Digital UNIX

Guide to Preparing Product Kits

Part Number: AA-QYW7B-TE

December 1996

Product Version: Digital UNIX Version 4.0 or higher

This book describes the procedures for creating, maintaining, and installing layered product kits.

Digital Equipment Corporation
Maynard, Massachusetts

Contents

Examples

Figures

Tables

About This Manual

This manual describes the procedures for creating, installing, and managing product kits to be installed on Digital UNIX® systems.

Audience

This manual is primarily for kit developers responsible for creating product kits. This manual assumes you are a moderately experienced user of the Digital UNIX system with knowledge of system administration.

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 the kit directory and build the product kit.
Chapter 3	Creating and Managing Subset Control Programs
	Describes how to write subset control programs (SCPs) to install and manage software subsets.
Chapter 4	Creating Subsets
	Describes how to create subsets and subset control files with the `newinv` and `kits` utilities.
Chapter 5	Producing Distribution Media
	Describes how to produce a product kit on the distribution media.
Chapter 6	Installing the Kit
	Describes how to install the product kit on the target system.
Appendix A	Digital UNIX Standard Directory Structure

	Describes the standard directory hierarchy of Digital UNIX systems.
Glossary	Defines terms used in this manual.

Related Documents

The printed version of the Digital 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 Digital.) 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, Glossary, and Master Index* provides information on all of the books in the Digital UNIX documentation set.

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

- *Installation Guide*

  This manual describes the procedures to perform an update installation, a basic installation, or an advanced installation of the Digital UNIX product on all supported processors. It explains how to prepare your system for installation, boot the processor, and perform the installation procedure. It also discusses system management procedures in a standalone environment.

- *System Administration*

  This manual describes how to configure, use, and maintain the Digital UNIX 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 for the system

administrators responsible for managing the operating system. It assumes a knowledge of operating system concepts, commands, and configurations.

- *Sharing Software on a Local Area Network*

  This manual describes Remote Installation Services (RIS) and Dataless Management Services (DMS). The RIS utility is used for installing 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 foreign device 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 Digital UNIX 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.

- *Reference Pages Section 8 and 1m*

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

- *Release Notes*

  The release notes describe problems you might encounter when working with the Digital UNIX system and 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 for the person installing the product and for anyone using the product following installation.

Reader’s Comments

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

You can send your comments in the following ways:

- Fax: 603-881-0120 Attn: UEG Publications, ZK03-3/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:

  Digital Equipment Corporation
  UEG Publications Manager
  ZK03-3/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 Digital UNIX that you are using.
- If known, the type of processor that is running the Digital UNIX software.

The Digital 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 Digital technical support office. Information provided with the software media explains how to send problem reports to Digital.

Conventions

The following typographical conventions are used in this manual:

`#`	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.

1

Introduction

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 a Digital UNIX system. The kit can be distributed on a CD–ROM, diskette, or tape for installation on the customer's system. Finally, a kit can be installed on the Digital UNIX system at single-user time, multiuser time, installation time, or when setting up a **Remote Installation Services** (RIS) environment.

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 Digital UNIX system?
- Will it be integrated into a RIS environment?

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 Digital UNIX system supports.

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

1.1 Product Types

The kitting process described in this book lets you deliver layered products for the Digital UNIX 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, then calls the appropriate driver interface.

- Foreign device

  A **foreign device** is a peripheral device installed during the initial system installation and bootstrap. Before a system manager can make use of the foreign device, the associated device driver must be configured into the kernel. The problem is that prior to the installation of the operating system, there is no kernel and there are no device drivers available to handle potential kernel and user requests of these foreign devices.

  To solve this problem, a kit for a foreign device contains a kernel product — the device driver for the foreign device — and other files needed for configuring the driver into a kernel at system installation time.

  The following are examples of hardware needed during the initial installation and bootstrap of the operating system:

  – Graphics controller

    A graphics controller is the hardware interface between the computer and a graphics terminal.

  – Disk controller

    A disk controller is the hardware interface between the computer and a disk device. (Currently, disk devices are not supported as foreign devices.)

  – Network controller

    A network controller (when bootstrapping from the network) is the hardware interface between the computer and a network device. (Currently, network controllers are not supported as foreign devices.)

1.2 Kit Formats

Prior to 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.

- Direct CD–ROM (DCD) format

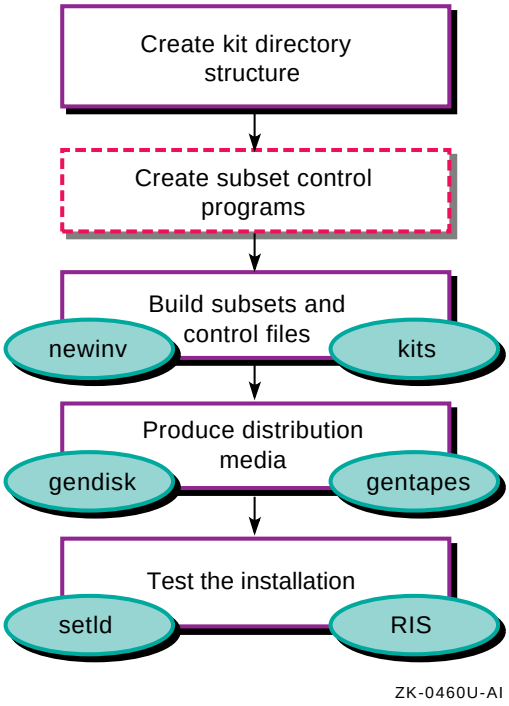
  In DCD format, the files are written to any disk media (CD–ROM, hard disk, or diskette) as a UNIX file system. Subsets distributed in DCD format cannot be compressed. The `gendisk` utility can create kits in DCD format.

Kits for user and kernel products should be produced in `tar` format; foreign device kits must be produced in DCD format.

1.3 Kit-Building Process

Figure 1–1 illustrates the process of creating and packaging a kit. In the figure, dashed boxes 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 the figure, the commands enclosed in ellipses perform the indicated steps of the kit-building process.

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



The kit-building process is divided into 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. A **master inventory file** lists all the product files and the subsets in which they belong. A **key file** identifies the product that the kit represents.
   - 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 foreign device kits. Only the contents of these directories differs among the product types. For example, a foreign device kit needs additional files that are unique to this specific kit type.

2. Creating subset control programs

   The `setld` utility can call a **subset control program** to perform installation steps specific to your kit. This program is optional for user products and kernel products. You supply it on your kit only if the product requires special installation steps. The program is required for RIS installations of foreign device kits. 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, while the subset control program for a user product would not.

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 file and key file. The `newinv` utility can help you maintain the master inventory.

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. You do this by selecting the appropriate packaging utilities, such as `gendisk` or `gentapes`. If you are creating a kit for a foreign device, you must also modify the kit and add files for the `osfboot` utility's bootstrap link support.

5. Testing the installation of the kit

   After you have successfully created the kit, you should test the installation. For user products and kernel products, you install the kit by running the `setld` utility. For foreign devices, you use the `osfboot` utility's bootstrap link technology to install the kit and bootstrap a custom kernel. You may also want to install the kit on a RIS server so that RIS clients can install it across a network.

1.4 Sample Products

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

- Orpheus Document Builder (ODB)

  This application is one of a set of applications that the fictitious company, Orpheus Authoring Tools, Inc., produces. This example shows how to build a kit for a user product.

- The /dev/none device driver

  Writing Device Drivers: Tutorial introduces this peripheral device driver, which the fictitious company, EasyDriver, Inc., produces. This example shows how to build a kit for a kernel product.

- The /dev/edgd device driver

  This graphics device driver, which EasyDriver, Inc. also produces, shows how to build a kit for a foreign device.

2

Creating the Kit Directory Structure

When engineers finish developing a product, they give the product files to you 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 of Digital UNIX systems and how to create a kit-building directory structure to fit within the standard directory structure.

2.1 Designing the Customer's Directory Structure

You can 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 Digital UNIX system directory structure is set up for efficient organization. It separates files by function and use. You should install product files in subdirectories of `/usr/opt`, `/var/opt`, and `/opt`, as follows:

- Read-only files

  Files that are nominally 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

  Files that users can read and write, such as lists of employee telephone numbers, are installed in a product-specific subdirectory of `/var/opt`.

- Boot files

  Files that are required at bootstrap time, such as device drivers, are installed in a product-specific subdirectory of `/opt`.

The name of the product-specific subdirectory should consist of a 3-character product or company code and a 3-digit version code, as specified in the key file (see Chapter 4). For example, the product-specific subdirectory names for the ODB product kit are `/opt/OAT100`, `/usr/opt/OAT100`, and `/usr/var/opt/OAT100`.

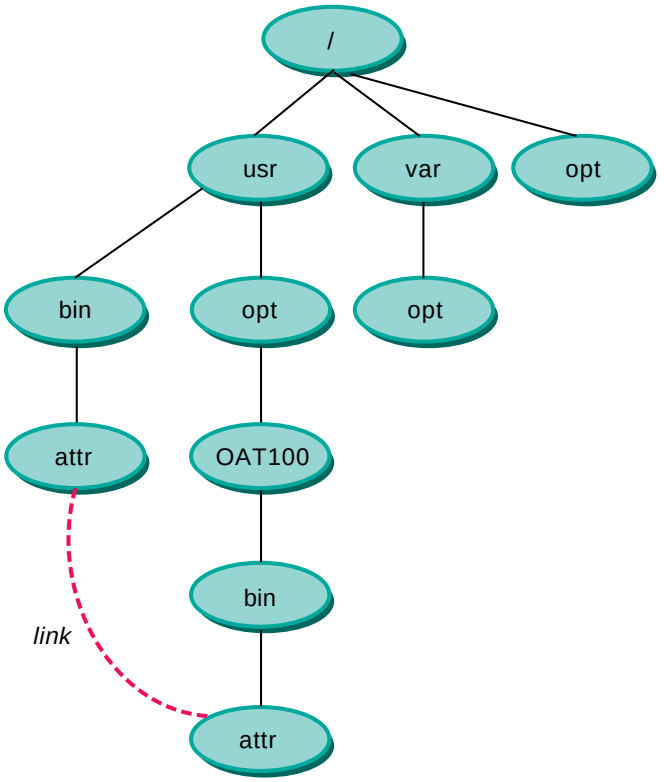
Using this standard directory structure has the following advantages:

- 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 `/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.

For users to make effective use of the file system after they install your product, files should be in directories that are in the normal search path as specified by the user's `.profile` or `.login` files, as appropriate. Your product directories would not ordinarily be in the user's search path. Therefore, the installation procedure must create links from the directory where your product files reside and the directory where users will search for them. The subset control programs create these links during product installation.

Figure 2–1 shows how the Orpheus Document Builder (ODB) product is installed in the standard directory structure, under `/opt`, `/usr/opt`, and `/var/opt`. A symbolic link makes each file accessible through the standard directories. For example, the ODB kit's `/usr/bin/attr` command is a symbolic link to `/usr/opt/OAT100/bin/attr`.

