Tru64 UNIX

Guide to Preparing Product Kits

Part Number: AA-RH9SA-TE

July 1999

Product Version: Tru64 UNIX Version 5.0 or higher

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

Compaq Computer Corporation
Houston, Texas

Contents

About This Manual

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

Audience

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

New and Changed Features

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

- Expanding and clarifying the kit definition at the beginning of Chapter 1.
- Adding notes to Section 1.3 and Section 3.4.3 to explain subset control program requirements for DMS compliance.
- Reorganizing the chapters for creating, testing, and delivering product kits:
  - Chapter 3 consolidates the procedures to create subsets.
  - Chapter 4 tells you how to produce user product kits.
  - Chapter 5 tells you how to produce kernel product kits.
- Changing the definition of the `DEPS` subset control file field in Section 3.5.3.
- Providing instructions on how to create and build a firmware consolidated CD-ROM in Appendix A. A firmware consolidated CD-ROM lets you upgrade your processor firmware at the same time that you install the operating system.

Organization

This manual is organized as follows:

Chapter 1	Product Kits Overview Presents an introduction to the kit-building process.
Chapter 2	Creating the Kit Directory Structure Describes how to create and populate kit directories.
Chapter 3	Creating Subsets Describes how to organize product files into subsets, create kit production files, prepare subset control programs, and run the `kits` utility to produce the subsets and related control files.
Chapter 4	Creating User Product Kits Describes how to create, test, and deliver user product kits.
Chapter 5	Creating Kernel Product Kits Describes how to create, test, and deliver kernel product kits.
Appendix A	Creating a Firmware Consolidated CD-ROM Describes how to create a CD-ROM that lets you upgrade your processor firmware at the same time that you install the operating system.
Glossary	Defines terms used in this manual.

Related Documents

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

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

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

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

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

- *Sharing Software on a Local Area Network*

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

- *Writing Device Drivers*

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

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

- *Installation Guide*

  This manual describes the procedures to perform an update installation or a basic installation of the operating system on all supported processors and single-board computers. It explains how to prepare your system for installation, boot the processor, and perform the installation procedure.

- *Installation Guide — Advanced Topics*

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

- *System Administration*

  This manual describes how to configure, use, and maintain the operating system. It includes information on general day-to-day activities and tasks, changing your system configuration, and locating and eliminating sources of trouble. This manual is intended for the system administrators responsible for managing the operating system. It assumes a knowledge of operating system concepts, commands, and configurations.

- *Reference Pages Sections 8 and 1m*

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

- *Release Notes*

  The *Release Notes* describe known problems you might encounter when working with the operating system and provides possible solutions for those problems. The printed format also contains information about new and changed features of the operating system, as well as plans to retire obsolete features of the operating system. Obsolete features are features that have been replaced by new technology or otherwise outdated and are no longer needed. The *Release Notes* are intended for anyone installing the operating system or for anyone using the operating system after it is installed.

Reader's Comments

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

You can send your comments in the following ways:

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

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

  ```
  /usr/doc/readers_comment.txt
  ```

- Mail:

  Compaq Computer Corporation
  UBPG Publications Manager
  ZKO3-3/Y32
  110 Spit Brook Road
  Nashua, NH 03062-9987

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

Please include the following information along with your comments:

- The full title of the book and the order number. (The order number is printed on the title page of this book and on its back cover.)
- The section numbers and page numbers of the information on which you are commenting.
- The version of Tru64 UNIX that you are using.
- If known, the type of processor that is running the Tru64 UNIX software.

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

Conventions

The following conventions are used in this manual:

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

1 Product Kits Overview

This guide is intended to provide product and kit developers with the proper method to create a **product kit**. The following topics are discussed in this chapter:

- Defining the different product types and describing the sample product used in this guide
- Describing the available formats for layered product kits
- Illustrating the kit-building process

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

Note

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

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

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

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

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

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

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

Table 1–1: Using the *Guide to Preparing Product Kits*

ALL product kits	
1. Product Kit Overview	
2. Creating Kit Directories	
3. Creating Subsets	
ONLY user product kits	**ONLY kernel product kits**
4. Producing User Product Kits	5. Producing Kernel Product Kits

This manual uses the fictitious Orpheus Document Builder (ODB) product to demonstrate how to build kits for each product type. The same product is used for each type of product kit, but Chapter 4 and Chapter 5 describe the files specific to user and kernel product kits.

1.1 Product Types

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

- User product

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

- Kernel product

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

1.2 Kit Formats

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

- `tar` format

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

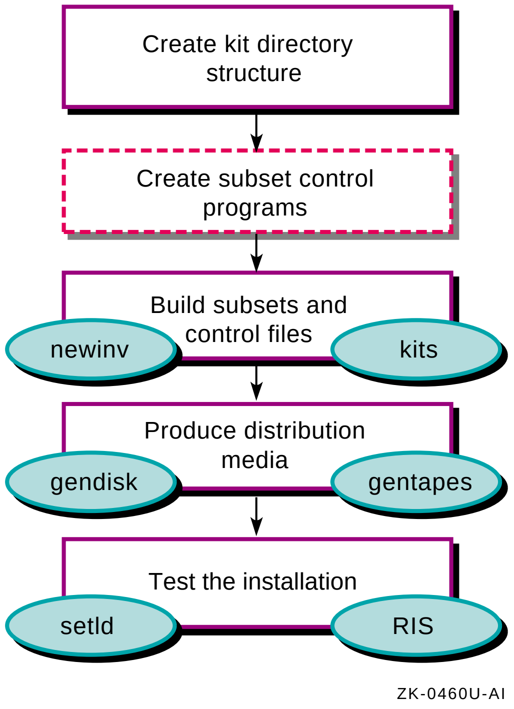
- Direct CD–ROM (DCD) format

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

1.3 Kit-Building Process

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

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


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

The kit-building process consists of the following steps:

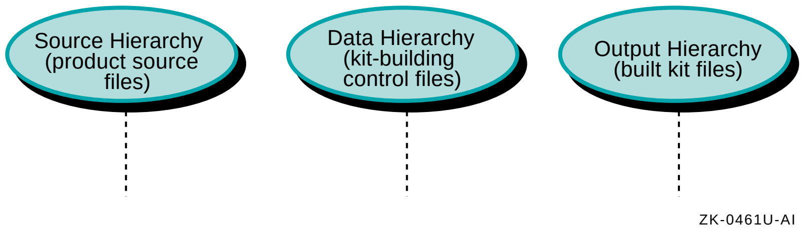
1. Create the kit directory structure that contains the source files.

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

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

   Figure 1–2 illustrates these directory hierarchies.

Figure 1–2: Kit Directory Hierarchies


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

This directory structure is the same for user products and kernel products. Only the contents of these directories differ among the product types. For example, a kernel product kit needs additional files that are unique to this specific kit type. Refer to Chapter 4 and Chapter 5 for additional information about the requirements for each product kit type.

2. Create kit production files.

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

3. Create subset control programs (if needed).

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

4. Build subsets and control files.

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

5. Test subsets.

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

6. Produce distribution media.

   When you have created the subsets for the product, you are ready to package the kit. At this point, you must decide whether to create the kit in DCD format or in `tar` format. To do this, use the `gendisk` or `gentapes` utility.

7. Test product kit media.

   After you have successfully created the kit, you should test its installation from the new media. Chapter 4 and Chapter 5 tell you how to test the installation of each of the product kit types.

2 Creating Kit Directories

After a product is developed, you are provided with the product files to package and process into a kit. Your first task is to organize these files by function and use, and place them in a kit-building directory structure. When you design the kit-building directory structure, first consider where you want to place the product files on the customer's system, then create a kit directory structure on the development system that closely mirrors that on the customer's system.

This chapter discusses the following topics:

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

2.1 Obtaining a Unique Product Code

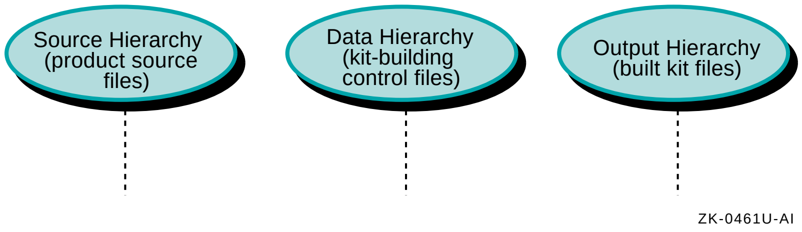
Before you can create a product kit, you must have a unique three-letter **product code**. To obtain this product code, send electronic mail to `Product@DSSR.enet.dec.com`. You use this product code and a product version number that you assign to name your product-specific subdirectories.

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

2.2 Creating a Kit Building Directory Structure

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

Figure 2–1: Kit Directory Hierarchies


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

The following describes each directory hierarchy:

- Source hierarchy

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

- Data hierarchy

  The **data hierarchy** is a directory structure that contains the following files to specify the contents of the kit and how it is organized:

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

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

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

  The `kits` utility is run from the directory in which these files are located. There is no specific requirement for the location of the data hierarchy, but it is good practice to place it under the same directory as the source hierarchy. Additional files may be required, depending on the kit type.

- Output hierarchy

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

To create the kit-building directory structure, issue the `mkdir` commands for each of the directories that you need, as in the following example:

```
mkdir /mykit
mkdir /mykit/src
mkdir /mykit/data
mkdir /mykit/output
```

Refer to the `mkdir`(1) reference page for additional information.

2.3 Populating the Source Directory

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

- Using standard directory structures
- Placing files in the kit source directory
- Setting up the sample product kit source directory

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

2.3.1 Using Standard Directory Structures

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

- Boot files reside in `/opt`

  Install files that are required at bootstrap time, such as device drivers, in a product-specific subdirectory of the `/opt` directory. This also includes any files to be accessed before non-root (`/`) file systems are mounted.

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

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

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

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

`OAT` is the code assigned to *Orpheus Authoring Tools, Inc.*, the fictitious company who developed the Orpheus Document Builder (ODB) product, and `100` is the product version number.

Using a standard directory structure has the following advantages:

- Kit components are easier to locate.

  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.

- Multiple versions of the same product can be served to different clients.

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

- Name space conflicts are avoided.

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

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

- You do not need any additional installation files for a user product.
- Section 5.1 describes the additional installation files you need for a kernel product.

2.3.2 Placing Files in the Kit Source Directory

Where you put files in the kit source directory hierarchy depends on where they should be located when the kit is installed. Table 2–1 shows where to put files that will be installed under the `root` (`/`), `/usr`, and `/var` directories.

Table 2–1: File Locations in Kit Directories

Installed Location	Kit Location
/ (root)	/opt/*PROD_CODE*
/usr	/usr/opt/*PROD_CODE*
/var	/usr/var/opt/*PROD_CODE*

For example, if you want a file to be located in the /sys/BINARY directory, place it in the /opt/*PROD_CODE*/sys/BINARY kit source directory.

You should not ship any kit files outside of the /opt, /usr/opt, and /usr/var/opt directories as it could cause a name space conflict with the base operating system or another layered product. Overwriting base operating system files can cause the following problems:

- Your product can be corrupted during an update installation of the operating system. The update installation will overwrite any file that was shipped as part of the old version of the operating system with the version of the file shipped with the operating system.
- An update installation may not complete successfully if you overwrite a base operating system file. This can make the system unusable.
- You product may have to be removed from the system to complete an update installation. Your product would have to be reinstalled after the update installation is completed.

