get appended to the `/etc/sysconfigtab` database when the subset control program calls the `sysconfigdb` utility during the installation of a kit. *See also* **sysconfigdb utility**.

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.

H

hardware product kit

A hardware product kit contains a hardware product, such as a device driver, that can be installed during the initial installation and bootstrap linking of the operating system. *See also* **kernel product**, **user product** *and* **layered product**.

K

kernel

The kernel is a software entity that runs in supervisor mode and does not communicate with a device except through calls to a device driver.

kernel product

A kernel product is a layered product that runs in kernel space. Users do not directly run kernel products, but the operating system and utilities access them to perform their work. *See also* **layered product hardware product kit** and **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**.

`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**, and **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 kit**, **kernel product**, and **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.

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* **data directory**, **newinv utility**, and **subset**.

N

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**.

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.

osf_boot utility

The osf_boot utility performs the initial installation and bootstrap of the operating system.

R

RIS

Remote Installation Services. Lets administrators install software kits into a RIS area for subsequent installation onto client systems over a network. Using a RIS server makes installation of layered products faster and easier for all the clients on the network.

Remote Installation Services

See **RIS**.

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 allows the transfer of the contents of a layered product kit to a customer's system.

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

A subset is the smallest installable component of a product kit for the `setld` 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** and **locking**.

`sysconfigdb` utility

The `sysconfigdb` utility is a system management tool that maintains the `sysconfigtab` database. *See also* **/etc/sysconfigtab database**.

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* **layered product** and **kernel product**.

Index

A

B

C

D

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.

To place an internal order, go to 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-MT4AA-G8
Tru64 UNIX Documentation Kit	QA-MT4AA-GZ
End User Documentation Kit	QA-MT4AB-GZ
Startup Documentation Kit	QA-MT4AC-GZ
General User Documentation Kit	QA-MT4AD-GZ
System and Network Management Documentation Kit	QA-MT4AE-GZ
Developer's Documentation Kit	QA-MT5AA-GZ
General Programming Documentation Kit	QA-MT5AB-GZ
Windows Programming Documentation Kit	QA-MT5AC-GZ
Reference Pages Documentation Kit	QA-MT4AG-GZ
Device Driver Kit	QA-MT4AV-G8

Reader's Comments

Tru64 UNIX
Guide to Preparing Product Kits
AA-QYW7C-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 9987

------- Do not cut or tear - fold here and tape -------

Cut on this line