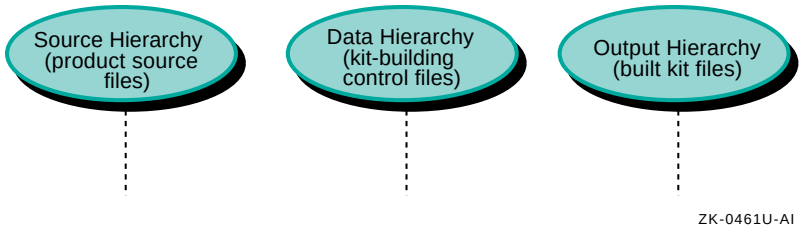
Figure 2–1: Linking Product Files to Standard Digital UNIX Directories



2.2 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 2–2.

Figure 2–2: Kit Directory Structure



The following list 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.

  There is no specific requirement for the location of the data hierarchy, but it is good practice to place it under the same directory as the source hierarchy.

- Output hierarchy

  The **output hierarchy** 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 `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.

Create the kit-building directory structure as follows:

1. Issue the appropriate `mkdir` commands for each of the directories and subdirectories that you need.
2. Populate the `src` directory with all the files that are to be part of the finished kit. Section 2.2.1 describes the files you need for a user product. Section 2.2.2 describes the files you need for a kernel product. Section 2.2.3 describes the files you need for a foreign device kit. You can choose any appropriate method for populating the source hierarchy. For example, you could create a makefile for use with the `make` command.

Caution

File attributes, such as ownership and permissions, for all the files and directories in the source hierarchy must be exactly as they should be on the customer's system. Usually, this requirement means that you must be a 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 the values they have in the kit's master inventory files.

Under most circumstances, your kit should not include any files whose pathnames exactly match those of existing system files. If you do, the kit's files are installed in place of the existing files.

2.2.1 Directory Structure for a User Product Kit

A user product should be designed so that the user sees it as an integral part of the system. This means that, as with base-system software, you should place programs such as commands and utilities in directories that are part of the normal search path, such as `/usr/bin`. Similarly, you should place libraries in directories where users would expect to find them, such as `/usr/lib`.

The example ODB kit places command files in the standard system directory (`/usr/bin`), the product's documentation in a directory created by another user product (`/usr/lib/br`), and template files for users employing the product in a directory unique to the ODB product (`/usr/share/doclib/templates`).

The actual files for the ODB kit, however, are not physically located in any of the directories listed in the preceding paragraph. The files are installed in directories under `/opt`, `/usr/opt`, and `/var/opt` so that the files are centrally located and easy to find. Then a symbolic link is created for each file that makes the file accessible through the standard system directories.

The ODB kit contains files to be installed in the following directories:

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

Figure 2–3 illustrates the complete directory structure for the ODB kit. In this figure, the dashed directory, `dcb_tools`, represents the existing directory under which you would create the source hierarchy's directories as shown.

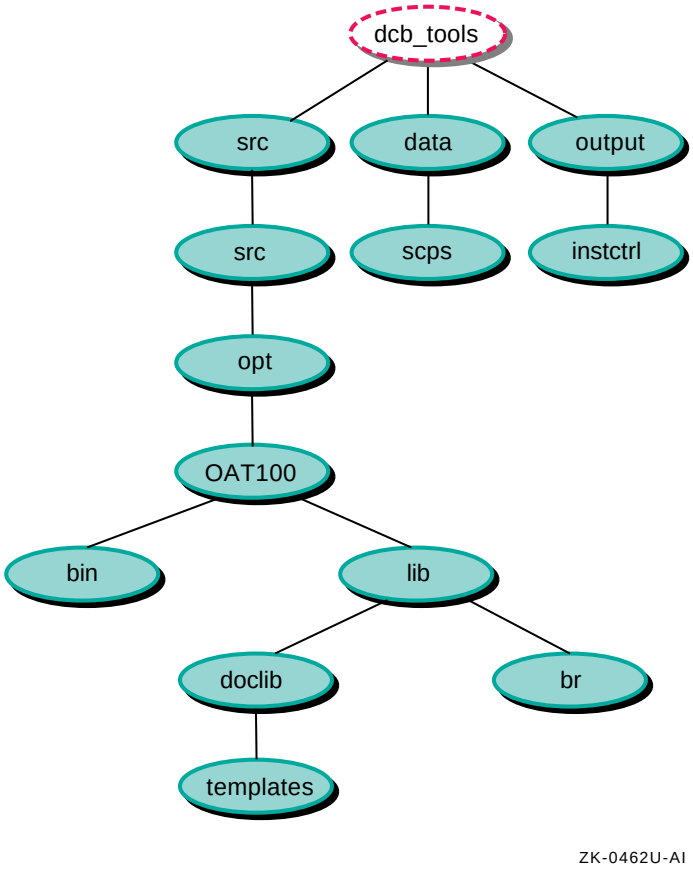
The `src` directory represents the root directory on the customer's system; the `usr` directory represents `/usr` on the customer's system. All the other directories in the source hierarchy are mapped to the customer's system in the same way.

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 3-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.

File names in the standard directory structure, where users would usually expect the files to appear, are linked symbolically to the actual files installed on the customer's system. For example, the command named `/usr/bin/attr` exists as a link to `/usr/opt/OAT100/bin/attr`.

If the ODB kit included user-writable files, they would be placed under `/var`, and there would also be a `/var/opt/OAT100` directory to contain those files. Digital recommends this convention for consistency among user products.

Figure 2–3: Directory Hierarchy for the ODB Kit

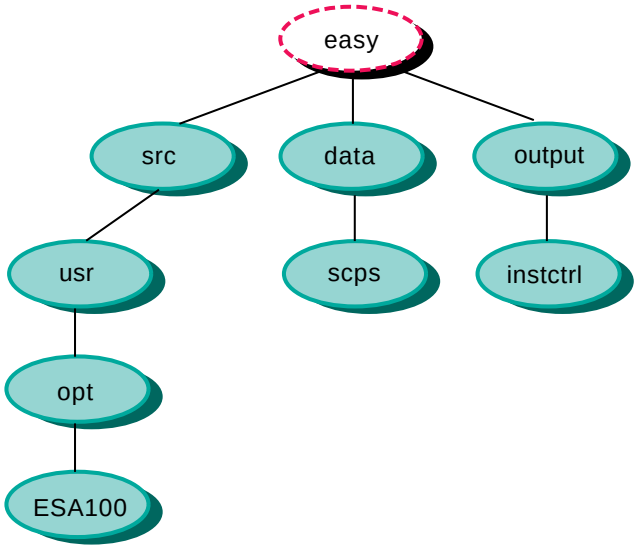


2.2.2 Directory Structure for a Kernel Product Kit

You set up a kit directory structure for a kernel product in the same way as you would for a user product. You create three directory hierarchies — source, 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 `/var/opt`, as appropriate. Unlike a user product, the kit for a kernel 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 introduced in *Writing Device Drivers: Tutorial*.

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



ZK-1198U-AI

The top-level directory (`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:

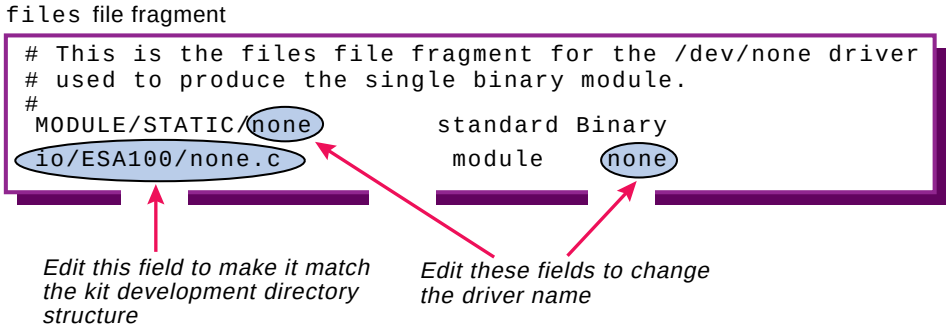
- A statically configured driver is configured in the bootpath. It is built into the kernel. The kit for a statically configured driver can contain either driver source files, single binary modules, or both. For example, a device driver for a foreign device is statically configured into the kernel. However, it cannot be distributed in source form, only single binary (`.mod`) form.
- A dynamically configured driver is configured at run time. It is added into an existing kernel but is not a permanent part of the kernel. Whenever the system restarts, the system manager must reconfigure the driver. Kits for dynamically configured drivers contain only single binary modules, not the source files for the driver.

The following list describes the files that go into a device driver kit, 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 the product directory, such as `ESA100`. 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.

Figure 2–5: Editing the files File Fragment

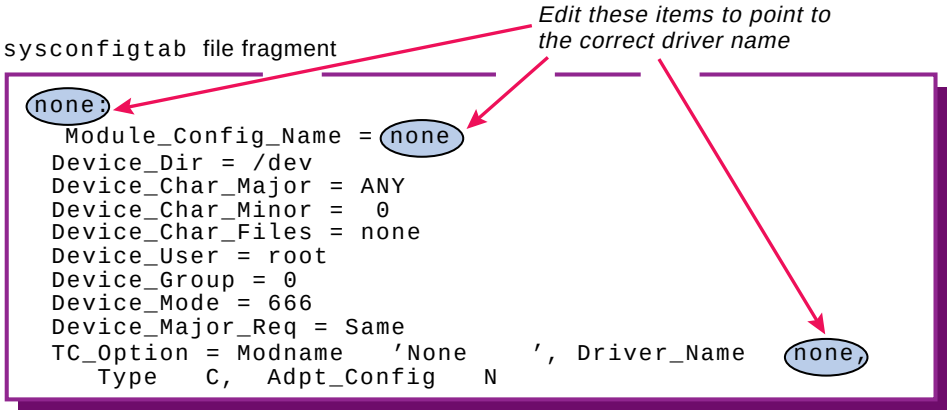


ZK-1199U-AI

- `sysconfigtab` file fragment

  Contains device special file information, bus option data information, and information on contiguous memory usage for statically and dynamically configured drivers. When the user installs a kernel product kit, the driver's `sysconfigtab` file fragment gets appended to the `/etc/sysconfigtab` database. You should place this file fragment in the product directory, such as `ESA100`. 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 the product directory, such as ESA100. The subset control program references or copies this file into the customer's BINARY and subsys directories when the driver is installed.

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

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

- *driver*.mth method files

  Contain driver methods that are called during autoconfiguration to create device special files for dynamically configured drivers. These files do not appear on 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 writer 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.

2.2.3 Directory Structure for a Foreign Device Kit

A foreign device kit contains all the files for a kernel product, but it requires the following additional files on the media to make the media accessible during initial system installation and bootstrap:

- A *name*.kit file, where *name* is based on the device name. You provide this file on the foreign device kit to control the actions of the osfboot

utility. The osfboot utility configures device drivers into a temporary kernel so that system managers can use devices that would not otherwise be available during the initial installation and bootstrap of the operating system. You place this file on the foreign device kit media after creating the kit media with the gendisk utility. Because this file is not part of the product and not installed on the system, it does not belong in the kit directory structure.