2.3.3 Setting Up the Sample Product Kit Source Directory

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

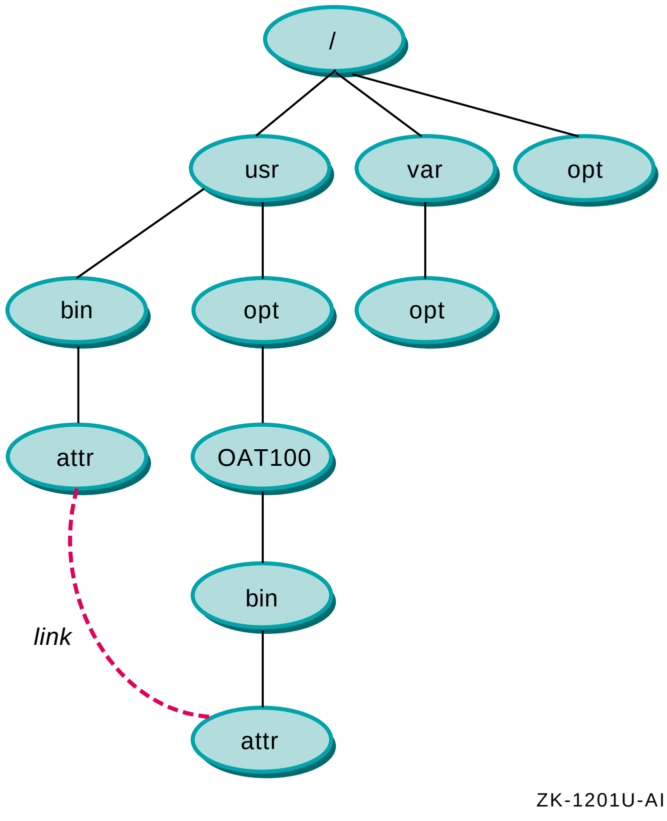
Caution

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

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

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

Figure 2–2: Sample Source Directory



The following files have been provided by the ODB developers for the OAT kit. Each of these files has a path and an associated description, and must be placed correctly in the source hierarchy for the OAT product to build successfully. The file paths in the source directory are different from the paths provided by the kit developers to avoid name space conflicts with other products.

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

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

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

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

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

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

4. `/var/log_files/odb_log` — the file listing all of the ODB documents created by a user. The `odb_log` file can be modified by users.

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

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

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

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

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

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

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

- The symbolic link does not conflict with any base operating system file. Using our example, it means that you could create the `/usr/bin/odb_start` link only if the operating system does not already contain a `/usr/bin/odb_start` file. If the operating system did contain a `/usr/bin/odb_start` file, shipping the symbolic link would overwrite an existing file on the customer's operating system.
- The command name does not conflict with any standard operating system command. For example, the `/usr/opt/OAT100/bin/odb_start` command is shipped in the ODB kit and as part of the standard operating system in `/bin/odb_start`. When the user enters the `odb_start` command, there would be a command name conflict. Depending upon whether `/bin` or `/usr/bin` is first in the search path, the user could be accessing the operating system version or the symbolically linked ODB product version.

3
Creating Subsets

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

A kit developer must perform the following tasks to build subsets and associated control files:

1. Organize product files into subsets.
2. Create a master inventory file containing information about each file in the subset.
3. Create a key file to define product attributes such as the product name, product version, and subset definitions.
4. Optionally, create subset control programs (SCPs).
5. Use the `kits` utility to produce the subsets and related control files.
6. Test your subsets to ensure that they can be loaded onto a running system, that the product runs on the system, and that the subsets can be deleted.

This chapter also discusses updating the master inventory file after subset creation.

3.1 Grouping Files into Subsets

Files that are required for the product to work should be grouped together by function. For example, if the product has two parts such as a user interface and underlying functional code, you should group them into two subsets.

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

The fictitious ODB user product requires two subsets:

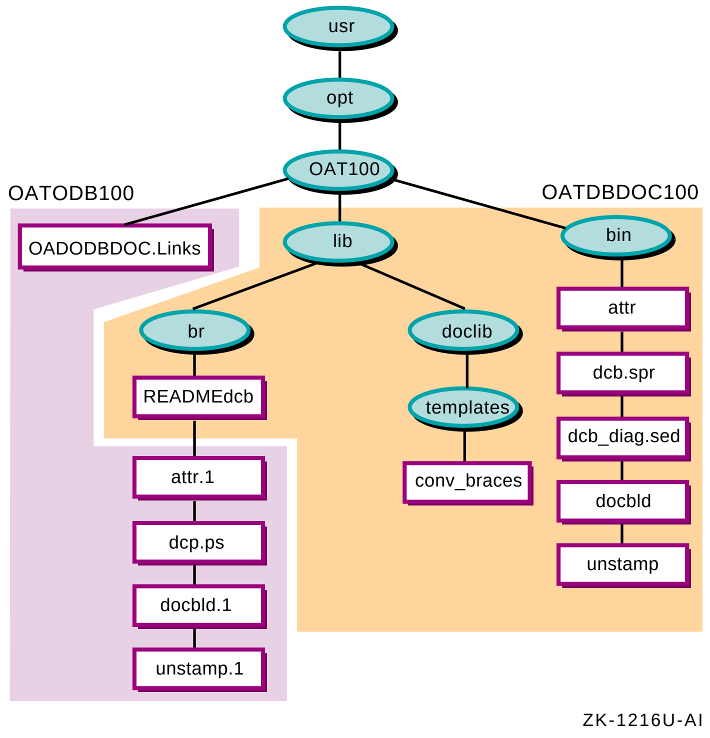
- `OATODB100`, a mandatory subset, contains the files needed to run the product. This includes all product files except the `odb_template` file.

- OATODBTEMPS100, an optional subset, contains documentation templates in the odb_template file.

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

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

Figure 3–1: ODB Kit Subsets and Files



3.2 Creating the Master Inventory File

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

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

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

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

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

Sorting inventories...done.

Joining...done.

Awking...done.

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

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

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

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

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

The `newinv` utility produces a list of files that are present in the source hierarchy and places you in the `vi` editor (or the editor specified by your `EDITOR` environment variable) to make the required changes.

Caution

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

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

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

Caution

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

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

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

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

Table 3–1: Master Inventory File

Field	Description
Flags	16-bit unsigned integer Bit 1 is the `v` (volatility) bit. When set, changes to the existing copy of the file can occur during kit installation. It usually is set for log 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, or 4.
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.

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

```
Merging...done.
```

```
Sorting...done.

Master inventory update complete.
```

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

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

Example 3–1: Sample ODB Kit Master Inventory File

```
0 ./opt RESERVED 1
0 ./opt/OAT100 OATODB100
0 ./opt/OAT100/README.odb OATODB100
0 ./opt/OAT100/sbin OATODB100
0 ./opt/OAT100/sbin/odb_recover OATODB100
0 ./usr RESERVED 1
0 ./usr/opt RESERVED 1
0 ./usr/opt/OAT100 OATODB100
0 ./usr/opt/OAT100/bin OATODB100
0 ./usr/opt/OAT100/bin/odb_start OATODB100
0 ./usr/var RESERVED 1
0 ./usr/var/opt RESERVED 1
0 ./usr/var/opt/OAT100 OATODB100
0 ./usr/var/opt/OAT100/log_files OATODB100
2 ./usr/var/opt/OAT100/log_files/odb.log OATODB100 2
0 ./usr/var/opt/OAT100/templates OATODBTEMPS100
0 ./usr/var/opt/OAT100/templates/odb_template OATODBTEMPS100
```

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

2 The `./usr/var/opt/OAT100/log_files/odb.log` file has the `OATODB100` subset identifier, indicating that the file belongs in that subset. The Flags field is set to `2`, indicating that the file may change to another file on the customer's system and that is not linked.

3.3 Creating the Key File

The **key file** defines product attributes such as the product name, product version, and subset definitions, as well as the name of the kit's master inventory file. It consists of a product attributes section and a subset descriptor section. The key file name must consist of the product code and version followed by `.k`, so that `OAT100.k` is the key file for the ODB kit. Example 3–2 illustrates the ODB product kit key file.

Example 3–2: Sample ODB Kit Key File

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

As shown in Example 3–2, the key file is divided into two sections separated by a line that contains two percent signs (`%%`):

1. The product attributes portion of the file describes the naming conventions for the kit and provides kit-level instructions for the `kits` command. This section of the key file consists of several lines of **attribute-value** pairs as described in Table 3–2. The order of these attribute-value paris is not significant. Each attribute name is separated from its value by an equal sign (=). You can include comment lines, which begin with a number sign (#).
2. The value of the `MI` attribute contains the path to the master inventory file. This may be either an absolute path or a relative path from the directory where the `kits` command is executed.
3. The `COMPRESS` attribute has a value of `0` for uncompressed subsets or `1` for compressed subsets. User and kernel product kit subsets may be compressed or uncompressed.
4. The subset descriptor portion of the file describes each of the subsets in the kit and provides subset-level instructions for the `kits` command. This section contains one line for each subset in the kit. Each line consists of four fields, each separated by a single Tab character. You cannot include comments in this section of the key file. Table 3–3 describes the subset descriptor fields.
5. In this entry, the Dependency list field for `OATODB100` is . (dot), meaning that the subset has no dependencies.

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

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

The Subset Description field must be enclosed in single quotes.

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

Table 3–2: Key File Product Attributes

Attribute	Description
`NAME`	The product name; for example, `'Orpheus Document Builder'`. Enclose the product name in single quotation marks (`'`) if it contains spaces.
`CODE`	A unique three-character product code, for example, `OAT`. The first character must be a letter. The first three letters of a subset name must be the same as the product code. In this guide, `OAT` is the three character code assigned to the fictional *Orpheus Authoring Tools, Inc.* company. Several product codes are reserved, including (but not limited to) the following: DNP, DNU, EPI, FOR, LSP, ORT, OSF, SNA, UDT, UDW, UDX, ULC, ULT, ULX, and UWS. Send mail to `Product@DSSR.enet.dec.com` to request a product code.
`VERS`	A three-digit version code; for example, `100`. The `setld` utility interprets this version code as 1.0.0. The first digit should reflect the product's major release number, the second the minor release number, and the third the upgrade level, if any. The version number cannot be lower than 100. The version number is assigned by the kit developer.
`MI`	The name of the master inventory file. If the master inventory file is not in the same directory where the `kits` utility is run, you must specify the explicit relative path from the utility where you are running the `kits` utility to the directory where the master inventory file resides. The file name of the product's master inventory file consists of the product code and version plus the `.mi` extension. You create and maintain the master inventory file with the `newinv` utility.

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

Attribute	Description
`ROOT`	Not illustrated in the example, the operating system has reserved this optional attribute for the base operating system. `ROOT` has a string value that names the root image file. Do not use this attribute for a layered product.
`COMPRESS`	An optional flag that is set to 1 if you want to create compressed subset files. For kits in Direct CD–ROM (DCD) format, you must set this flag to 0 (zero). Compressed files require less space on the distribution media (sometimes as little as 40% of the space required by uncompressed files), but they take longer to install than uncompressed files. If missing, this flag defaults to 0 (zero).

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

Table 3–3: Key File Subset Descriptors

Field	Description
Subset identifier	A character string up to 80 characters in length, composed of the product code (for example, `OAT`), a mnemonic identifying the subset (for example, `ODB`), and the three-digit version code (for example, `100`). In this example, the subset identifier is `OATODB100`. All letters in the subset identifier must be uppercase.
Dependency list	Either a list of subsets upon which this subset is dependent (`OATODB100|OSFDCMT500`), or a single period (.) indicating that there are no subset dependencies. Separate multiple subset dependencies with the pipe character (|).

Table 3–3: Key File Subset Descriptors (cont.)

Field	Description
Flags	A 16-bit unsigned integer; the operating system defines the use of the lower 8 bits. Set bit 0, the sticky bit, to indicate that the subset cannot be removed. Set bit 1 to indicate that the subset is optional. Set bit 2 to indicate that the subset is uncompressed. Bits 3-7 are reserved for future use. You can use bits 8-15 to relay special subset-related information to your subset control program.
Subset description	A short description of the subset, delimited by single quotation marks ('); for example, `'Document Builder Tools'`. The percent sign character (`%`) is reserved in this field and must not be used for layered products.

3.4 Creating Subset Control Programs

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

- Creating SCP source files
- Setting up initial SCP processing
- Working in a DMS environment
- Stopping the installation
- Associating SCP tasks with `setld` processing phases
- Creating SCPs for different product types

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

- Some of your kit's files have to be customized before the product will work properly.
- You want to offer the user the option to install some of the files in a nonstandard location.
- You want to register and statically or dynamically configure a device driver.
- Your kit depends on the presence of other products.
- You need to establish nonstandard permissions or ownership for certain files.
- Your kit requires changes in system files such as `/etc/passwd`.

A subset control program can perform all of these tasks.

3.4.1 Creating SCP 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 your subset control program must be executable on all platforms on which the kit can be installed. If your product works on more than one hardware platform, you cannot write your subset control program in a compiled language. For this reason, it is recommended that you write your subset control program as a script for `/sbin/sh`. All of the examples in this chapter are written in this way.

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 `OATODBTEMPS100`. If each of these subsets required a subset control program, the source file names would be `OATODB100.scp` and `OATODBTEMPS.scp`.

When you create the subsets as described in Section 3.5, the `kits` utility copies the subset control programs from the `./data/scps` directory to the `./output/instctrl` directory. If a subset has no SCP, the `kits` utility creates an empty subset control program file for it in the `./output/instctrl` directory.

3.4.2 Setting Up Initial SCP Processing

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

- Include library routines
- Set global variables

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

3.4.2.1 Including Library Routines

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

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

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

Table 3–4: Available SCP Library Routines

Purpose	Library Routine
Dependency checking	`STL_DepInit`
	`STL_DepEval`
Architecture checking	`STL_ArchAssert`
Dependency locking	`STL_LockInit`
	`STL_DepLock`
	`STL_DepUnLock`
Dataless environment checking	`STL_IsDataless`
	`STL_NoDataless`
Forward symbolic linking	`STL_LinkCreate`
	`STL_LinkRemove`
Backward symbolic linking	`STL_LinkInit`
	`STL_LinkBack`
SCP initialization	`STL_ScpInit`

3.4.2.2 Setting Global Variables

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

Note

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

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

Table 3–5 lists global variables that the subset control program can use to access information about the current subset.

Table 3–5: STL_ScpInit Global Variables

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

3.4.3 Working in a Dataless Environment

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

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

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

A subset control program may need to perform differently in a dataless environment or disallow installation of the subset on such a system. If the product will be installed onto a DMS server, do not not specify absolute pathnames in your subset control program. Otherwise, the `setld` utility will install the product into a dataless area of `/var/adm/dms/dms`*N*`.alpha` rather than `root` (`/`), as if it were installing onto the system itself.

Caution

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

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

`STL_IsDataless`

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

`STL_NoDataless`

Declines installation of a subset into a dataless environment.

3.4.4 Stopping the Installation

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

To stop the installation or deletion of the subset, the subset control program must return a nonzero status to the `setld` utility upon exiting from the particular phase for which it was called. If the subset control program returns a status of 0 (zero), the `setld` utility assumes that the subset control program is satisfied that the `setld` process should continue.

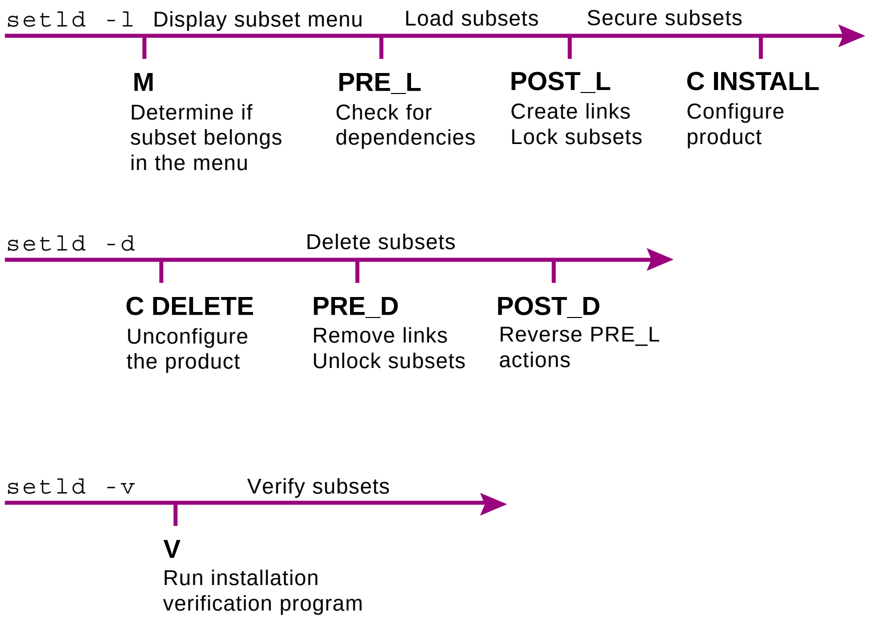
3.4.5 Associating SCP Tasks with setld Utility Phases

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

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

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

Figure 3–2: Time Lines for setld Utility Phases



ZK-1220U-AI

The actions taken by the `setld` utility are shown above the time lines. The SCP actions taken during each `setld` processing phase are shown below the time lines.

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

Caution

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

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

- Displaying the subset menu (`M` phase)
- Before loading the subset (`PRE_L` phase)
- After loading the subset (`POST_L` phase)
- After securing the subset (`C INSTALL` phase)
- Before deleting a subset (`C DELETE` phase)
- Before deleting a subset (`PRE_D` phase)
- After deleting a subset (`POST_D` phase)
- Verifying the subset (`V` phase)

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

3.4.5.1 Displaying the Subset Menu (M Phase)

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

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

Example 3–3: Sample setld Installation Menu

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

Enter your choices or press RETURN to redisplay menus.

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

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

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

When `setld` extracts a subset into a RIS server's product area, the server also executes the subset control program to make use of the program's code for the `M` phase of installation. You should code the `M` phase to detect the difference between extraction of the subset into a RIS area and loading of the subset for use of its contents. To make this determination, check the value of the `$1` command argument (either –`x` for RIS extraction or –`l` for loading). For RIS extraction, the subset control program should do nothing during the `M` phase. When loading subsets, the SCP should perform a `machine` test. The following Bourne shell example illustrates one way to code the `M` phase. In Example 3–4, the subset control program is checking to determine the type of processor on which it is running. In this example, there is no special code for the RIS extract case.

Example 3–4: Sample Test for Alpha Processor During M Phase

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

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

`0` - offer the subset on the menu
`1` - do not offer the subset on the menu

Note

Installation for a dataless client uses the client's local copy of the `machine` shell script even though the installation is performed in a DMS area on the server. Refer to Section 3.4.3 and *Sharing Software on a Local Area Network* for more information about DMS.

3.4.5.2 Before Loading the Subset (PRE_L Phase)

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

Overwriting base operating system files can cause the following problems:

- Your product can be corrupted during an update installation of the operating system. The update installation will overwrite any file that was shipped as part of the old version of the operating system with the version of the file shipped with the operating system.
- An update installation may not complete successfully if you overwrite a base operating system file. This can make the system unusable.
- Your product may have to be removed from the system to complete an update installation. Your product would have to be reinstalled after the update installation is completed.

If your subset control program is designed to overwrite existing files, it first should make a backup copy of the original file during the `PRE_L` phase described in Section 3.4.5.2 and restore the copy in the `POST_D` phase described in Section 3.4.5.7.

The subset control program can 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. When you check for subset dependencies in the subset control program, you can specify what action to take. For example, since the `setld` utility can install and remove subsets, a system administrator could attempt to remove one or more subsets on which your product depends. Because those subsets do not depend on your product's subsets, the `setld` utility usually removes them without question, leaving your product unusable or disabled. You can prevent this inadvertent destruction of your product's environment by determining the subsets on which your subset depends and then **locking** them during the `POST_L` phase described in Section 3.4.5.3.

The `/usr/share/lib/shell/libscp` file contains several Bourne shell script routines to help you implement dependency management. These routines use dependency expressions to examine conditions on the system. A **dependency expression** is a postfix logical expression that describes the conditions upon which the subset depends. Dependency expressions are left to right recursive and are processed using conventional postfix techniques. Dependency expressions are defined in Backus-Naur form, as follows:

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

Table 3–6 lists the elements of a dependency expression (*depexp*):

Table 3–6: Dependency Expression Elements

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

The following are valid dependency expressions:

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

The last of these expressions evaluates as follows:

- The `and` operator is satisfied if both `SUBSETX100` and `SUBSETY200` are present.

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

You can call the following routines to perform dependency checking:

`STL_DepInit`

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

`STL_DepEval` *depexp*

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

In Example 3–5, the subset control program checks a list of files to be backed up if they already exist on the system. If it finds any, it creates a backup copy with an extension of .OLD.

Example 3–5: Sample Backup of Existing Files During PRE_L Phase

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

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

2 If the specified file exists, a backup copy is made. This will be restored in the POST_D phase.

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

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

3.4.5.3 After Loading the Subset (POST_L Phase)

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

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

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

Caution

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

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

`STL_LinkInit`

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

STL_LinkBack *link_file file_path link_path*

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

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

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

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

1 The STL_LinkBack routine creates a backward link in the product-specific area. If you used the STL_LinkCreate routine, it would create an uacceptable link in the OSF name space. The STL_ScpInit routine sets the value of the $_PVCODE global variable to OAT100. Using this variable allows the SCP to be used for the next version of the product, OAT200, without changing the path names.

2 The SCP uses the `rm` command to remove the links created in the `POST_L` phase. The `STL_LinkRemove` routine is used only to remove links created by the `STL_LinkCreate` routine.

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

`0` - continue subset configuration
`1` - terminate subset configuration; create a marker to indicate a corrupt subset

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

- A *`subset-id`*`.lk` lock file to indicate successful subset installation
- A *`subset-id`*`.dw` lock file to indicate failed corrupt subset installation

The `setld` utility creates one of these two empty lock files when it installs a subset. After successful installation, that subset is then available for dependency checks and locking performed when other subsets are installed later. A subset's lock file can then contain any number of records, each naming a single dependent subset.

For example, the ODB kit requires that some version of the Orpheus Document Builder base product must be installed for the ODB product to work properly. Suppose that the `OATBASE200` subset is present. When the `setld` utility installs the `OATODBTEMPS100` subset from the ODB kit, it inserts a record that contains the subset identifier `OATODBTEMPS100` into the `OSFDCMT500.lk` file. When the system administrator uses the `setld` utility to remove the `OSFDCMT500` subset, the `setld` utility checks `OSFDCMT500.lk` and finds a record that indicates that `OATODBTEMPS100` depends on `OSFDCMT500`, displays a warning message, and requires confirmation that the user really intends to remove the `OSFDCMT500` subset.

If the administrator removes the `OATODBTEMPS100` subset, the `setld` utility removes the corresponding record from the `OSFDCMT500.lk` file. Thereafter, the administrator can remove `OSFDCMT500` 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 `OSFDCMT5`*`nn`*`.lk` file:

```
STL_DepLock OATODB100 OSFDCMT5??
```

3.4.5.4 After Securing the Subset (C INSTALL Phase)

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

Note

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

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

Example 3–7 shows the `C INSTALL` portion of the SCP that issues a message upon successful subset installation.

Example 3–7: Sample Message Output During C INSTALL Phase

```
#
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
 .
 .
 .
#
# This is the configuration phase of the SCP
#
C)
 #
 # Setld invokes the C phase with an argument that is
 # either INSTALL or DELETE.  The INSTALL argument is
 # used on a setld load, while the DELETE argument is
 # used on a setld delete.
 #
 case $1 in
 INSTALL) 1
  #
  # Output a message letting the user know
  # that they should read the README file
  # before using the product.
  #
    echo "
The installation of the $_DESC ($_SUB) 2
software subset is complete.
Please read the /opt/$_PVCODE/README.odb file before 3
using the $_DESC product." 2
  ;;
 .
 .
 .
 esac
 ;;
 .
 .
 .
```

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

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

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

3 The `STL_ScpInit` routine sets the value of the `$_PVCODE` global variable to `OAT100`. Using this variable allows the SCP to be used for

the next version of the product, OAT200, without changing the path names.

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

0 - successful load and configure
1 - unsuccessful load and configure; create a marker to indicate a corrupt subset

3.4.5.5 Before Deleting a Subset (C DELETE Phase)

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

Note

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

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

Example 3–8: Sample C DELETE Phase

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

Example 3–8: Sample C DELETE Phase (cont.)

```
  #
    echo "
The installation of the $_DESC ($_SUB)
software subset is complete.
Please read the /opt/$_PVCODE/README.odb file before
using the $_DESC product."
  ;;
   DELETE)
       ;; 1
   esac
 ;;
  .
  .
  .
```

1 This phase should reverse any changes made during the `C INSTALL` phase. Since no changes were made in Example 3–7, no action is taken in the `C DELETE` phase.

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

`0` - continue with the delete
`1` - terminate the delete, leave subset installed and intact

3.4.5.6 Before Deleting a Subset (PRE_D Phase)

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

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.