- Additional *name*.kit files for any other foreign device kits on the same distribution media. A single media may contain several kits and several *name*.kit files.
- An /etc/sysconfigtab database. This database defines attributes of the modules that osfboot configures. Like the *name*.kit file, this file does not belong in the kit directory structure.
- Links to modules in the kit directory structure. Remember that a foreign device kit is shipped as a UNIX file system, so it contains the entire kit-building directory structure, not only the subsets. By creating links, you save space because you do not create duplicate copies of files.
- Possibly a *kitname*.kk file, where *kitname* is the product name. If the foreign device kit will be installed on a RIS server, you must supply this file in the instctrl subdirectory.

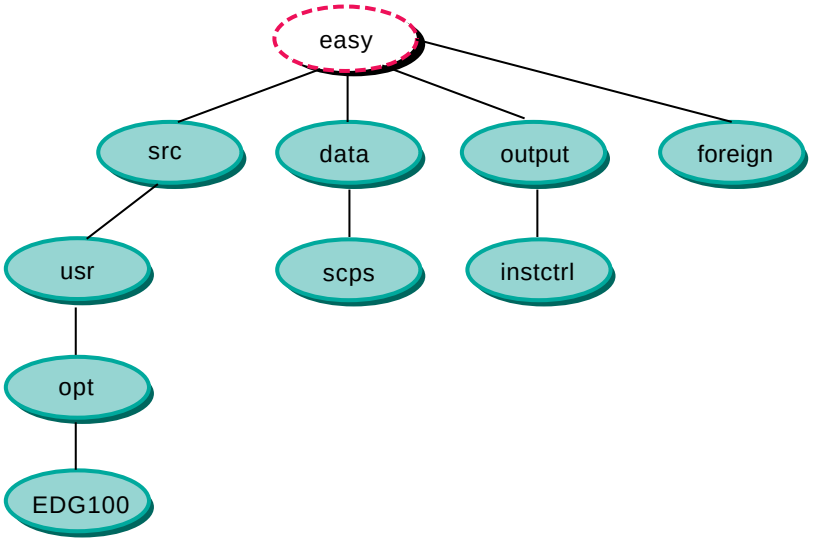
Note

Foreign device kits require several files that are not part of the kit directory structure. These files are not installed on the customer's system, but are used by the osfboot utility.

Neither the *name*.kit file, the *kitname*.kk file, the /etc/sysconfigtab database, nor the links are placed on the customer's system when setld runs; the osfboot utility and the Remote Installation Services (RIS) procedure use them at installation time.

Figure 2–7 shows the directory structure for the /dev/edgd graphics device driver product.

Figure 2–7: Directory Structure for the /dev/edgd Foreign Device Kit



The following sections describe the contents of the *name*.kit and *kitname*.kk files.

2.2.3.1 The name.kit File

Commands in the *name*.kit file describe how osfboot needs to modify the bootstrap link process to build this kit. When bootstrap linking from a foreign device kit, osfboot sets the default directory to the kit root directory. (Ordinarily, the default is /sys/BINARY on the system disk.) Commands in the *name*.kit file indicate which modules should be added, removed, or replaced in the kernel. When you specify the modules for the device driver, the directory path is relative to the kit directory by default.

Commands in the *name*.kit file have the following form:

+[*device*:] [*/path/*]*file*.mod

Adds *file*.mod from the root or the specified device. You can specify a full path or accept the default.

–*file*.mod

Deletes *file*.mod from the module list on the default path for *file*.mod.

file.mod=[*device*:][*/path/*]*new*.mod

Replaces *file*.mod on the default path with the module you specify.

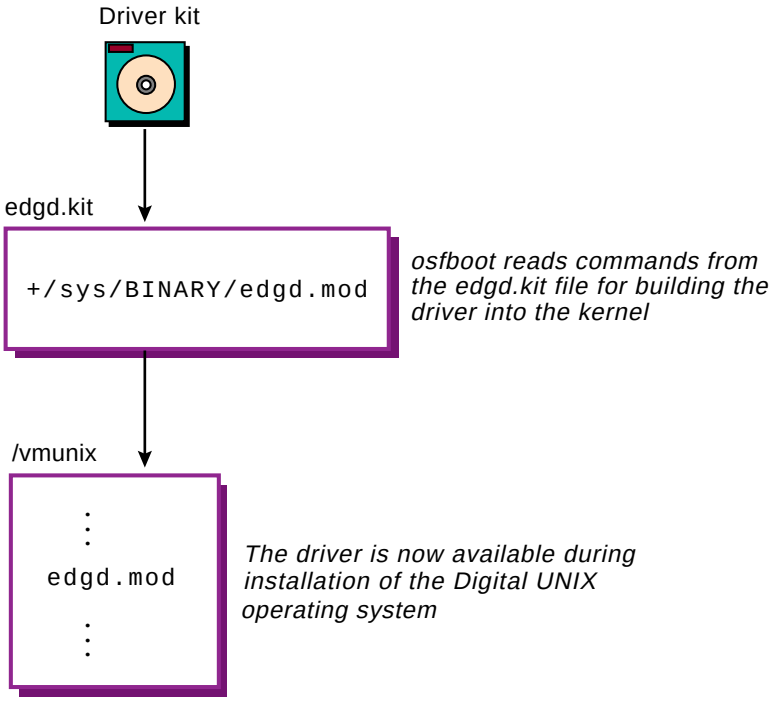
The foreign device kit for the /dev/edgd driver supplies one *name*.kit file, which adds the edgd.mod single binary module to the kernel. In the following example, /sys/BINARY refers to the directory on the kit, not the system disk:

```
+/sys/BINARY/edgd.mod
```

Figure 2–8 shows how the *name*.kit file works with the osfboot software during installation of a foreign device kit.

In the figure, the driver kit contains a *name*.kit file called edgd.kit. The edgd.kit file instructs the osfboot utility to build and configure a temporary /vmunix kernel that includes the /dev/edgd driver. Upon completion, this temporary /vmunix kernel makes the /dev/edgd driver available to handle user and system requests of a specific hardware device during the installation of Digital UNIX.

Figure 2–8: Using a name.kit File During System Installation



2.2.3.2 The kitname.kk File

The Remote Installation Services (RIS) utility provides the ability to install foreign device kits into a RIS area for subsequent installation on a client.

For more information about this feature, see *Sharing Software on a Local Area Network*.

If you want to allow the device driver product to use this feature, create a *kitname*.kk file in the instctrl directory. The file may be empty, but it must exist. It indicates to RIS that a foreign device kit exists.

3

Creating Subset Control Programs

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

- Some of your kit's files must be customized before the product will work properly.
- You want to offer the user the option of installing 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`.
- You want to provide RIS support for a foreign device.

A subset control program can perform all of these tasks. Layered product kits designed according to the guidelines in Chapter 2 must have subset control programs to create the required links.

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

3.1 Common Characteristics of a Subset Control Program

Regardless of the specific tasks that 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 that Digital UNIX supplies.
- They are invoked at various times by the `setld` utility.
- If errors occur, they must exit and return an error status to `setld`.

- 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, Digital recommends 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

Digital UNIX provides a set of routines in the form of Bourne shell script code. These routines are in the file `/usr/share/lib/shell/libscp`.

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

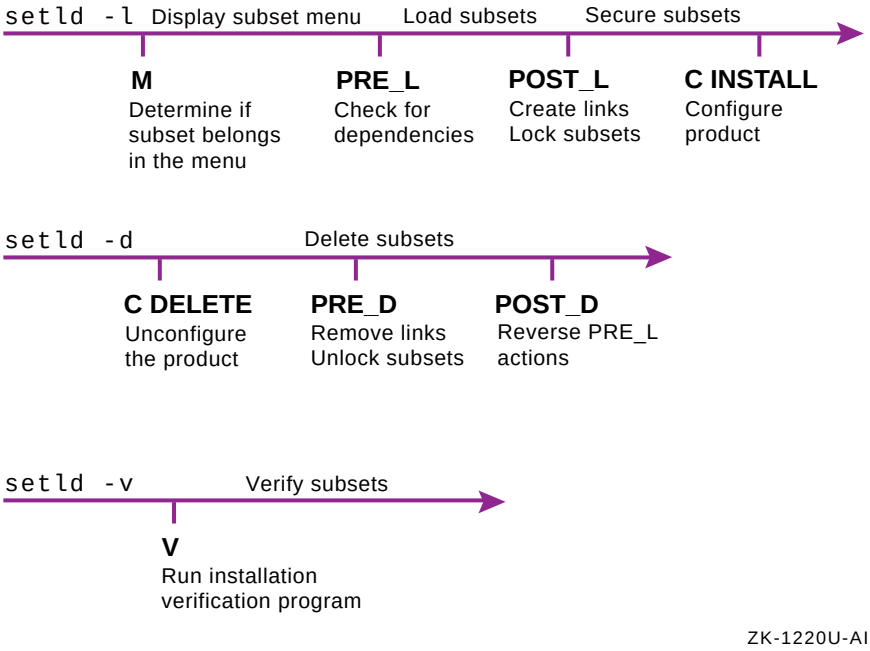
Your kit does not need to do anything to invoke its subset control program. The `setld` utility invokes it during various phases of the installation procedure. The subset control program can perform any tasks that it needs during a phase, such as creating or deleting a file or displaying messages. Certain 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 time lines of the `setld` utility when invoked with the –`l`, –`d`, and –`v` options. The actions that `setld` takes are written above the lines; the value of the `ACT` environment variable and the actions that the subset control program takes at each phase are written below the lines.

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, `setld` 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 `setld` continues.

Figure 3–1: Time Line of the setld Utility



3.1.4 Aborting the Program

Depending on the tests it makes, your subset control program could decide at some point to abort 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 abort the process.

To abort the installation or deletion of the subset, the subset control program must return a nonzero status to `setld` upon exiting from the particular phase for which it was called. If the subset control program returns a status of 0 (zero), `setld` 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 Building 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 Authoring Tools
_ROOT	The root directory of the installation
_SMDB	The location of the subset control files, ./usr/.smdb.
_INV	The inventory file, for example, OATODB100.inv
_CTRL	The subset control file, for example, OATODB100.ctrl
_OPT	The directory specifier /opt/
_ORGEXT	File extension for files saved by the STL_LinkCreate routine, set to pre$_PVCODE
_OOPS	The NULL string, for dependency checking

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 Dataless Environment

In a dataless 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.

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 RIS 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 /, 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 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, setld 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 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 by Digital for extraction of the subset into a 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. The subset control program checks to see if it is running on Digital's Alpha processor.

```
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.

For example, the subset might contain files with the same names as existing files. Duplicating existing file names is usually considered a poor practice. However, you might do this when installing a kit that contains

binary files that would usually be installed by other kits but which must be replaced when your kit is installed.

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 `setld` can both install and remove subsets, the 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, `setld` usually removes them without question, leaving your product 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 3.2.3.3).

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

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 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
```

The elements of a dependency expression (*depexp*) are as follows:

wc_subset_id

Represents a subset identifier that can contain file name expansion characters (asterisks, question marks, or bracketed sets of characters) 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 are usually 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.