Example 3–6 uses `STL_LinkBack` in the `POST_L` phase to create a link named `/opt/OAT100/sbin/ls` that refers to the real file `/sbin/ls`. Example 3–9 shows the SCP removing this link in the `PRE_D` phase.

Example 3–9: Sample PRE_D Phase Reversal of POST_L Phase Actions

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

[1] The SCP uses the `rm` command to remove the links created in the `POST_L` phase. The `STL_LinkRemove` routine is only used to remove links created by the `STL_LinkCreate` routine.

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

`0` - continue with the deletion
`1` - terminate the deletion, leave subset installed and intact

3.4.5.7 After Deleting a Subset (POST_D Phase)

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

In Example 3–10, the subset control program checks a list of files to be backed up if they already exist on the system. If it finds any, it restores the backup copy.

Example 3–10: Sample File Restoration During POST_D Phase

```
        ⋮
#
# Here is a list of files to back up if found on
# the installed system.
#
BACKUP_FILES="\
 ./usr/var/opt/$_PVCODE/templates/odb_template \ [1]
 ./usr/var/opt/$_PVCODE/log_files/odb.log"
#
# The ACT variable is set by setld and determines which
# phase of the SCP should be executed.
#
case $ACT in
```

Example 3–10: Sample File Restoration During POST_D Phase (cont.)

```
       .
       .
       .
#
# This is the post-deletion phase of the SCP
#
POST_D)
 #
 # Restore the backup copies created during the PRE_L phase
 #
 for FILE in $BACKUP_FILES
 do
   [ -f $FILE.OLD ] &&
   mv $FILE.OLD $FILE [2]
 done
 ;;
       .
       .
       .
esac
```

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

[2] Restores any files backed up in the `PRE_L` phase, as shown in Example 3–5.

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

`0` - continue with the deletion
`1` - terminate the deletion; subset files have been removed and evidence of installation has been removed except for changes that failed in the `POST_D` phase

3.4.5.8 Verifying the Subset (V Phase)

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

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

3.4.6 Creating SCPs for Different Product Kit Types

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

- Creating SCPs for user product kits
- Creating SCPs for kernel product kits

3.4.6.1 User Product Kit SCPs

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

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

Example 3–11: Sample ODB User Product SCP

```
#!/sbin/sh

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

#
# Initialize the global variables, except in the M phase
#
if [ "$ACT" != "M" ]
then
 STL_ScpInit
fi

#
# Here is a list of files to back up if found on
# the installed system.
#
BACKUP_FILES="\
 ./usr/var/opt/$_PVCODE/templates/odb_template \
 ./usr/var/opt/$_PVCODE/log_files/odb.log"

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

#
# This is the menu phase of the SCP
#
M)
 #
 # Setld invokes the M phase with an argument and if
 # the argument is "-l" it means that a software load
 # is occurring.
 #
```

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

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

PRE_L)
 #
 # Loop through the list of backup files and create
 # backup copies for any file that is found on the
 # system.
 #
 for FILE in $BACKUP_FILES
 do
  #
  # If the file to be backed up exists, create
  # a backup copy with a .OLD extention.
  #
  if [ -f $FILE ]
  then
   cp $FILE $FILE.OLD
  fi
 done
 ;;

POST_L)
 #
 # Initializes the variables so that the STL_LinkBack
 # routine can be executed
 #
 STL_LinkInit

 #
 # Create a symbolic link in the ./opt/$_PVCODE/sbin
 # directory that points to the ./sbin/ls file.
 #
 STL_LinkBack ls ./sbin ./opt/$_PVCODE/sbin
 ;;

PRE_D)
 #
 # Remove the links created in the POST_L phase
 #
 rm -f ./opt/$_PVCODE/sbin/ls
 ;;

POST_D)
 #
 # Restore the backup copies created during the PRE_L phase
 #
 for FILE in $BACKUP_FILES
```

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

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

C)
 #
 # Setld invokes the C phase with an argument that is
 # either INSTALL or DELETE.  The INSTALL argument is
 # used on a setld load, while the DELETE argument is
 # used on a setld delete.
 #
 case $1 in
 INSTALL)
  #
  # Output a message letting the user know
  # that they should read the README file
  # before using the product.
  #
  echo "
The installation of the $_DESC ($_SUB)
software subset is complete.

Please read the /opt/$_PVCODE/README.odb file before
using the $_DESC product."
  ;;

 DELETE)
  ;;

 esac
 ;;

esac

exit 0
```

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

Note

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

3.4.6.2 Kernel Product Kit SCPs

In addition to the optional processing described in Section 3.4.5, a subset control program for a kernel product such as a device driver must also configure the driver into the kernel. When building subset control programs for a kernel product, you can choose one of the following configuration strategies:

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

Example 3–12 shows the subset control program for the single binary module associated with the `odb_kernel` driver. The user can choose to configure this single binary module into the kernel either statically or dynamically. The subset control program runs the `doconfig` utility to configure the driver into the kernel.

Example 3–12: Sample ODB Kernel Product SCP

```
#!/sbin/sh

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

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

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

#
# This routine rebuilds the static kernel
#
Rebuild_Static_Kernel()
{
 HNAME='/sbin/hostname -s'
 HOSTNAME='ToUpper $HNAME'
 if doconfig -c $HOSTNAME
 then
  echo "\nThe /sys/${HOSTNAME}/vmunix kernel has been"
  echo "moved to /vmunix and the changes will take effect"
  echo "the next time the system is rebooted."
  return 0
 else
  Error "
An error occurred while building the static kernel."
```

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

```
  return 1
 fi
}


#
# Initialize the global variables, except in M phase
#
if [ "$ACT" != "M" ]
then
 STL_ScpInit
fi

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

C)
 #
 # The kreg database file where all of the kernel
 # layered products are registered.
 #
 KREGFILE=./usr/sys/conf/.product.list

 case $1 in
 INSTALL)

  #
  # Merge the graphics support into the existing
  # /etc/sysconfigtab file
  #
  sysconfigdb -m -f ./opt/$_PVCODE/etc/sysconfigtab odb_graphics

  echo "*** $_DESC Product Installation Menu ***\n"
  echo "1. Statically configure the graphics support"
  echo "2. Dynamically configure the graphics support"
  echo "\nType the number of your choice []: \c"

  read answer
  case ${answer} in
  1)
   #
   # Determine if the product is already registered
   # with the kreg database, and if it is, skip
   # registering it.
   #
   grep -q $_SUB $KREGFILE
   if [ "$?" != "0" ]
   then
    #
    # Register the product with the
    # kernel using kreg
    #
    /sbin/kreg -l $_PCODE $_SUB \
     ./opt/$_PVCODE/sys/BINARY
   fi

   #
```

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

```
   # Rebuild the static kernel
   #
   Rebuild_Static_Kernel

   #
   # Successful rebuild, so back up the existing
   # kernel and move the new one into place.
   #
   if [ "$?" = "0" ]
   then
    #
    # Make a backup copy of the kernel
    # as it existed prior to installing
    # this subset.  Since a subset can
    # be installed more than once (due to
    # load/configuration failures or even
    # because the user removed files) make
    # sure that the backup does not already
    # exist.
    #
    if [ ! -f /vmunix.pre_${_SUB} ]
    then
     mv /vmunix /vmunix.pre_${_SUB}
    fi

    #
    # Move the new kernel into place
    #
    mv /sys/${HOSTNAME}/vmunix /vmunix

    #
    # Place a marker on the system so that
    # upon subset removal the SCP can
    # determine if it needs to remove a
    # static or dynamic configuration.
    #
    touch ./opt/$_PVCODE/sys/BINARY/odb_graphics_static
   fi
   ;;

  2)
   #
   # Dynamically load the odb_graphics subsystem
   # into the kernel
   #
   sysconfig -c odb_graphics
   ;;

  esac
  ;;

 DELETE)
  #
  # If the marker is present then the kernel option
  # was added statically.
  #
  if [ -f ./opt/$_PVCODE/sys/BINARY/odb_graphics_static ]
  then
   #
   # Clean-up the marker
```

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

```
   #
   rm -f ./opt/$_PVCODE/sys/BINARY/odb_graphics_static

   #
   # Deregister the product using kreg
   #
   /sbin/kreg -d $_SUB

   #
   # Rebuild the static kernel
   #
   Rebuild_Static_Kernel

   #
   # Successful rebuild, remove the old backup
   # copy that was created when we installed.
   #
   if [ "$?" = "0" ]
   then
    mv /sys/${HOSTNAME}/vmunix /vmunix
    rm -f /vmunix.pre_${_SUB}
   fi
  else
   #
   # Unload the dynamic kernel module
   #
   sysconfig -u odb_graphics
  fi

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

esac

exit 0
```

3.5 Producing Subsets

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

- The compression flag file
- The image data file
- The subset control files
- The subset inventory file

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

Use the following syntax for the `kits` command:

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

- The mandatory *key-file* parameter is the path name of the key file created in Section 3.3.
- The mandatory *input-path* parameter specifies the top of the file hierarchy that contains the source files.
- The mandatory *output-path* parameter specifies the directory to be used to store the subset images and data files produced.
- The optional *subset* parameter specifies the name of an individual subset to be built. You may specify multiple subsets in a space-separated list. If you use the *subset* argument, the `kits` utility assumes the following:
  - Only the subsets named as arguments to this parameter are to be built.
  - The *key-file* contains descriptors for each of the named subsets.
  - All other subsets in the product have been built already.
  - The *output-path* directory contains images of the previously built subsets.

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

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

Caution

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

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

```
# cd /mykit/data
# kits OAT100.k ../src ../output
%%
Creating 2 Orpheus Document Builder subsets.
1   Subset OATODB100 1

        Generating media creation information...done
        Creating OATODB100 control file...done.
        Making tar image...done. [2]
        Compressing [3]
           OATODB100: Compression: 92.64%
 -- replaced with OATODB100.Z

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

2   Subset OATODBTEMPS100 [1]

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

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

Creating OAT.image [4]

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

Media image production complete.
```

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

[1] Creates the subsets.

[2] If the subset is not in DCD format, creates a `tar` image of the subset.

[3] Compresses each subset if you specify the `COMPRESS` attribute in the key file.

[4] Creates the image data file `OAT.image`.

[5] Creates the `INSTCTRL` file, which contains a `tar` image of all the following installation control files:

- Compression flag file *`product-id`*`.comp`
- Image data file *`product-code`*`.image`
- Subset control file *`subset-id`*`.ctrl`
- Subset inventory file *`subset-id`*`.inv`
- Subset control program file *`subset-id`*`.scp`

These files are described in Table 3–7.

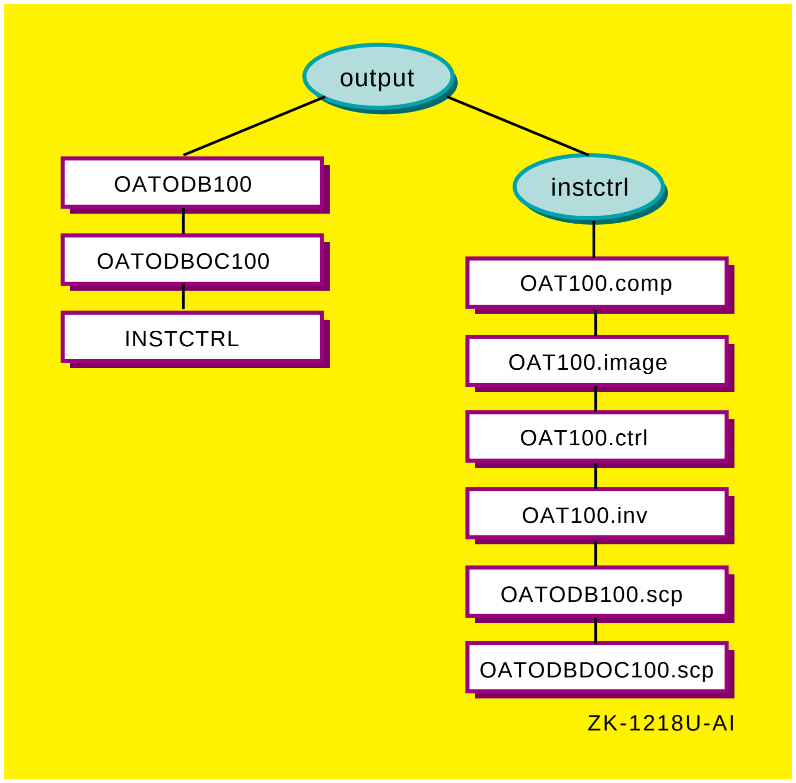
The `INSTCTRL` file is placed in the output directory.

Table 3–7: Installation Control Files in the instctrl Directory

File	Description
product-id.comp	Compression flag file. This empty file is created only if you specified the COMPRESS attribute in the key file. Its presence signals to the `setld` utility that the subset files are compressed. The ODB kit's compression flag file is named `OAT100.comp`.
product-code.image	Image data file. This file contains size and checksum information for the subsets.
subset-id.ctrl	Subset control file. This file contains the `setld` utility control information. There is one subset control file for each subset.
subset-id.inv	Subset inventory file. This file contains an inventory of the files in the subset. Each record describes one file. There is one subset inventory file for each subset.
subset-id.scp	Subset control program. If you created subset control programs for your kit, these files are copied from the `scps` directory to the `instctrl` directory. There is one subset control program for each subset; if you have not created a subset control program for a subset, the `kits` utility creates a blank file.

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

Figure 3–3: ODB output Directory



The subset files and the `instctrl` files are constituents of the final kit. The following sections describe the contents of the installation files.

3.5.1 Compression Flag File

The **compression flag file** is an empty file whose name consists of the product code and the version number with the string `comp` as a suffix; for example, `OAT100.comp`. If the compression flag file exists, the `setld` utility knows that the subset files are compressed.

3.5.2 Image Data File

The **image data file** is used by the `setld` utility to verify subset image integrity before starting the actual installation process. The image data file name consists of the product code with the string `image` as a suffix. The image data file contains one record for each subset in the kit; Table 3–8 describes the three fields in each record.

Table 3–8 describes the image data file.

Table 3–8: Image Data File Field Descriptions

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

Example 3–13 shows `OAT.image`, the image data file for the ODB kit:

Example 3–13: Sample Image Data File

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

3.5.3 Subset Control Files

The `setld` utility uses the subset control files as a source of descriptive information about subsets. Subset control file fields are described in Table 3–9.

Table 3–9: Subset Control File Field Descriptions

Field	Name	Description
1	`NAME`	Specifies the product name; from the Name field in the Key File.
2	`DESC`	Briefly describes the subset; from the Subset Description field in the Subset Descriptor section of the Key File.
3	`ROOTSIZE`	Specifies (in bytes) the space the subset requires in the `root` (/) file system.
4	`USRSIZE`	Specifies (in bytes) the space the subset requires in the `usr` file system; calculated by the `kits` utility at execution.
5	`VARSIZE`	Specifies (in bytes) the space the subset requires in the `var` file system; calculated by the `kits` utility at execution.
6	`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); calculated by the `kits` utility at execution.

Table 3–9: Subset Control File Field Descriptions (cont.)

Field	Name	Description
7	`MTLOC`	Specifies the tape volume number and subset's location on the tape as two colon-separated integers (the volume number of the tape that contains the subset archive and the file offset at which the subset archive begins). On tape volumes, the first three files are reserved for a bootable operating system image and are not used by the `setld` utility. An offset of 0 (zero) indicates the fourth file on the tape. The fourth file is a `tar` archive named `INSTCTRL`, which contains the kit's installation control files (listed in Table 3–7). Calculated by the `kits` utility at execution.
8	`DEPS`	Specifies either a list of subsets upon which this subset is dependent (`DEPS="OATODB100 OSFDCMT500"`), or a single period (`DEPS="."`) indicating that there are no subset dependencies. If there is more than one subset dependency, each subset name is separated by a Space character. From the Dependency List field in the Subset Descriptor section of the Key file.
9	`FLAGS`	Specifies the value in the flags field of the subsets record in the key file. Bit 0 is the sticky bit which indicates that the subset cannot be removed. Bit 1 indicates that the subset is optional. Bits 2 to 7 are reserved; bits 8 to 16 are undefined. From the Flags field in the Subset Descriptor section of the Key file.

Example 3–14 shows `OATODB100.ctrl`, the control file for the ODB kit's `OATODB100` subset:

Example 3–14: Sample Subset Control File

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

3.5.4 Subset Inventory File

The **subset inventory file** describes each file in the subset, listing its size, checksum, permissions, and other information. The `kits` utility generates this information, which reflects the exact state of the files in the source hierarchy from which the kit was built. The `setld` utility uses the information to duplicate that state, thus transferring an exact copy of the source hierarchy to the customer's system. Table 3–10 describes subset inventory file fields.

Each record of the inventory is composed of 12 fields, each separated by single Tab characters. Table 3–10 describes the contents of these fields.

Table 3–10: Subset Inventory File Field Descriptions

Field	Name	Description
1	Flags	A 16-bit unsigned integer. Bit 1 is the `v` (volatility) bit. When set, changes to the existing copy of the file can occur during kit installation. It usually is set for files such as `usr/spool/mqueue/syslog`. Bit 2 is the `l` (link) bit. When set, the `STL_LinkCreate` routine creates a forward link from the standard system directories to the layered product areas. The remaining bits are reserved; possible values for this field are therefore 0, 2, or 4.
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 six-digit octal representation of the file's mode.
7	Date	The file's last modification date.
8	Revision	The version code of the product that includes the file.
9	Type	A letter that describes the file: `b` – Block device. `c` – Character device. `d` – Directory containing one or more files. `f` – Regular file. For regular files with a link count greater than one, see file type `l`. `l` – Hard link. Other files in the inventory have the same inode number. The first (in ASCII collating sequence) is listed in the referent field. `p` – Named pipe (FIFO). `s` – Symbolic link.

Table 3–10: Subset Inventory File Field Descriptions (cont.)

Field	Name	Description
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.

Example 3–15 shows the `OATODB100.inv` inventory file for the ODB kit's `OATODB100` subset.

Example 3–15: Sample Subset Inventory File

```
0 8192 00000 0 0 040755 5/13/99 100 d\
 ./opt/OAT100 none OATODB100
0 133 21616 0 0 100644 5/11/99 100 f\
 ./opt/OAT100/README.odb none OATODB100
0 8192 21616 0 0 040755 5/11/99 100 d\
 ./opt/OAT100/sbin none OATODB100
0 151 28636 3 4 100755 5/11/99 100 f\
 ./opt/OAT100/sbin/odb_recover none OATODB100
0 8192 28636 0 0 040755 5/13/99 100 d\
 ./usr/opt/OAT100 none OATODB100
0 8192 28636 0 0 040755 5/11/99 100 d\
 ./usr/opt/OAT100/bin none OATODB100
0 75 26280 0 0 100755 5/11/99 100 f\
 ./usr/opt/OAT100/bin/odb_start none OATODB100
0 8192 26280 0 0 040755 5/11/99 100 d\
 ./usr/var/opt/OAT100 none OATODB100
0 8192 26280 0 0 040755 5/11/99 100 d\
 ./usr/var/opt/OAT100/log_files none OATODB100
4 0 00000 0 0 100644 5/11/99 100 f\
 ./usr/var/opt/OAT100/log_files/odb.log none OATODB100
```

Note

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

Fields are separated by single Tab characters.

3.6 Testing Subsets

You must test your subsets to ensure that they can be loaded onto a running system, that the product runs on the system, and that the subsets can be deleted. The following sections describe this testing:

1. Loading all of the subsets onto a running system
2. Deleting all of the subsets from a running system
3. If your kit includes optional subsets, loading only the mandatory subsets onto a running system

It is important that you perform these tests in sequence.

3.6.1 Loading All Subsets

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

Perform the following procedure to test the loading of all subsets:

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

   ```
   # setld -l /mykit/output
   ```
3. When prompted, select the option to install all subsets from the `setld` installation menu.
4. Verify that all files in your subsets were loaded. If any files are missing, check the master inventory file. Subset inventory files are created from master inventory file entries.
5. Verify each file's installed location, permissions, owner, and group. The `setld` utility uses the information in the subset inventory file to determine these attributes. If any are incorrect, modify the file in the source directory and rebuild the master inventory file and the subsets.
6. If you supplied SCP files, verify any actions that should have occurred in the `M`, `PRE_L`, `POST_L`, and `C INSTALL` phases. Refer to Section 3.4.5 for discussions of SCP tasks associated with these phases.
7. After successful installation, test all commands or utilities included with your product. Since file locations may have changed, especially if you installed in the `/opt`, `/usr/opt`, or `/usr/var/opt` directories, it

is important that you test your product thoroughly to verify that everything works correctly.

3.6.2 Removing All Subsets

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

Perform the following procedure to test the removal of all subsets:

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

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

3. Verify that all files loaded onto your system in Section 3.6.1 were deleted.
4. If you supplied SCP files, verify any actions that should have occurred in the `C DELETE`, `PRE_D`, and `POST_D` phases. Refer to Section 3.4.5 for discussions of SCP tasks associated with these phases.

3.6.3 Loading Mandatory Subsets Only

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

Perform the following procedure to test the loading of mandatory subsets only:

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

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

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

6. If you supplied SCP files, verify any actions that should have occurred in the `M`, `PRE_L`, `POST_L`, and `C INSTALL` phases. Refer to Section 3.4.5 for discussions of SCP tasks associated with these phases.
7. After successful installation, test all commands or utilities included with your product. Since file locations may have changed, especially if you installed in the `/opt`, `/usr/opt`, or `/usr/var/opt` directories, it is important that you test your product thoroughly to verify that everything works correctly.

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

3.7 Updating the Master Inventory File After Subset Creation

After subsequent updates to the kit source directory, run the `newinv` utility to update the master inventory file, using the existing master inventory file as input. The `newinv` utility performs the following additional steps:

- 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 (new records)
  - Records that represent files now present that were also present in the previous inventory. This list is empty the first time you create the inventory.
  - Records that were in the previous inventory but are no longer present (defunct records). This list is also empty the first time you create the inventory.
- Lets you edit the group of defunct records, 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 3–1).
- Merges the three groups of records and sorts the result to produce a finished master inventory file that matches the source hierarchy.

Run the `newinv` utility to update the master inventory file any time that you modify the source directory by adding or removing files.

4 Producing User Product Kits

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

Note

The information in this chapter describes how to create user product kits. If you want to create a kernel product kit, go to Chapter 5.

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

1. Read Chapter 1 for an overview of product kits.
2. Design the kit directory structure as described in Chapter 2.
3. Create subsets as described in Chapter 3.
4. Create the kit distribution media as described in Section 4.1.
5. Test the distribution media as described in Section 4.2.

No additional installation files are required for user product kits.

4.1 Producing Distribution Media

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

1. Edit the `/etc/kitcap` file.
2. Build the kit on the distribution media:
   - Use the `gentapes` utility to build a `tar` format kit on magnetic tape.
   - Use the `gendisk` utility to build a DCD format kit on disk media.

Although you can use direct CD-ROM (DCD) format, the best way to produce a user product kit is in `tar` format.

- `tar` format

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

- direct CD-ROM (DCD) format

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

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

  Installation time for kits in DCD format is slower than for kits in `tar` format. User product kits only should be produced in `tar` format if they require access to kit files before or during the installation.

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

- Magnetic tape

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

- Diskette

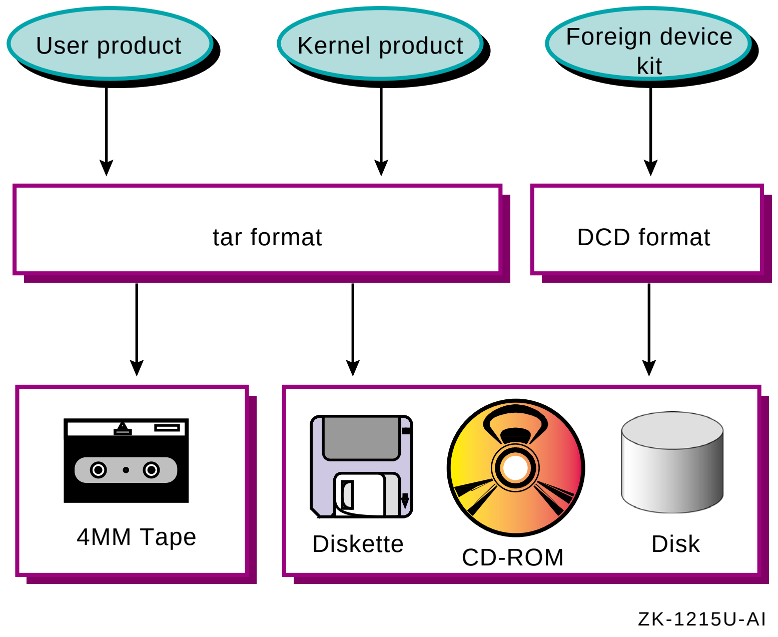
  Diskettes are a good media for testing purposes or for small products. The product must fit on a single diskette; it cannot span multiple diskettes. Use the `gendisk` utility to produce kits for diskette media.