As indicated in Chapter 2, a layered product's files should be installed in the `/usr/opt` and `/var/opt` areas and accessed by means of symbolic

links in the standard UNIX directory structure, such as /usr/bin. These symbolic links, referred to as **forward links**, must be created during the POST_L phase, after the referent files are in place. Do not try to create these links during the C INSTALL phase because the /usr file system is not guaranteed to be writeable at that time. If your product includes links in /var, create these links also in POST_L. To maintain symmetry, you must remove links during the PRE_D phase, not during the C DELETE phase.

Symbolic links for layered products are usually created in the standard UNIX directories to refer to files that are actually in the layered product areas /usr/opt and /var/opt. These links are relatively straightforward.

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 can themselves 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 an 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 Forward Links

To create a forward link, you must first call the STL_ScpInit routine to initialize global variables, then call the STL_LinkCreate routine. The STL_ScpInit routine looks in the master inventory file for any entries flagged for linking. The entry in the master inventory indicates where the file has been installed. The STL_LinkCreate routine creates a link to that file from a directory within the standard directory structure, such as /usr/bin. Thus, for the STL_LinkCreate routine to work correctly, you must specify the files that require symbolic links by setting the link bit in the master inventory file. (See Chapter 4 for more information on the master inventory file.)

Any nonempty directories in the inventory should leave the link bit unset (set to 0) to maximize the performance of STL_LinkCreate. See

Example 3–2 for a subset control program that creates and removes symbolic links.

The following routines create forward links:

STL_LinkInit

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

STL_LinkCreate

Creates forward links from the installed system to the product areas, such as the /opt areas. Call STL_ScpInit first to initialize required global variables. A forward link from the system to the product areas (under /usr/opt or /var/opt) is created for each file whose link flag is set in the master inventory file. For example, the link bit of the ./usr/opt/OAT100/bin/attr file is set as follows in the master inventory file:

```
4        ./usr/opt/OAT100/bin/attr                OATODB100
```

After STL_LinkCreate runs, a symbolic link from ./usr/bin/attr points to ../../usr/opt/OAT100/bin/attr. If a file already exists in the same name space, setld saves it before the link takes its place. In the previous example, if a ./usr/bin/attr file already exists, it is saved to ./usr/bin/attr.preOAT100 before the link gets created. All links are created relative to the install root and are dataless safe.

3.2.3.2 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. You can use `STL_LinkBack` repeatedly to create as many links as required. *`link_file`* is the name of 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. This routine returns no status.

Example 3–1 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 3–1: Example of 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
```

3.2.3.3 Locking Subsets

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

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

When it installs a subset, `setld` 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 Authoring Tools base product be installed for the ODB product to work

properly. Suppose that the OATBASE200 subset is present. When setld 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 setld to remove the OATBASE200 subset, setld checks OATBASE200.lk and finds a record that indicates that OATODB100 depends on OATBASE200. Then setld 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, setld 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 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. At this time the subset control program can perform tests to verify that the subset is installed correctly.

The setld utility verifies the size and checksum information for each file in the subset during loading (when the user invokes setld with the –l option). Therefore, the setld utility does not call the subset control program for verification during the installation process. However, in a kit that contains multiple subsets, the last subset control program to be called could execute an installation verification program (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 unconfigure 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 `setld` 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 into which this information was placed when the kit was built. During 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, do so with extreme care. Bits 0 through 7 of the flags field are reserved by the `setld` utility, and you should 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=34816
```

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:

```
#! /sbin/sh

. /usr/share/lib/shell/BitTest

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

3.4 User Product Subset Control Program

Example 3–2 shows a subset control program for the ODB product. This program illustrates one correct method for obtaining 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 setld invokes the program at these times, the program simply exits with a success status.

Example 3–2: Subset Control Program for the ODB 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 3–2: Subset Control Program for the ODB 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 Authoring Tools (${_PCODE}TOOLS)"
    }

    STL_DepEval ${_PCODE}BASE[2-9]?? ||
    {
        _OOPS="$_OOPS
    Orpheus Authoring 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
    STL_LinkCreate

    # 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 3–2: Subset Control Program for the ODB Product (cont.)

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

    # dependency unlocking 8
    STL_LockInit
    STL_DepUnLock $_SUB ${_PCODE}TOOLS??? ${_PCODE}BASE[2-9]?? and
    ;;

esac

exit 0  9
```

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 setld.

3 For the M phase, if the installation is running on an Alpha platform, allows setld to continue. If not, the subset control program returns a nonzero status and exits. As a result, setld 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 setld, which aborts 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 and STL_LinkCreate routines. 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 setld 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 and STL_LinkRemove routines to remove the symbolic links that STL_LinkCreate created during the POST_L phase.

8 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.

9 Ensures that the subset control program returns a success status to setld 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 Kernel Product Subset Control Program

In addition to the optional processing described in Section 3.4, a subset control program for a kernel product such as a device driver must also configure the driver into the kernel. When building subset control programs for a kernel product, 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 3–3 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 3–3: 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
C)
```

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

```
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

        # Add the files associated with the statically configured
        # /dev/none device driver product to the customer's
        # /etc/sysconfigtab database
        sysconfigdb -a -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
        # Add the files associated with the dynamically configured
        # /dev/none device driver product to the customer's
        # /etc/sysconfigtab database
        sysconfigdb -a -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 *****"
        echo "*****                                      *****"
```

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

```
        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 During the C INSTALL phase, displays a menu of installation options. The user can choose to install the driver for static configuration or dynamic configuration.

3 If the user chooses menu item 1, performs a static configuration.

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 3-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.

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 –a flag

  This flag causes sysconfigdb to add 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 If the user chooses menu item 2, performs a dynamic configuration.

8 Calls `sysconfigdb` to add 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 During the `C DELETE` phase, displays a menu of options for deleting subsets. 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 If the user chooses menu option 1 (delete a statically configured driver), calls the `kreg` utility to deregister the driver with the kernel. 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 If the user chooses menu item 2 (delete a dynamically configured driver), calls the `sysconfig` utility with the `-u` flag to unconfigure the

dynamically configured /dev/none device driver from the running system.

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.

3.6 Foreign Device Subset Control Program

Because a driver for a foreign device is installed during UNIX installation, it must be statically configured. In addition, its subset control program must support installation of the kit into a Remote Installation Services (RIS) area for use by network installation clients.

The RIS utility provides the ability to install kits into a RIS area for subsequent installation on a client system. When installing the kit into a RIS area, the RIS installation procedure calls the subset control program, passing EXTRACT as an argument. The RIS utility, not the setld utility, defines this phase. The subset control program must set the ACT environment variable to EXTRACT in this situation.

The RIS utility invokes the subset control program at the end of the extract procedure when installing the kit on a RIS server. During this phase, the subset control program needs to invoke the kreg utility to register the new product with the kernel so that the driver is properly installed when the RIS server builds a new install kernel. In addition, the subset control program invokes the sysconfigdb utility to modify the /etc/sysconfigtab database. The modifications to the server by kreg and sysconfigdb occur only in the RIS area, as opposed to the server's root directory.

The /etc/sysconfigtab database created during this phase of the subset control program is copied onto the client system as part of the installation. This copy replaces the /etc/sysconfigtab database installed as part of the base operating system subset load. This ensures that the proper support for the driver exists on the system.

The setld utility calls the subset control program during the C INSTALL phase when performing a network installation of RIS clients. At this time, the subset control program calls the kreg utility to register the driver with the new system. This adds the driver into the kernel that is built as part of the installation process. No modification to the /etc/sysconfigtab database is required at this point because the installation process takes care of the required modification during the EXTRACT phase.

A kit for a foreign device may be installed by osfboot or by the RIS utility during bootstrap linking of the kernel. The subset control program in

Example 3–4 supports both types of installation. The subset control program does not provide a menu of configuration options because a driver for a foreign device must be statically configured.

Example 3–4: Subset Control Program for the /dev/edgd Driver

```
#!/sbin/sh
#
#
#  EDGD.scp - Install files associated with the statically
#             configured /dev/edgd device driver product.
#
#


#
# RIS server installation of a foreign kit
#
# In the case of RIS extract, the variable ACT is NULL, and
# the first parameter passed to the subset control program
# specifies the phase.
#
[ "$ACT" ] ||  1
    ACT=$1

case "$ACT" in

#
# Configuration INSTALL phase takes place after the subsets
# are loaded.  This phase configures the device driver into
# the system.  It is invoked on all installations of the kit
# during CD-ROM or RIS-client Digital UNIX installation, and
# setld -l on an installed system.
#
C)
    case $1 in
    INSTALL) 2
        echo "INSTALL phase "

        # Register the files associated with the static
        # /dev/edgd device driver product.
        kreg -l EasyDriverInc EDGSTATIC100 /usr/opt/EDG100 3

        # Add the sysconfigtab file fragment associated with the
        # static /dev/edgd device driver product to the customer's
        # /etc/sysconfigtab database.
        sysconfigdb -a -f /usr/opt/EDG100/sysconfigtab edgd 4
        ;;
    esac
    ;;

#
# RIS server kit installation phase
#
EXTRACT)  5

    echo "EXTRACT phase "

    #
    # The RIS server does this with ROOT set to the RIS area,
    # and the RIS area must be extracted, not linked to a CD-ROM.
    #
```

Example 3–4: Subset Control Program for the /dev/edgd Driver (cont.)

```
    # Register the files associated with the static /dev/edgd
    # device driver product.
    kreg -l EasyDriverInc EDGSTATIC100 /usr/opt/EDG100 6

    # Break link between /etc/sysconfigtab and /etc/.new..sysconfigtab
    # so the subset control program can run sysconfigdb for the RIS
    # installation.
    rm /etc/sysconfigtab  7

    # Copy the /etc/sysconfigtab database to client system.
    cp /etc/.new..sysconfigtab /etc/sysconfigtab 8

    # Add the files associated with the static /dev/edgd device
    # driver product to the customer's /etc/sysconfigtab database.
    sysconfigdb -a -f /usr/opt/EDG100/sysconfigtab edgd 9
    ;;

#
#  This phase is executed on a setld -d command, which removes
#  the subset from the system.
#
POST_D)  10

    kreg -d EDGSTATIC100    11
    rm -rf /usr/opt/EDG100  12
    sysconfigdb -d  edgd    13

    echo "The /dev/edgd device driver is no longer on the system." 14
    echo "Remember to build a new kernel by running doconfig to"
    echo "remove the /dev/edgd driver functionality."
    ;;

esac
exit 0
```

1 If ACT is null, sets the environment variable to the value of the first argument. The ACT environment variable is null during the RIS extract phase, when the RIS utility calls the subset control program with an argument of EXTRACT.

2 Handles the C INSTALL phase when the device driver is configured into the system.

3 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 EDGSTATIC100. The subset name consists of the product code, subset mnemonic, and 3-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/EDG100. The kreg utility places this directory in the driver files path field of the customer's /usr/sys/conf/.product.list file.

4 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 –a flag

  This flag causes sysconfigdb to add 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/EDG100/sysconfigtab to indicate the location of the sysconfigtab file fragment for the /dev/edgd device driver.

- Device driver name

  The kit developer at EasyDriver, Inc. specifies edgd 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*.

5 During the EXTRACT phase, handles installation of the kit on the RIS server.

6 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.

7 Breaks the link to the /etc/sysconfigtab database on the client's system.

8 Copies the /etc/sysconfigtab database from the RIS area on the server to the client system.

9 Adds the sysconfigtab file fragment for the statically configured driver to the system's /etc/sysconfigtab database by calling the sysconfigdb utility with the –a and –f flags.

10 Handles the POST_D phase, when setld deletes the product subsets from the system.

11 Calls the kreg utility with the –d flag to deregister the driver with the kernel.

12 Removes the files from the product directory, EDG100, and any of its subdirectories, making the product unavailable on the system.

13 Calls the sysconfigdb utility with the –d flag to delete the device driver from the client's /etc/sysconfigtab database.

14 Displays a message on the console terminal informing the user that the device driver that controls the foreign device is no longer available on the system.

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. For example, the ODB product defines 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 system administrator can choose not to install that 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.

You specify the subsets and the files that each subset contains in a master inventory file. In a key file, you specify the product attributes, such as the product name and version, and the subset definitions. With these two files in place — the master inventory and key file — you can run the `kits` utility to create the subsets and control files in the output directory.

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

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

Figure 4–1: Grouping Files into Subsets

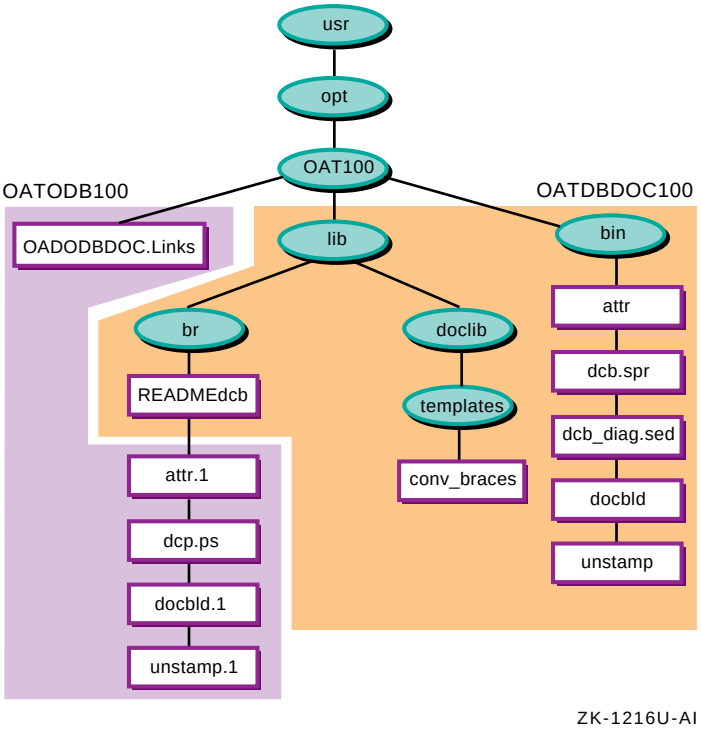
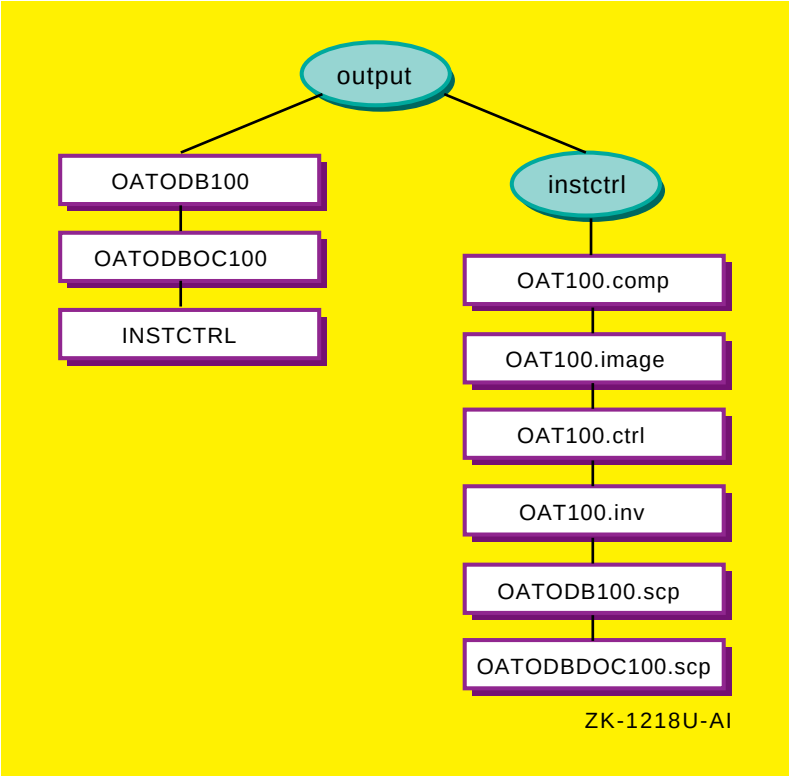


Figure 4–2: Contents of the ODB output Directory



4.1 Creating the Master Inventory File

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. For example:

```
% 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 ../src
```

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.

Caution

- Use extreme care when editing the master inventory file; fields in this file must be separated by single tab characters, not by spaces.
- 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 Master Inventory Records

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 is usually 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.
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 if the product is to be used. The OATODBDOC100 subset contains the product's documentation and is not required to make the product work.

Example 4–1: 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 setld 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.

4.2 Creating the Key File

The key file identifies the product that the kit represents, such as its name, version number, and the name of the master inventory file for the kit. You create this file in the data directory with any text editor that you like. Its 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 Authoring Tools'
CODE=OAT
VERS=100
MI=OAT100.mi
COMPRESS=1
#
#       Subset definitions
#
%%
OATODB100       .       0       'Document-Building Tools'
OATODBDOC100    .       2       'Document Tools Documentation'
```

As shown in the example, the key file is divided into two sections separated by a line that contains two percent signs (%%), as follows:

- The product attributes section 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. Table 4–2 describes the possible attribute-value pairs. 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 section describes each of the subsets in the kit and provides subset-level instructions for the `kits` command. This section contains one line for each subset in the kit. Each line consists of four fields separated by tab characters. 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` subset, which contains the documentation, is optional; its Flags field is set to 2.

Table 4–2: Key File Attributes Section

Attribute	Description
`NAME`	The product name; for example, Orpheus Authoring Tools. Enclose the product name in single quotation marks (') if it contains spaces.
`CODE`	A unique product code that consists of three numbers or uppercase letters, for example, OAT; the first character must be a letter. Note: The first three letters of the subset name must be the same as the product code. Otherwise, any shell routines that the subset control program calls to create links and `STL_LinkBack` will not work. Digital has reserved the following codes: DNP, DNU, EPI, FOR, LSP, ORT, OSF, SNA, UDT, UDW, UDX, ULC, ULT, ULX, UWS. Internal Digital product developers should contact the Software New Products Committee (SNPC) through product management to register a unique code. Third-party developers should contact their Digital representatives to register a unique product code.
`VERS`	A 3-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.
`MI`	The name of the product's master inventory file, which consists of the product name, code, and version plus the `mi` extension. You create and maintain the master inventory file with the `newinv` utility.

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

Attribute	Description
ROOT	Not illustrated in the example, Digital 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). 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 3-digit version code (for example, 100). All letters in the subset identifier must be uppercase.
Reserved	Must be a single period (.).
Flags	A 16-bit unsigned integer. Digital 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. 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-Building Tools’. Note: The percent sign character (%) is reserved in this field and must not be used for layered products.

4.3 Running the kits Utility

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

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

Note

When you run the `kits` utility, make sure you are in the data directory. If you are not in the data directory, the utility cannot find your subset control files.

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

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

The subset files and the `INSTCTRL` file are constituents of the final kit.

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 `setld` 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 `setld` control information. There is one subset control file for each subset.

Table 4–4: Installation Control Files in the instctrl Directory (cont.)

File	Description
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, kits creates a blank file.

The following sections describe the contents of the installation control files in detail.

4.3.1 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.

4.3.2 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

4.3.3 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 a descriptive product name and subset identifier.

- `DESC`

  Specifies a descriptive subset identifier.

- `ROOTSIZE`

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

- `USRSIZE`

  Specifies (in bytes) the amount of the `usr` file system the subset requires.

- `VARSIZE`

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

- `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 `setld`. 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 a dependency list (reserved — see Table 4–3).

- `FLAGS`

  Specifies subset control flag bits.

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

```
NAME='Orpheus Authoring Tools OATODBDOC100'
DESC='Document Tools Documentation'
```

```
ROOTSIZE=0
USRSIZE=522090
VARSIZE=0
NVOLS=1:2
MTLOC=1:1
DEPS="."
FLAGS=34816
```

4.3.4 Subset Inventory File

The subset inventory file describes each file in the subset, listing its size, checksum, permissions, and other information. The `kits` command 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 4–3 shows the inventory file, `OATODBDOC100.inv`, for the ODB kit's `OATODBDOC100` subset. The backslashes (\) in this example indicate line continuation and are not present in the actual file.

Example 4–3: Sample Subset Inventory File