- CD-ROM

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

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

Figure 4–1: User Product Kit File Formats



4.1.1 Editing the /etc/kitcap File

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

When you add a record to the `/etc/kitcap` file, use the following conventions:

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

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

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

4.1.1.1 Tape Media kitcap Record Format

The kitcap record for tape media contains the following elements:

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

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

  ```
  # cd /mykit/output
  # touch space
  # tar -cf SPACE space
  ```
- The instctrl directory, relative to the output directory.
- The names of the subsets that make up the kit. Each subset listed must be stored in one of the specified directories.
- An optional volume identifier. Multiple tapes are supported.

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

Example 4–1 shows the record to be added to the /etc/kitcap file to produce the ODB kit on TK50 tapes:

Example 4–1: Sample /etc/kitcap Record for Magnetic Tape

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

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

4.1.1.2 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.
- The product description. This entry is taken from key file `NAME` field. You must replace any spaces with the underscore (_) character.
- The destination directory for the subsets on the disk media. Allows a hierarchical structure so you can put multiple products on one disk, or put parts of one product on different areas of the same disk.
- Name of the kit's output directory, where the `gendisk` utility can find the product subsets.
- The `instctrl` directory, relative to the output directory specification.
- The names of the subsets that make up the kit.

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

Example 4–2 shows the record to be added to the `/etc/kitcap` file to produce the ODB kit on disk media:

Example 4–2: Sample /etc/kitcap Record for Disk Media

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

Based on the information shown in Example 4–2, the `gendisk` utility places the kit on the `c` partition in the `/` (`root`) directory of the disk media. The product description is `Orpheus_Document_Builder` and the kit output directory is `/mykit/output`. The kit consists of two subsets: `OATODB100` and `OATODBTEMPS100`.

4.1.2 Building a tar Format User Product Kit on Magnetic Tape

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

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

- The `-v` option verifies the product media without writing it first. This assumes that you have already written the files to the product media. The `-v` option cannot be used together with the `-w` option.
- The `-w` option writes the product media without verification. The `-w` option cannot be used together with the `-v` option.
- If you do not use either the `-v` or `-w` option, the `gentapes` utility writes the tape, rewinds it, and then verifies the files in the kit descriptor.
- The optional *`hostname`* argument is the name of a remote TCP/IP network machine that contains the `/etc/kitcap` file. 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 the local system. You can use NFS file sharing to mount the kit files remotely on a system with the required disk drive.
- The mandatory *`product-code`* argument is a user-defined code that describes the product. It should match the product name specified in the `kitcap` record. This is a concatenation of the `NAME` and `VERS` fields of the key file, such as `OAT100`. The `gendisk` utility searches for *`product-code`* in the `/etc/kitcap` file on *`hostname`* and uses it to create the media.
- The mandatory *`special`* argument is the name of the device special file for the disk device, such as `/dev/ntape/dat`.

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

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

4.1.3 Building a User Product Kit on Disk Media

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

Note

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

Use the following syntax for the `gendisk` command:

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

- The `-v` option verifies the product media without writing it first. This assumes that you have already written the files to the product media. The `-v` option cannot be used together with the `-w` option.
- The `-w` option writes the product media without verification. The `-w` option cannot be used together with the `-v` option.
- If you do not use either the `-v` or `-w` option, the `gendisk` utility writes and then verifies the files in the kit descriptor. This default behavior is the same as if you use both the `-v` and `-w` options together.
- The `-d` option specifies that the kit be produced in DCD format. If used, the `-d` option must be after any `-v` or `-w` options on the command line. If you do not use this option, the kit is produced in `tar` format.
- The optional *hostname* argument is the name of a remote TCP/IP network machine that contains the `/etc/kitcap` file. 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 the local system. You can use NFS file sharing to mount the kit files remotely on a system with the required disk drive.
- The mandatory *product-code* argument is a user-defined code that describes the product. It should match the product name specified in the `kitcap` record. This is a concatenation of the `NAME` and `VERS` fields of the key file, such as `OAT100`. The `gendisk` utility searches for *product-code* in the `/etc/kitcap` file on *hostname* and uses it to create the media.

- The mandatory *special* argument is the name of the device special file for the disk device, such as /dev/rdisk/dsk1a. The actual partition that the utility will write the files on is defined in the kitcap file entry for the *product-code* being used, so it makes no difference what partition is appended to the special device file used on the command line.

To prepare a kit on disk for a user product, use the gendisk utility without the -d option. Specify the product name and the device special file name. In Example 4–3, the command creates a kit in tar format for the ODB product on the c partition of the disk named dsk0:

Example 4–3: Sample gendisk Command for ODB User Product in tar Format

```
% gendisk OAT100 /dev/rdisk/dsk0c
```

4.2 Testing the Distribution Media

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

To test a user product kit, log in to the system as superuser or root and run the setld utility. For example, the ODB product on CD-ROM could be tested as follows:

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 cdrom0, you would enter the following command:

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

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

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

   *** Enter subset selections ***

   The following subsets are mandatory and will be installed automatically
   ```

```
unless you choose to exit without installing any subsets:

      * Document Builder Kernel Support
      * Document Builder Tools

The subsets listed below are optional:

 - Other:
     1) Document Builder Templates

Or you may choose one of the following options:

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

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

Enter your choices or press RETURN to redisplay menus.

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

You are installing the following mandatory subsets:

        Document Builder Kernel Support
        Document Builder Tools

You are installing the following optional subsets:

 - Other:
        Document Builder Templates

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

Is this correct? (y/n): y

Checking file system space required to install selected subsets:

File system space checked OK.

3 subset(s) will be installed.

Loading subset 1 of 3 ...

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

Loading subset 2 of 3 ...

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

Loading subset 3 of 3 ...

Document Builder Kernel Support
   Copying from /mykit/output (disk)
   Verifying

3 of 3 subset(s) installed successfully.

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

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

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

Configuring "Document Builder Templates" (OATODBTEMPS100)

Configuring "Document Builder Kernel Support" (OATODBKERNEL100)

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

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

Type the number of your choice []: 1

*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***

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

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

*** PERFORMING KERNEL BUILD ***
        Working....Fri May 14 14:49:25 EDT 1999

The new kernel is /sys/TEST01/vmunix

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

The `setld` utility displays prompts and messages to guide you through the process of selecting the subsets you want to install. After it loads the subsets, the `setld` utility calls the subset control program for each subset.

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

```
# umount /cdrom
```

6. Verify that the installed product functions correctly.

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

5 Producing Kernel Product Kits

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

Note

The information in this chapter describes how to create kernel product kits. If you want to create a user product kit, go to Chapter 4.

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

1. Read Chapter 1 for an overview of product kits.
2. Design the kit directory structure as described in Chapter 2.
3. Create subsets as described in Chapter 3.
4. Create additional installation files as described in Section 5.1.
5. Create the kit distribution media as described in Section 5.2.
6. Test the kernel product kit installation as described in Section 5.3.

5.1 Creating Additional Installation Files

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

- A statically configured product is linked statically into the kernel at build (or bootlink) time and configured at boot time. A static product can be built from source files, binary objects, modules, or all three.

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

Note:

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

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

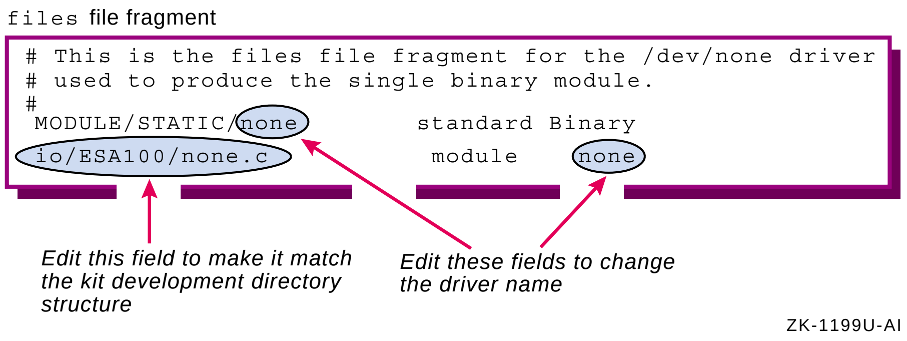
- The `files` file fragment contains information about the location of the source code and modules associated with the driver, tags indicating when the driver is loaded into the kernel, and whether the source or binary form of the driver is supplied to the customer. For both statically and dynamically configured drivers, place this file in a product directory, such as `/OAT100/sys/BINARY`. You need to edit this file if the kit development directory structure differs from the driver development directory structure or if you must change the driver name for any reason.

  Caution

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

  Figure 5–1 shows which fields within the `files` file fragment need to change.

Figure 5–1: Editing the files File Fragment

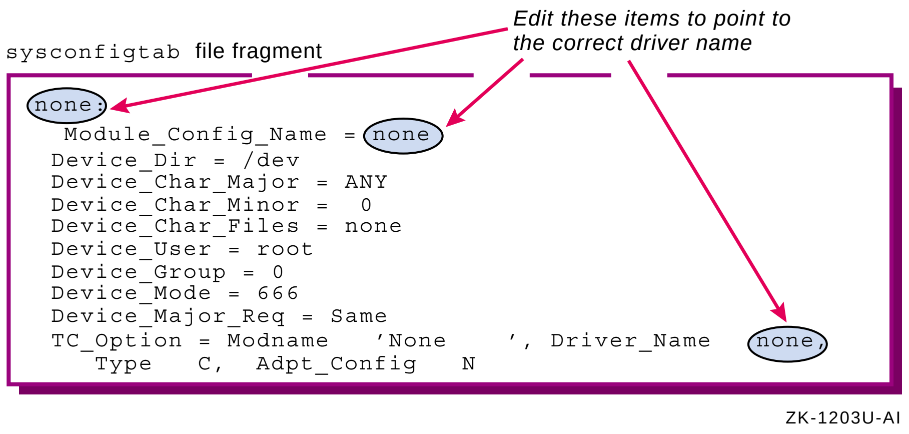

files file fragment
Edit this field to make it match the kit development directory structure
Edit these fields to change the driver name
ZK-1199U-AI

```
# This is the files file fragment for the /dev/none driver
# used to produce the single binary module.
#
 MODULE/STATIC/none         standard Binary
 io/ESA100/none.c            module     none
```

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

  You do not need to change the `sysconfigtab` file fragment unless you change the driver (subsystem) name. The driver name appears in three places within the file, as shown in Figure 5–2. In the example, the driver runs on a TURBOchannel bus (indicated by the `TC_Option` entry), but a similar set of bus options would be specified for other bus types.

Figure 5–2: Editing the sysconfigtab File Fragment


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

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

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

Caution

Compressed modules cannot be linked using RIS. Use the `file` command to determine if a file is compressed. You see output similar to the following:

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

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

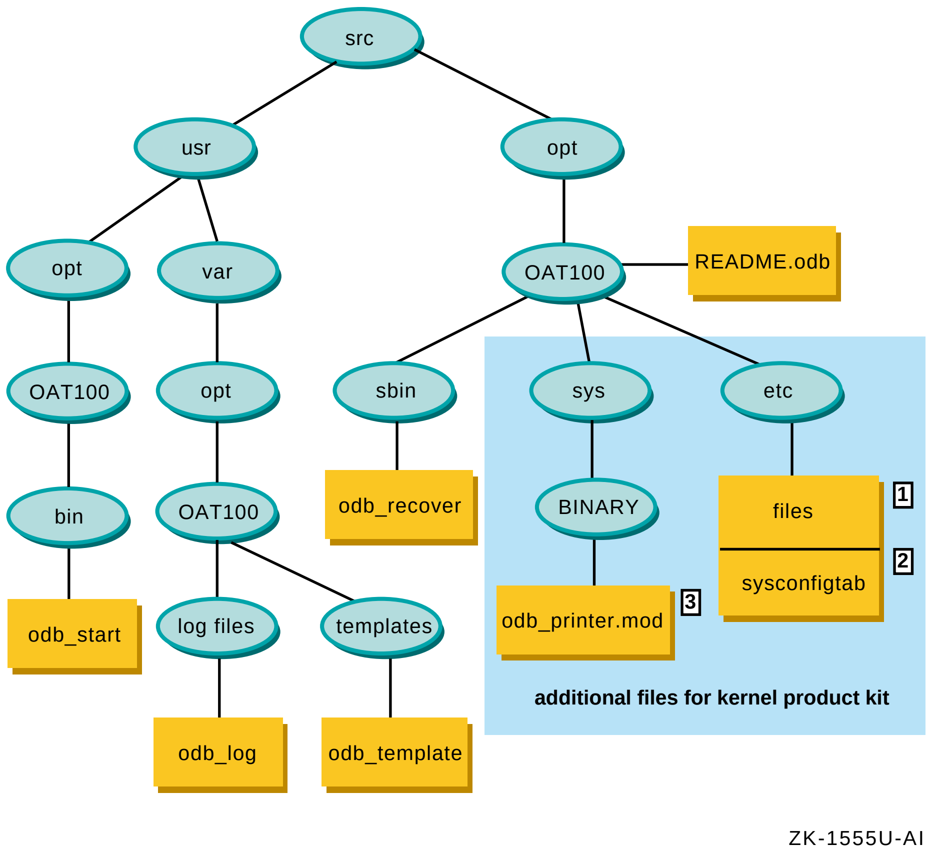
  Contain the source code for the device driver. You should include these files in a product directory, such as `/usr/opt/OAT100/src`, when the driver is statically configured and distributed in source form.

- The *device*`.mth` method files

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

Figure 5–3 shows the files that make up the ODB product kit source directory. Additional kernel product files are highlighted.

Figure 5–3: Kernel Product Source Directory



1 `files` — the `files` file fragment contains information about the location of the source code and modules associated with the driver, tags indicating when the driver is loaded into the kernel, and whether the source or binary form of the driver is supplied to the customer. Refer to Section 5.1 for more information about the `files` file fragment.

2 `sysconfigtab` — the `sysconfigtab` file fragment contains device special file information, bus option data information, and information on contiguous memory usage for statically and dynamically configured drivers. The `sysconfigtab` file fragment gets appended to the `/etc/sysconfigtab` database when the kernel product kit is installed. Refer to Section 5.1 for more information about the `sysconfigtab` file fragment.

3 `odb_graphics.mod` — the object module file containing the single binary module for the ODB kernel product. Refer to Section 5.1 for more information about the *`driver`*`.mod` file.

5.2 Producing Distribution Media

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

- Edit the `/etc/kitcap` file.
- Build the kit on the distribution media:
  - Use the `gentapes` utility to build a `tar` format kit on magnetic tape.
  - Use the `gendisk` utility to build a kit on disk media.

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

- `tar` format

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

- direct CD-ROM (DCD) format

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

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

  Installation time for kits in DCD format is slower than for kits in `tar` format. User product kits only should be produced in `tar` format if they require access to kit files before or during the installation.

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

- Magnetic tape

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

- Diskette

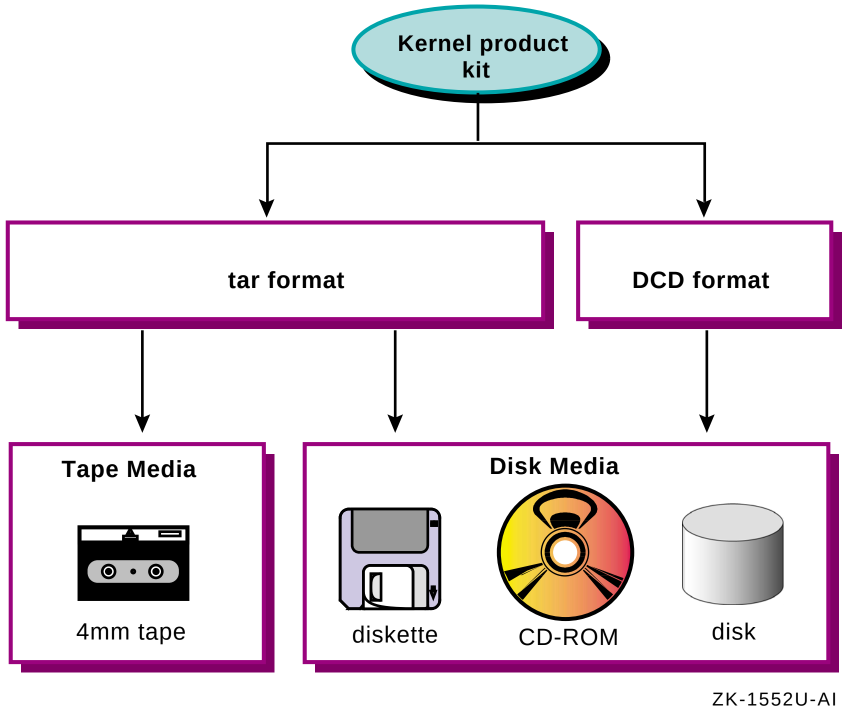
  Diskettes are a good media for testing purposes or for small products. The product must fit on a single diskette; it cannot span multiple diskettes. Use the `gendisk` utility to produce kits for diskette media.

- CD-ROM

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

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

Figure 5–4: Kernel Product Kit File Formats



5.2.1 Editing the /etc/kitcap File

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

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

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

When you add a record to the /etc/kitcap file, use the following conventions:

- Separate fields with colons (:).

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

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

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

5.2.1.1 Tape Media kitcap Record Format

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

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

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

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

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

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

Example 5–1 shows the record to be added to the /etc/kitcap file to produce the ODB kit on TK50 tapes:

Example 5–1: Sample /etc/kitcap Record for Magnetic Tape

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

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

5.2.1.2 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.
- The product description. This entry is taken from key file NAME field. You must replace any spaces with the underscore (_) character.
- The destination directory for the subsets on the disk media. Allows a hierarchical structure so you can put multiple products on one disk, or put parts of one product on different areas of the same disk.
- Name of the kit's output directory, where the gendisk utility can find the product subsets.
- The instctrl directory, relative to the output directory specification.
- The names of the subsets that make up the kit.

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

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

Example 5–2: Sample /etc/kitcap Record for CD-ROM or Diskette

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

Based on the information shown in Example 5–2, the gendisk utility places the kit on the c partition in the / (root) directory of the disk media. The product description is Orpheus_Document_Builder and the kit output directory is named /mykit/output. The kit consists of three subsets named OATODB100, OATODBTEMPS100, and OATODBKERNEL100.

5.2.2 Building a tar Format Kernel Product Kit on Magnetic Tape

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

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

- The -v option verifies the product media without writing it first. This assumes that you have already written the files to the product media. The -v option cannot be used together with the -w option.
- The -w option writes the product media without verification. The -w option cannot be used together with the -v option.
- If you do not use either the -v or -w option, the gentapes utility writes the tape, rewinds it, and then verifies the files in the kit descriptor.
- The optional *hostname* argument is the name of a remote TCP/IP network machine that contains the /etc/kitcap file. 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 the local system. You can use NFS file sharing to mount the kit files remotely on a system with the required disk drive.
- The mandatory *product-code* argument is a user-defined code that describes the product. It should match the product name specified in the kitcap record. This is a concatenation of the NAME and VERS fields of the key file, such as OAT100. The gendisk utility searches for *product-code* in the /etc/kitcap file on *hostname* and uses it to create the media.
- The mandatory *special* argument is the name of the device special file for the disk device, such as /dev/ntape/dat.

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

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

5.2.3 Building a Kernel Product Kit on Disk Media

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

Note

The `gendisk` utility supports diskettes but does not 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.

Use the following syntax for the `gendisk` command:

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

- The `-v` option verifies the product media without writing it first. This assumes that you have already written the files to the product media. The `-v` option cannot be used together with the `-w` option.
- The `-w` option writes the product media without verification. The `-w` option cannot be used together with the `-v` option.
- If you do not use either the `-v` or `-w` option, the `gendisk` utility writes and then verifies the files in the kit descriptor. This default behavior is the same as if you use both the `-v` and `-w` options together.
- The `-d` option specifies that the kit be produced in DCD format. If used, the `-d` option must be after any `-v` or `-w` options on the command line. If you do not use this option, the kit is produced in `tar` format.
- The optional *hostname* argument is the name of a remote TCP/IP network machine that contains the `/etc/kitcap` file. 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 the local system. You can use NFS file sharing to mount the kit files remotely on a system with the required disk drive.
- The mandatory *product-code* argument is a user-defined code that describes the product. It should match the product name specified in the `kitcap` record. This is a concatenation of the `NAME` and `VERS` fields of the key file, such as `OAT100`. The `gendisk` utility searches for *product-code* in the `/etc/kitcap` file on *hostname* and uses it to create the media.

- The mandatory *special* argument is the name of the device special file for the disk device, such as /dev/rdisk/dsk1a. The actual partition that the utility will write the files on is defined in the kitcap file entry for the *product-code* being used, so it makes no difference what partition is appended to the special device file used on the command line.

To prepare a kit on disk for a kernel product, use the gendisk utility without the -d option. Specify the product name and the device special file name. In Example 5–3, the command creates a kit in tar format for the ODB product on the c partition of the disk named dsk0:

Example 5–3: Sample gendisk Command for ODB Kernel Product in tar Format

```
% gendisk OAT100 /dev/rdisk/dsk0c
```

5.3 Testing the Distribution Media

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

Run the following tests to test a kernel product kit:

1. Use the setld utility to verify that the subsets have been built correctly and that the files get installed into the correct locations on the target system.
2. Use the ris utility to add a kernel product kit into a RIS area and verify that the correct files are present on the kit. Then, register the client system to the RIS area and start a full installation on the client system.

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

5.3.1 Testing a Kernel Product Kit with the setld Utility

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

For example, if the odb_graphics driver was distributed on CD-ROM, you would perform the following procedure to install the kit:

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 `cdrom0`, you would enter the following command:

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

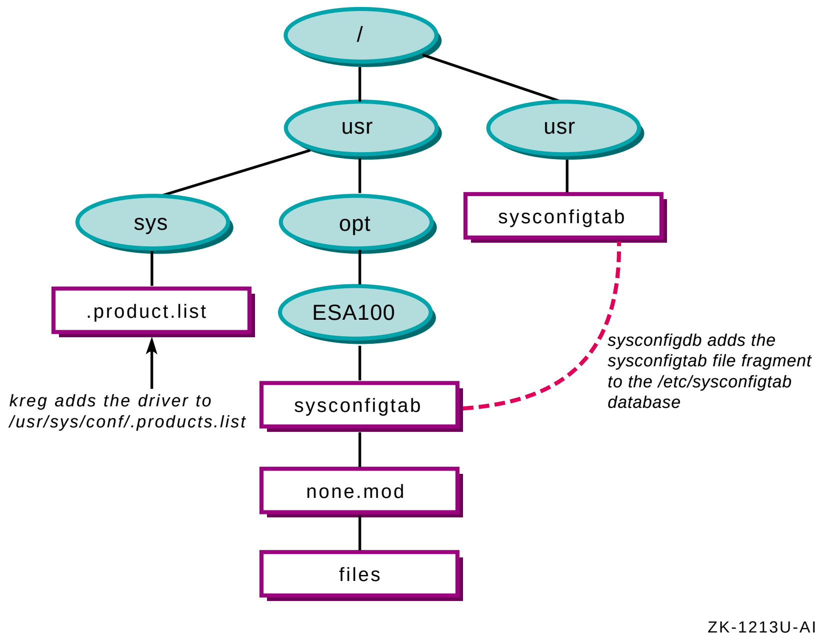
4. Install the ODB subsets with the `setld -l` command. You see output similar to the following:

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

*** Enter subset selections ***

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

      * Document Builder Kernel Support
      * Document Builder Tools

The subsets listed below are optional:

 - Other:
     1) Document Builder Templates

Or you may choose one of the following options:

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

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

Enter your choices or press RETURN to redisplay menus.