```
4       983     01851   1065    0       100644  3/21/91 100     f\
       ./usr/opt/OAT100/lib/br/attr.1  none    OATODBDOC100
4       424997  63356   1065    10      100644  4/15/91 100     f\
       ./usr/opt/OAT100/lib/br/dcb.ps  none    OATODBDOC100
4       7283    03448   1065    10      100644  4/15/91 100     f\
       ./usr/opt/OAT100/lib/br/docbld.1        none    OATODBDOC100
4       6911    37501   1065    0       100644  3/21/91 100     f\
       ./usr/opt/OAT100/lib/br/docbld.5        none    OATODBDOC100
4       985     41926   1065    0       100644  3/21/91 100     f\
       ./usr/opt/OAT100/lib/br/unstamp.1       none    OATODBDOC100
```

Each record of the inventory is composed of 12 fields separated by 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 is usually set for files such as `usr/spool/mqueue/syslog`.

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

Field	Name	Description
		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.
6	Mode	The 6-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

Producing Distribution Media

After you have gathered product files into subsets, you can move the subsets 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 target system, preserving the files' original directory structure. Kits for user and kernel products should be in `tar` format.

- 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. Foreign device kits must be in DCD format.

You can distribute 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 foreign device kits on magnetic tape because this media does not support DCD format. Use the `gentapes` utility to produce kits for magnetic tape media.

- Diskette

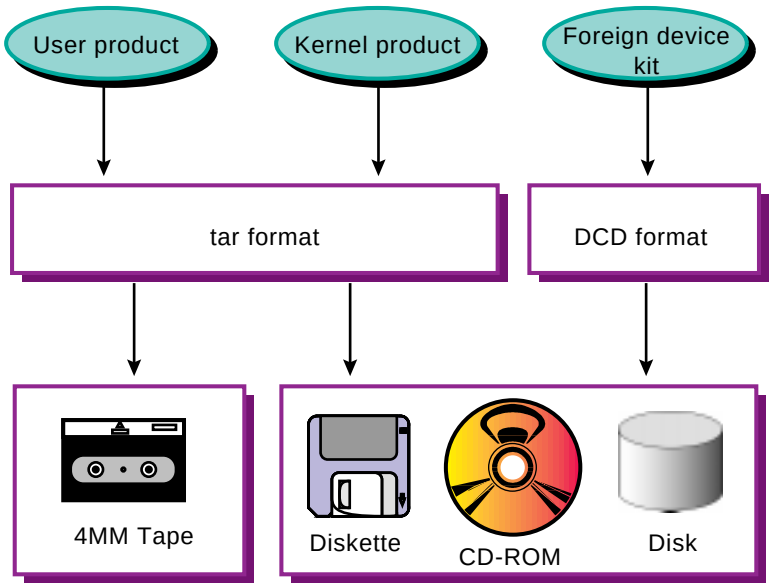
  Diskettes are a good media for testing purposes or for small products, such as device drivers. However, the product must fit on a single diskette; it cannot span multiple diskettes. Use the `gendisk` utility to produce kits for diskette media.

- CD–ROM

  CD–ROM media can support large kits or multiple kits on a single media. However, you must dedicate an entire hard disk on the kit development system to creating a kit master. 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 5–1 shows the types of file formats and distribution media that are available for layered product kits.

Figure 5–1: File Formats for Layered Product Kits



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.

5.1 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 going to produce tape or disk media. Therefore, you must add one record for each media type on which you plan to distribute your kit.

5.1.1 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.
- A code that indicates the media type, either TK for TK50 tapes or MT for 9-track magnetic tapes.
- Product description. This entry is usually 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.

For example, the following record would be added to the /etc/kitcap file to produce the ODB kit on TK50 tapes:

```
OAT100TK | Orpheus Authoring Tools: \
   /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 Authoring Tools) 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.

5.1.2 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.
- 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.

The following example shows the kitcap record for the /dev/none driver:

```
ESA100HD:c:/: \
  EasyDriver_none_driver: \
  /easy/output:instctrl:ESANONESTATIC100
```

Based on the information supplied in this record, 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.

5.2 Building a Kit on Magnetic Tape Media in tar Format

With the product subsets 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 /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*, gentapes looks on your

own system. You can use 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/nrmt0h.

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

```
% gentapes OAT100 /dev/nrmt0h
```

5.3 Building a Kit on Disk Media

With the product subsets 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.

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/rrz1a.

You can use gendisk to produce kits in either tar or DCD format, depending on whether you use the –d option.

5.3.1 Preparing a 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
```

5.3.2 Preparing a Foreign Device Kit in DCD Format

To prepare a foreign device kit, you run gendisk with the –d option. The utility creates a kit in DCD format on diskette or hard disk, as specified in the /etc/kitcap entry. In addition to running gendisk, you need to include the following files to support foreign device installation and RIS:

- The *name*.kit file that the osfboot utility uses to interpret the kit.
- Links for modules that the osfboot utility's bootstrap link support references.
- The /etc/sysconfigtab database that contains configuration information for the devices on the kit. The osfboot utility reads this information and passes it to the bootstrap-linked /vmunix image when it starts up.
- A *kitname*.kk file in the instctrl directory of the kit. The RIS utility uses this file when installing the kit into a RIS area.

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 partition for creating a media master area, you must reference your kit to the mount point of the device.

To create a kit on diskette for the /dev/edgd product, you would perform the following steps:

1. Place the diskette in the drive and format it, as follows:

```
# fddisk -fmt /dev/rfd0c
Disk type: 3.50 inch, HD (1.44MB)
Number of sectors pertrack: 18
Number of surfaces:   2
Number of cylinders: 80
Sector size:  512
Interleave factor:  2:4
Formatting disk...
  Percentage complete: Format complete, checking...
Quick check of disk passes OK.
```

2. Write a disk label to the diskette:

```
# disklabel -wr fd0 rx23
```

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

Warning

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

```
# gendisk -d EDG100 /dev/rfd0c
Generating EDG100 Kit from visier on /dev/rfd0c

WARNING:  this will remove any information stored in
/dev/rfd0c
Are you sure you want to do this? (y/n): y
Do you want to clean the entire disk first? (y/n): n
Preparing /dev/rfd0c (floppy)
done.

Checking /dev/rfd0c
/sbin/usf_fdck /dev/rfd0c
** /dev/rfd0c
File system unmounted cleanly - no fsck needed

Mounting /dev/rfd0c on /usr/tmp/cd_mdt8344

Writing Images (dd=/).

 Image instctrl...done.
 Image EDGSTATIC100...done.

Verifying Images (dd=/).
```

```
 Image instctrl...done.
 Image EDGSTATIC100...done.

Kit EDG100 done.

Cleaning up working directories.
Unmounting /dev/rfd0c
```

4. Mount the diskette in preparation for making foreign device kit modifications:

   ```
   # mount /dev/fd0c /mnt
   # cd /mnt
   ```

5. Create the *kitname*.kk file:

   ```
   # touch ./instctrl/EDGSTATIC100.kk
   ```

6. Create the /sys/BINARY and /etc directories on the diskette:

   ```
   # mkdir -p ./sys/BINARY
   # mkdir ./etc
   ```

7. Copy the *name*.kit file from the kit-building area to the diskette:

   ```
   # cp /kit_area/edgd.kit ./
   ```

8. Create a link from the driver's module file in the kit area to the /sys/BINARY directory:

   ```
   # cd ./sys/BINARY
   # ln -s ../../usr/opt/EDG100/edgd.mod ./edgd.mod
   ```

9. Create a link from the sysconfigtab database file in the product area to /etc/sysconfigtab:

   ```
   # cd ../../etc
   # ln -s ../usr/opt/EDG100/sysconfigtab ./sysconfigtab
   # cd /
   ```

10. Move the instctrl directory to an empty directory. (The gendisk utility places the instctrl directory in the top-level directory, by default. To support RIS installation, you must move the instctrl directory to an empty directory. Otherwise, the RIS utility cannot recognize the kit.) For example:

    ```
    # mkdir -p /mnt/ALPHA/EDGSTATIC100
    # mv /mnt/instctrl /mnt/ALPHA/EDGSTATIC100
    ```

11. Unmount the diskette:

    ```
    # umount /mnt
    ```

To create the /dev/edgd kit on CD–ROM media, perform steps 3 through 11 and use a hard disk as the target instead of a diskette. The hard disk

serves as the master for the kit. You can then burn the kit onto the CD–ROM, following the instructions that come with your CD–ROM device.

6

Testing the Installation of a Kit

Digital UNIX provides several options for installing layered product kits:

- The `setld` utility can install a kit either during system installation or after the system is running.
- The `osfboot` utility installs foreign device kits before the initial bootstrap of a system. 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 foreign device kit into a RIS environment. Client systems can then install the kit from the RIS area by calling the `setld` utility.

Before shipping a product 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.

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

6.1 Installing a User Product

To install a user product, log onto the system as superuser or `root` and run the `setld` utility. For example, the ODB product could be installed 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
   ```

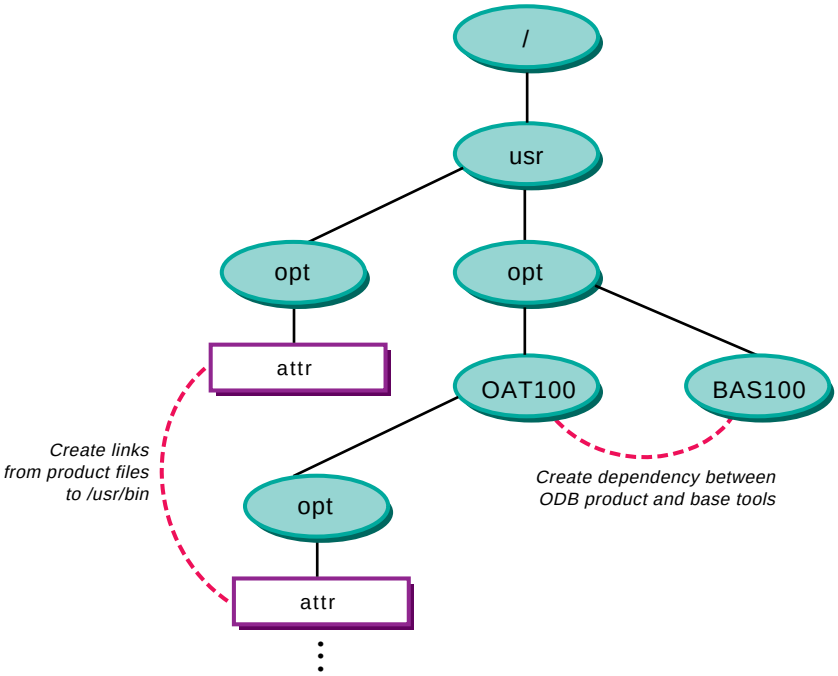
After mounting the CD–ROM, you can change to the `/cdrom` directory and view the directories on the CD–ROM.

4. Install the user product subsets:

   # **`setld -l /cdrom/ALPHA/OAT100`**

   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, `setld` calls the subset control program for each subset. Figure 6–1 shows the links and dependencies that the ODB subset control program creates.

Figure 6–1: Defining Links and Dependicies 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.

6.2 Installing a Kernel Product

To install a kernel product, log onto 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 `/dev/none` 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/ALPHA/ESA100**

   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 6–2 shows the steps the subset control program takes to statically configure the driver; Figure 6–3 shows the steps the subset control program takes to dynamically configure the driver.

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

   # **umount /cdrom**

6. Restart the system with the new kernel:

   # **/usr/sbin/shutdown –r now**

   When the system starts up, the `/dev/none` driver is available on the system.

Figure 6–2: Statically Configuring a Driver

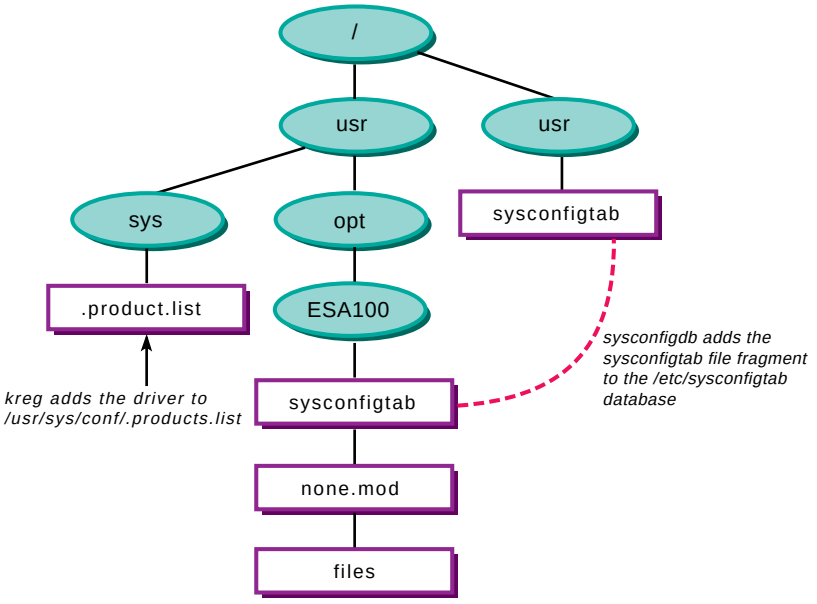
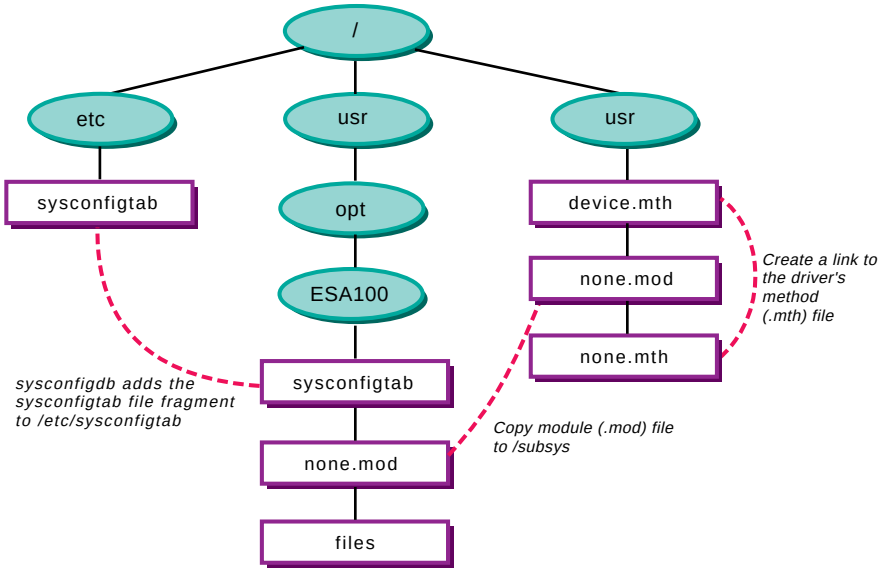


Figure 6–3: Dynamically Configuring a Driver



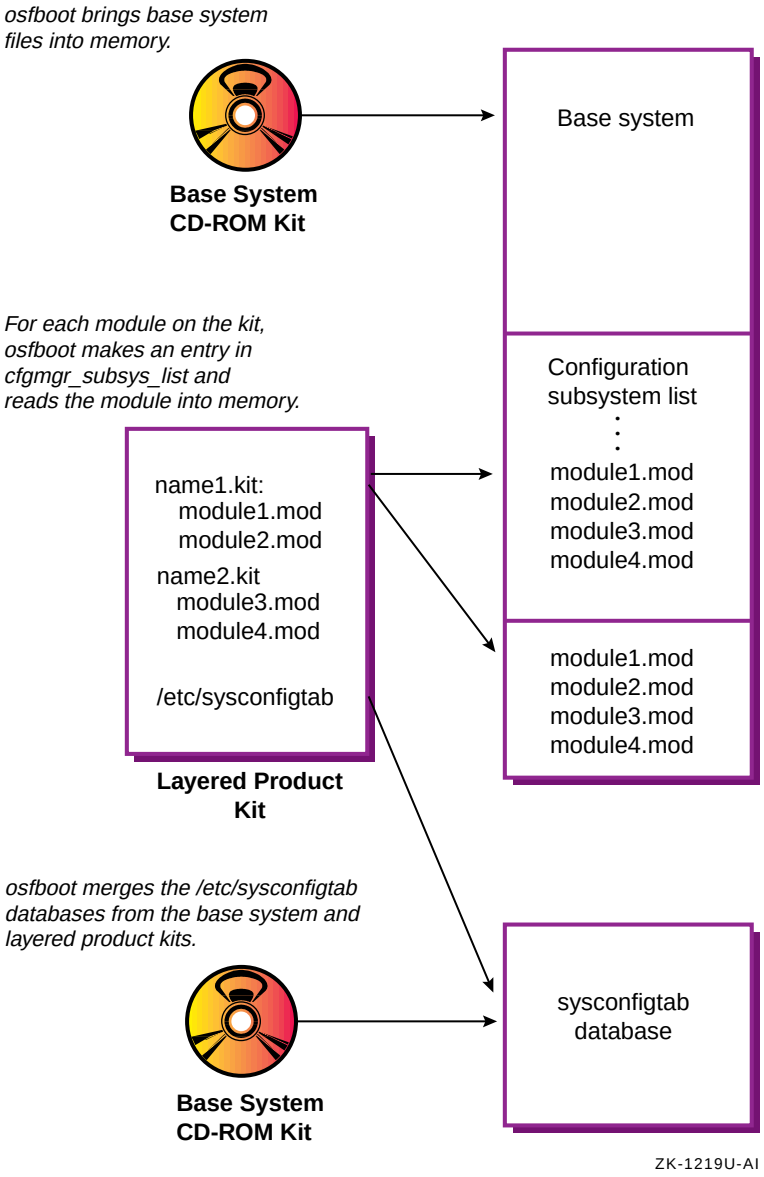
See the *Installation Guide* for more information on using the `setld` utility to install layered products. See `doconfig`(8) for more information on the `doconfig` utility.

6.3 Installing a Foreign Device Kit

You install a foreign device kit with the `osfboot` utility, which runs from the console prompt. To follow this procedure, you need two kits: the CD–ROM that contains the Digital UNIX base system and the diskette or CD–ROM that contains the foreign device kit. You specify the kit for the foreign device during the initial installation and bootstrap of the system. The `osfboot` utility runs the device driver from the foreign device kit during installation, then builds it into the kernel during the bootstrap operation.

Figure 6–4 shows the steps that `osfboot` takes to build a kernel that includes the foreign device.

Figure 6–4: Bootstrap Linking with a Foreign Device Kit



The following steps show how to install the `/dev/edgd` driver as a foreign device:

1. Insert the Digital UNIX kit in the CD–ROM drive.
2. From the console prompt, invoke the `osfboot` utility with the `–fl` option and the `fg` flags, as follows:

```
>>> boot –fl "fg" dka400
```

The utility reads bootstrap code from the CD–ROM.

3. The osfboot utility prompts you for the name of the device from which to install the foreign device kit. Enter the appropriate console firmware device name, for example (for a diskette device):

   ```
   Enter Device Name: dva0
   ```

   For a list of device names, enter the following command:

   ```
   >>> sho dev
   ```

4. The osfboot utility prompts you for the name of the foreign device kit. With the information you supply here, the utility builds a list of the modules that will go into the kernel. To install the /dev/edgd driver, enter the following:

   ```
   Enter Kit Name: edgd.kit
   ```

5. Insert the media and press the return key:

   ```
   Insert media for kit 'dva0:edgd.kit', press Return
   when ready: Return
   ```

6. You may want to install more than one foreign device kit. The osfboot utility prompts you to enter more kit names. These names are also added to the list of modules that will go into building the kernel. Each time you enter a kit name, you must also insert the media that contains the kit. Press the return key (without entering a kit name) in response to this prompt to indicate that there are no more kits:

   ```
   Enter Kit Name: Return
   ```

7. When all the kit names have been entered, osfboot prompts you to reinsert the media that contains the Digital UNIX base system. This is to ensure that you have not replaced the base system kit with a foreign device kit during the previous steps:

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

8. At this point, osfboot prompts you to reenter the kits that you want included in the kernel. This time, osfboot reads the modules from the kits and links them into the kernel.

9. When the link operation has completed, osfboot prompts you to reboot the system. As the system boots up, you must reinsert the foreign device kit one more time to load the subsets and install them on the system.

   ```
   Insert media for kit 'dva0:edgd.kit', press Return
   when ready: Return
   ```

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

This step repeats once for each kit you named. Later in the installation procedure, the `setld` utility installs the kits and builds a kernel configuration that includes them.

6.4 Installing a User or Kernel Product into a RIS Area

You can use the RIS utility to install a kernel product kit onto a RIS server for use 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. From the RIS Utility Main Menu, choose `INSTALL software products` 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
   ```

3. The RIS Software Installation Menu appears. At the prompt, 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:
   ```

   See *Sharing Software on a Local Area Network* for more information on installing software in the RIS area.

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
```

   Enter the client information requested by the prompts, as described in *Sharing Software on a Local Area Network*.

2. As superuser or `root` on the client system, install the product subsets from the RIS area. For example, if the RIS area were on node `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.

6.5 Installing a Foreign Device Kit into a RIS Area

To install a foreign device kit into a RIS area, you must first install Digital UNIX Version 4.0 into an extracted RIS area.

1. On the RIS server, 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
```

2. From the RIS Software Installation Menu, choose `Add software into an existing area`. The RIS utility displays the name of the existing environment. In this example, the name is `/usr/var/adm/ris/ris0.alpha`.

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

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

3. The RIS utility prompts you to enter the location of the software that you want to install. In this example, the name is `/floppy/ALPHA/EDGSTATIC100`. The path you enter is the name of the top-level directory on the distribution media for the product kit.

```
Enter the device special file name or the path of the
directory where the software is located (for example,
/mnt/ALPHA/BASE): /floppy/ALPHA/EDGSTATIC100
```

4. The RIS utility displays a list of the base products that are installed in the RIS area. You must choose one; this is the base product to which the foreign device kit will be added. In this example, there is only one base product to choose from.

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

     1   'Digital UNIX V4.0 Operating System ( Rev 375 )'

Enter your selection or (return) to quit : 1
```

5. The RIS utility loads the product from the distribution media into the RIS area in preparation for the installation. No user interaction is required at this time.