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

You are installing the following mandatory subsets:

        Document Builder Kernel Support
        Document Builder Tools

You are installing the following optional subsets:

 - Other:
        Document Builder Templates

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

Is this correct? (y/n): y

Checking file system space required to install selected subsets:

File system space checked OK.

3 subset(s) will be installed.

Loading subset 1 of 3 ...
```

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

Loading subset 2 of 3 ...

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

Loading subset 3 of 3 ...

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

3 of 3 subset(s) installed successfully.

Configuring "Document Builder Tools" (OATODB100)

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

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

Configuring "Document Builder Templates" (OATODBTEMPS100)

Configuring "Document Builder Kernel Support" (OATODBKERNEL100)

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

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

Type the number of your choice []: 1

*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***

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

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

*** PERFORMING KERNEL BUILD ***
        Working....Fri May 14 14:49:25 EDT 1999

The new kernel is /sys/TEST01/vmunix

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

The `setld` utility displays prompts and messages to guide you through the process of selecting the subsets you want to install. After it loads the subsets onto the system, `setld` invokes the subset control program to statically or dynamically configure the driver. Figure 5–5 shows the steps the subset control program takes to statically configure the driver.

Figure 5–5: Static Configuration of a Driver


/
usr
usr
sys
opt
sysconfigtab
.product.list
ESA100
kreg adds the driver to /usr/sys/conf/.products.list
sysconfigtab
sysconfigdb adds the sysconfigtab file fragment to the /etc/sysconfigtab database
none.mod
files
ZK-1213U-AI

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

```
# umount /cdrom
```

6. Restart the system with the new kernel if the product was statically configured:

```
# /usr/sbin/shutdown –r now
```

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

7. Verify that the installed product functions correctly.

8. Install the ODB subsets with the `setld -d` command. You see output similar to the following:

```
# setld -d OATODB100 OATODBTEMPS100 OATODBKERNEL100

Deleting "Document Builder Templates" (OATODBTEMPS100).

*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***

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

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

*** PERFORMING KERNEL BUILD ***
        Working....Fri May 14 14:55:31 EDT 1999

The new kernel is /sys/TEST01/vmunix

The /sys/TEST01/vmunix kernel has been
```

```
moved to /vmunix and the changes will take effect
the next time the system is rebooted.

Deleting "Document Builder Kernel Support" (OATODBKERNEL100).

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

9. Restart the system to ensure that it reboots with the new kernel after the product is removed:

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

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

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

5.3.2 Testing a Kernel Product Kit in a RIS Area

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

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

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

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

2. The RIS Utility Main Menu is displayed. Choose the `Install software products` option by entering **i** at the prompt:

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

   *** RIS Utility Main Menu ***

   Choices without key letters are not available.

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

   Enter your choice: i
   ```

3. The RIS Installation menu displays the installation options:

   ```
   RIS Software Installation Menu:

       1)  Install software into a new area
       2)  Add software into an existing area
       3)  Return to previous menu
   ```

```
Enter your choice:
```

4. Depending on your test environment, select option 1, `Install software into a new area` or option 2, `Add software into an existing area`.
5. Install the software as described in *Sharing Software on a Local Area Network*.

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

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

```
# /usr/sbin/ris

*** RIS Utility Main Menu ***

Choices without key letters are not available.

    a)  ADD a client
    d)  DELETE software products
    i)  INSTALL software products
     )  LIST registered clients
     )  MODIFY a client
     )  REMOVE a client
    s)  SHOW software products in remote installation
        environments
    x)  EXIT
Enter your choice: a
```

2. Enter the client information requested by the prompts as described in *Sharing Software on a Local Area Network*.
3. When the client is registered to the RIS server, log in to the client as superuser or `root`.
4. Use the `setld` utility to install the product subsets from the RIS area. For example, if the RIS server was named `test01`, you would enter the following command:

```
# setld –l test01:
```

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

A
Creating a Consolidated Firmware CD-ROM

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

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

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

A.1 Build Instructions

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

Note

The examples in this appendix use the C shell.

A.1.1 Prepare for the Build Session

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

1. Use the `disklabel` utility to set up a 635 Mb partition on a spare disk, starting at block 0, with a size of 1300480 512–byte blocks and a

file system type of `unused`. Create a mount point for this partition (such as `/cdimage`) to use later.

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

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

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

Note

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

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

```
% mount -r /dev/disk/cdrom4c /mnt
% cd /mnt
% tar cf /spare/os_copy.tar .
% cd /
% umount /mnt
```

A.1.2 Build the Consolidated Firmware CD-ROM

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

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

Note

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

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

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

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

   a. Mount the *Alpha Systems Firmware CD-ROM* to a temporary mount point such as `/mnt`. For example, using `/dev/disk/cdrom4c` as the CD-ROM drive:

      ```
      % mount -t cdfs -r /dev/disk/cdrom4c /mnt
      ```

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

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

      Caution

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

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

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

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

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

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

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

      Caution

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

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

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

e. Unmount and remove the firmware CD-ROM.

```
% umount /mnt
```

Note

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

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

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

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

4. Use the `disklabel`command to insert a label into the file generated in Step 3.

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

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

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

A.2 Sample Build Session

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

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

Note

The examples in this appendix use the C shell.

A.2.1 Preparing for the Build Session

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

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

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

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

```
% mount -r /dev/disk/cdrom4c /mnt
% cd /mnt
% tar cf /spare/os_copy.tar .
% cd /
% umount /mnt
```

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

A.2.2 Building the Consolidated Firmware CD-ROM

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

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

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

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

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

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

   The following example uses the same firmware images as Step 2d of Section A.1.2:

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

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

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

   Output is similar to the following:

```
Using OSFMANWO.000;1 for \
/cdimage/ALPHA/BASE/instctrl/OSFMANWOS500.scp \
(OSFMANWOP500.scp)
 Using OSFMANWO.001;1 for \
/cdimage/ALPHA/BASE/instctrl/OSFMANWOS500.inv \
 (OSFMANWOP500.inv)
 Using OSFMANWO.002;1 for \
 /cdimage/ALPHA/BASE/instctrl/OSFMANWOS500.ctrl \
 (OSFMANWOP500.ctrl)
                .
                .
                .
 Using PROCFS_V.000;1 for \
 /cdimage/usr/sys/procfs/procfs_vnops_stubs.c \
(procfs_vfsops_stubs.c)
   3.92% done, estimate finish Fri Oct 22 15:36:59
   5.87% done, estimate finish Fri Oct 22 15:39:24
                .
                .
                .
  99.74% done, estimate finish Fri Oct 22 15:41:52
 Total extents actually written = 255673
 Total translation table size: 0
 Total rockridge attributes bytes: 2066594
 Total directory bytes: 4239360
 Path table size(bytes): 10130
 Max brk space used b9ec60
 255673 extents written (499 Mb)
```

Note

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

5. Enter the following commands:

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

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

Glossary

This glossary defines terms used in this document.

A

attribute-value pair

In a product kit's key file, attribute-value pairs specify the names and values of the attributes of the kit, such as the name and version of the product. Attribute-value pairs control how the `kits` utility builds the kit and how the `setld` utility installs it.

B

Backus-Naur form

A conventional notation for describing context-free grammars, commonly used for defining syntax in computer languages. It is named for John Backus, developer of FORTRAN, and Peter Naur, developer of ALGOL. The term **BNF** is often used to refer to grammar specifications based on this form.

backward link

A backward link is a symbolic link from the directories in a layered product area to files in the standard hierarchy. The subset control program for a product creates backward links during installation.

boot utility

The `boot` utility performs the initial installation and bootstrap of the operating system. It is invoked from the `>>>` console prompt.

C

control files

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

D

Dataless Management Services

See *DMS*

DMS

Dataless Management Services. A service where a server maintains the `root`, `/usr`, and `/var` file systems for client computer systems connected to the server by means of a local area network (LAN).

data hierarchy

In the kit-building directory structure, the data hierarchy contains the files that direct the `setld` utility in making subsets for the kit, such as the master inventory and key files. An `scps` subdirectory contains subset control programs written by the kit developer.

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*, *subset dependency*, *postfix*

distribution media

The distribution media for a product kit may be diskette, CD-ROM, or tape. A hard disk is sometimes referred to as a distribution media because it is used as the master copy for a CD-ROM kit.

E

/etc/sysconfigtab database

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

See also *sysconfigdb utility*

F

forward link

A forward link is a symbolic link that connects a product file in the `/opt`, `/usr/opt`, or `/var/opt` directory to a standard UNIX directory, such as `/usr/bin`. Forward links allow layered products to be installed in a central location (the `opt` directories) and still be accessible to users through the standard directory structure.

K

kernel

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

kernel product

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

See also *layered product*, *user product*

key file

A key file identifies the product that the kit represents. You create this file in the `data` directory before running the `kits` utility.

kit

A kit is a collection of files and directories that represent one or more layered products. It is the standard mechanism by which layered product modifications are delivered and maintained on the operating system.

See also *layered product*

kits utility

The `kits` utility creates subsets according to the specifications you define in the master inventory file and key file. *See also* **key file**, **master inventory file**, *and* **subset**.

L

layered product

A layered product is an optional software product designed to be installed as an added feature of the operating system.

See also *kernel product*, *user product*

locking

In products installed by the `setld` utility, locking inserts a subset name in the lock file of another subset. Any attempt to remove the latter subset warns the user of the dependency. The user can choose whether to remove the subset in spite of the dependency.

M

master inventory file

A master inventory file lists all the product files and the subsets in which they belong. You create this file in the `data` directory by running the `newinv` utility. The file must exist before you can create the product subsets.

See also *newinv utility, subset*

N

Network File System

See *NFS*

newinv utility

The `newinv` utility creates the master inventory file from the list of files in the current working directory. The list does not contain all the information needed in the master inventory file. You must edit this file to include information about the subsets to which the files belong.

See also *master inventory file*

NFS

Network File System, an open operating system that allows all network users to access shared files stored on computers of different types. Users can manipulate shared files as if they were stored locally on the user's own hard disk.

O

output hierarchy

The output hierarchy contains the result of the kit-building process, including the subsets that make up the kit and installation control files to direct the `setld` utility during the installation of the product.

P

postfix

A form of logical expression where the operators follow the operands, rather than being placed between them. Also know as **reverse Polish notation**, or RPN.

R

RIS

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

Remote Installation Services

See *RIS*

S

SCP

Subset control program. A program written by the kit developer to perform installation operations that the `setld` utility would not otherwise perform. The `setld` utility invokes the subset control program several times during the installation of the kit.

setld utility

The `setld` utility allows the transfer of the contents of a layered product kit to a customer's system.

source hierarchy

In the kit-building directory structure, the source hierarchy contains the files that make up the product. These files are grouped into subsets by the `kits` utility.

subset

A subset is the smallest installable component of a product kit for the `setld` utility. It contains files of any type, usually related in some way.

subset control program

See *SCP*

subset dependency

A subset dependency is the condition under which a given subset requires the presence (or absence) of other subsets in order to function properly.

See also *dependency expression, locking*

sysconfigdb utility

The `sysconfigdb` utility is a system management tool that maintains the `sysconfigtab` database.

See also */etc/sysconfigtab database*

U

user product

A user product is a layered product that runs in user space. Commands, utilities, and user applications fall into this category.

See also *kernel product*, *layered product*

Index

How to Order Tru64 UNIX Documentation

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

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

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

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

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

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

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

Reader's Comments

Tru64 UNIX
Guide to Preparing Product Kits
AA-RH9SA-TE

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

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

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

Please rate this manual:

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

Please list errors you have found in this manual:

Page	Description
________	____
________	____
________	____
________	____

Additional comments or suggestions to improve this manual:

____
____
____
____

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

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

Do Not Cut or Tear - Fold Here and Tape

COMPAQ

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPAQ COMPUTER CORPORATION
UBPG PUBLICATIONS MANAGER
ZKO3-3/Y32
110 SPIT BROOK RD
NASHUA NH 03062-9987

Do Not Cut or Tear - Fold Here

Cut on Dotted Line