```
Preparing new product area...
        Working....Mon Apr 29 15:06:33 EDT 1996
        Working....Mon Apr 29 15:08:34 EDT 1996
        Working....Mon Apr 29 15:10:35 EDT 1996
          .
          .
          .
```

6. When all the subsets have been loaded, the RIS utility lets you choose which subsets you want to install. In this example, the /dev/edgd 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) EDGD kit subset

 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
```

7. The RIS utility gives you the chance to confirm your choice.

```
You are installing the following optional subsets:

        EDGD kit subset

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

8. The RIS utility extracts the foreign device kit into the RIS area. No user interaction is needed at this time.

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

File system space checked OK.
Extracting EDGSTATIC100...
Media extraction complete.
.
.
.
EXTRACT phase for EDGD kit installation into RIS area
```

9. The foreign device has been extracted into the RIS area and installed into a new version of the base system. If you choose SHOW software products in remote installation environments from the RIS Utility Main Menu, you can see that there are now two base products — Digital UNIX Version 4.0 without the /dev/edgd foreign device kit and Digital UNIX Version 4.0 with the /dev/edgd foreign device kit.

```
*** 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: s

1  /usr/var/adm/ris/ris0.alpha
      'Digital UNIX V4.0 Operating System ( Rev 375 )'
      'Digital UNIX V4.0 Operating System ( Rev 375 )' w/
'EDGSTATIC software version 1'
```

The foreign device subsets are now available in the RIS area. However, before a client can perform an installation from this RIS area, you must reigister the client, as follows:

1. On the RIS server, 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
```

2. 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.

```
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
```

3. The RIS utility prompts you for the client processor's host name. In this example, the name is `xnite`.

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

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

4. The RIS utility prompts you to choose the products for the client to install. In this example, the user chooses the base system that includes the `/dev/edgd` foreign device.

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

Product    Description
   1    'Digital UNIX V4.0 Operating System ( Rev 375 )'
   2    'Digital UNIX V4.0 Operating System ( Rev 375 )' w/
'EDGSTATIC software version 1'

The following products:  1 2, are Operating System Base
products. Please select only one of these products when
making your selections.

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

5. The RIS utility displays the product you have chosen and gives you the opportunity to confirm your choice.

```
You chose the following products:

   2    'Digital UNIX V4.0 Operating System ( Rev 375 )' w/
'EDGSTATIC software version 1'

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

6. The RIS utility prompts you to 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: 08-00-2b-e2-3a-43
```

7. If this is the first client added for this product, RIS builds the kernel at this time. No user interaction is required.

```
A new generic install kernel for the client systems will
now be built which includes support for the 3rd party
device(s).

*** PERFORMING KERNEL BUILD ***
        Working....Mon Apr 29 14:37:39 EST 1996
        Working....Mon Apr 29 14:39:40 EST 1996
         .
         .
```

8. Preparation of the kit in the RIS area 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 foreign device subsets. For example:

```
>>> boot ewa0
```

The bootup procedure installs the kernel from the RIS area, then performs a normal installation — loading the subsets that make up the system and including the subsets from the foreign device kit.

A

Digital UNIX Standard Directory Structure

Digital recommends that you install products in the `/opt`, `/usr/opt`, and `/var/opt` directories. As part of installation, the subset control program that you provide creates links from these directories to directories that would typically be in the users' search paths. Most Digital UNIX 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 A–1 and Figure A–2 show the directories in the Digital UNIX standard directory structure. These are the directories that you should use to ensure that your product is portable to other systems. (Some of the illustrated directories are actually symbolic links.)

Figure A–1: Base System Directory Structure

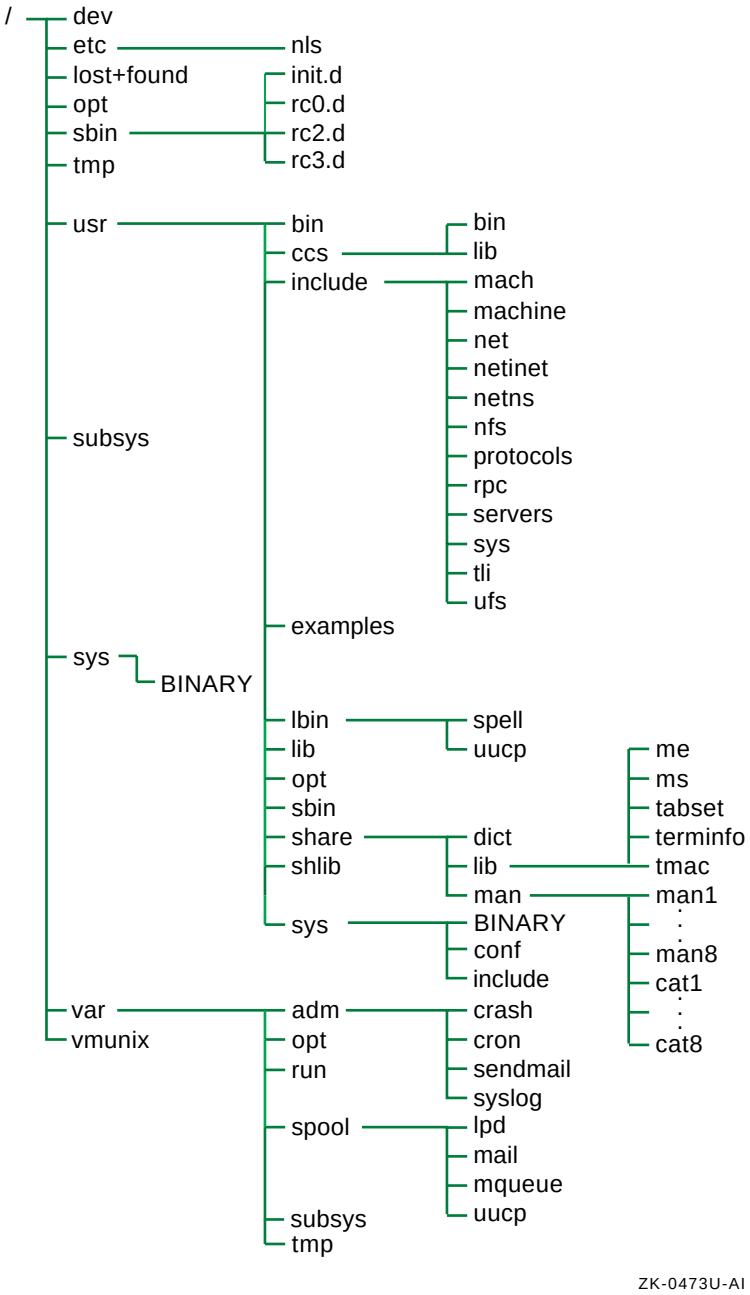


Table A–1 describes the contents and purpose of the directories shown in Figure A–1.

Table A–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 A–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 A–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
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 A–2: X Directory Structure

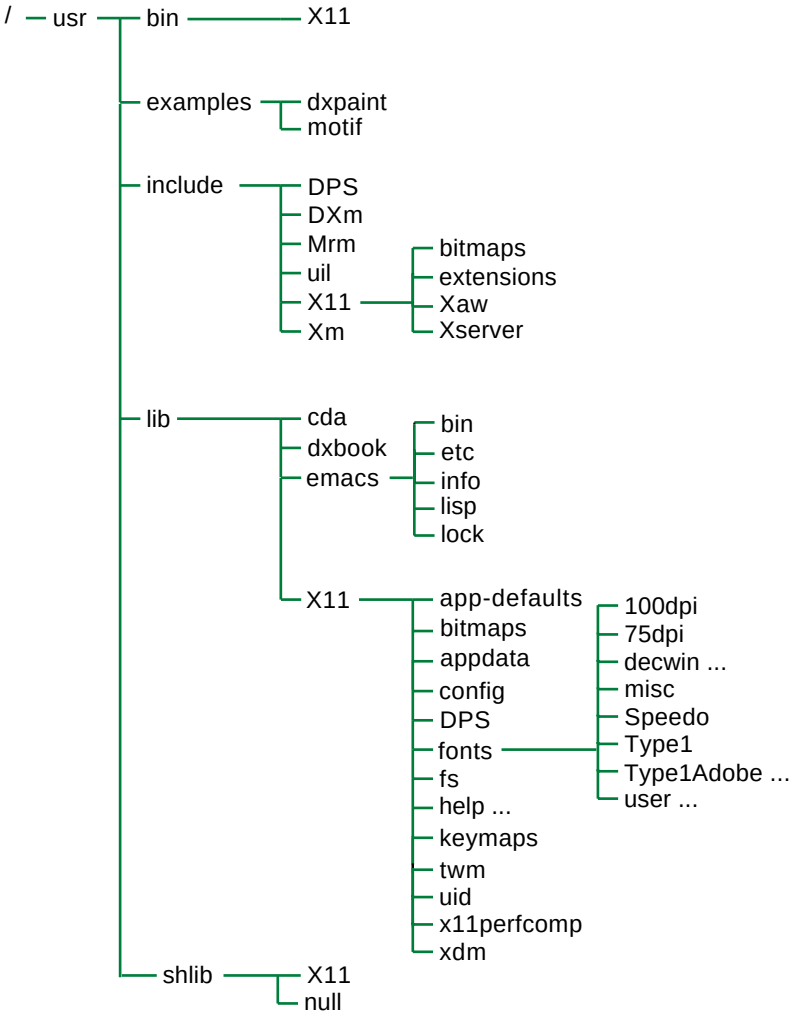


Table A–2 describes the contents and purpose of the directories shown in Figure A–2.

Table A–2: Contents and Purpose of X Directories

Directory	Description
`/usr/`	Most user utilities and applications
`bin/`	Common utilities and applications
`X11/`	X applications

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

Directory	Description
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
fonts/	Font files
100dpi/	100 dpi fonts from X Consortium
75dpi/	75 dpi fonts from X Consortium
decwin/	DECwindows fonts
100dpi/	100 dpi fonts

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

Directory	Description
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 manual.

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.

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.

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.

data hierarchy

In the kit-building directory structure, the data heirarchy 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.

/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**.

foreign device kit

A foreign device kit contains a kernel product that must be installed during the initial installation and bootstrap linking of the Digital UNIX system. See also **kernel product** and **layered product**.

forward link

A foward 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.

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**.

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 a Digital UNIX 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**.

layered product

A layered product is an optional software product designed to be installed as an added feature of the Digital UNIX system. See also **foreign device 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.

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**.

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**.

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.

osfboot utility

The `osfboot` utility performs the initial installation and bootstrap of a Digital UNIX system.

RIS

See **Remote Installation Services**.

Remote Installation Services

Remote Installation Services (RIS) is a utility that lets users install kits into a RIS area for subsequent installation over a network onto client systems. Using a RIS server makes installation of layered products faster and easier for all the clients on the network.

SCP

See **subset control program**.

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

A subset control program (SCP) is 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.

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**.

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**.

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825) before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-bps modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	—	Local Digital subsidiary or approved distributor
Internal (submit an Internal Software Order Form, EN-01740-07)	—	SSB Order Processing – NQO/V19 *or* U.S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063-1260

Reader's Comments

Digital UNIX
Guide to Preparing Product Kits
AA-QYW7B-TE

Digital 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) 881-0120, Attn: UEG Publications, ZK03-3/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

digital™

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

DIGITAL EQUIPMENT CORPORATION
UEG PUBLICATIONS MANAGER
ZKO3−3/Y32
110 SPIT BROOK RD
NASHUA NH 03062−9987

Do Not Cut or Tear − Fold Here

Cut on
Dotted
Line