ULTRIX

digital

Guide to Preparing Software for Distribution on ULTRIX Systems

Order Number: AA–MG62B–TE

ULTRIX

Guide to Preparing Software for Distribution on ULTRIX Systems

Order Number: AA-MG62B-TE

June 1990

Product Version: ULTRIX Version 4.0 or higher

digital equipment corporation
maynard, massachusetts

Contents

About This Manual

1 Overview

2 How setld Utilities Work

3 Files Used by setld

7 How setld Options Work

A Sample make Files

Figures

Tables

About This Manual

This guide describes how to prepare software products for distribution on ULTRIX systems.

Audience

This guide is intended for applications programmers who are preparing a software product for installation on ULTRIX systems. The programmer is presumed to understand ULTRIX and know how to program in an ULTRIX environment. Knowledge of the C programming language is helpful, but not necessary.

Organization

This guide consists of seven chapters and one appendix.

Chapter 1 Overview

Presents an overview of the software preparation requirements. The chapter also lists the components you need to produce `setld`-compatible distribution kits.

Chapter 2 How `setld` Utilities Work

Describes the operations that the major `setld` utilities perform to produce software subsets.

Chapter 3 Files Used by `setld`

Describes the files that the `setld` utility uses during subset installation.

Chapter 4 Writing `setld` Subset Control Programs

Describes how to write Subset Control Programs (SCP) to install and manage software subsets.

Chapter 5 Producing `setld`-Compatible Kits

Describes how to produce `setld`-compatible software distribution kits.

Chapter 6 Reproducing `setld`-Compatible Kits

Describes how to how to reproduce `setld`-compatible software distribution kits.

Chapter 7 How `setld` Options Work

Describes the operations that each `setld` option performs when specified on a `setld` command line.

Appendix A	Sample make Files Contains sample `make` files for managing the transfer of a product from source directories to the input directory from which the files will be processed by the `setld` utility.

Related Documentation

You should have read the relevant sections of:

- *ULTRIX Reference Pages*

  You should read the descriptions of any commands referred to in this guide with which you are not familiar.

Conventions

>>> CPU*nn*>>	The console subsystem prompt is two right angle brackets on RISC systems, or three right angle brackets on VAX systems. On a system with more than one central processing unit (CPU), the prompt displays two numbers: the number of the CPU, and the number of the processor slot containing the board for that CPU.
`user input`	This bold typeface is used in interactive examples to indicate typed user input.
`system output`	This typeface is used in interactive examples to indicate system output and also in code examples and other screen displays. In text, this typeface is used to indicate the exact name of a command, option, partition, pathname, directory, or file.
UPPERCASE lowercase	The ULTRIX system differentiates between lowercase and uppercase characters. Literal strings that appear in text, examples, syntax descriptions, and function definitions must be typed exactly as shown.
. . .	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
`cat(1)`	Cross-references to the *ULTRIX Reference Pages* include the appropriate section number in parentheses. For example, a reference to `cat(1)` indicates that you can find the material on the `cat` command in Section 1 of the reference pages.
Mbyte	Throughout the text, the abbreviation Mbyte is used for megabyte. One megabyte equals 1,048,576 bytes.

Overview 1

This guide describes how to prepare software products for distribution on ULTRIX systems. The guide describes ULTRIX utility programs that produce software distribution kits compatible with the `setld` utility. The `setld`–compatible kits contain software products that can be installed on ULTRIX systems. The guide also describes the following:

- How to produce a software distribution kit for a product to be installed and managed using the `setld` utility.
- How to reproduce a `setld`–compatible software distribution kit.
- The operations that `setld` options perform.

1.1 Introduction

The software for each product distributed for ULTRIX systems consists of a hierarchical group of files and directories. The developer producing the software product determines how the files and directories are grouped within the hierarchy.

The `setld` utility and related programs are production, distribution, and management tools to use with ULTRIX software. You can use these tools to add, subtract, and combine software products. The `setld` installation process preserves the integrity of each product's hierarchy when it is transferred from a development system to a customer system.

The kitting process includes grouping the component files for a software product into subsets. The `kits` utility imposes a format compatible with the `setld` utility on the files that make up the product. The utilities also generate control information, for example, the location of each subset on the media.

The subsets are transferred to distribution media. The control information for the product is located in one place and governs the transfer of multiple subsets.

Extraction, validation, and configuration take place at the destination system using the ULTRIX utilities, `setld`, `fverify`, the Remote Installation Service (`ris`), and Diskless Management Services (`dms`). The integrity of the software product is preserved as it is transferred to the destination system.

1.2 Benefits of setld Utility Use

Using the `setld` utility to install and manage software products on ULTRIX systems ensures the following:

- Installation security
- Flexibility

- Uniformity
- Media support

1.2.1 Installation Security

When you use the `setld` utility to install a software product, each subset is verified immediately after transfer. Each subset is recoverable if you want to reinstall it if it is damaged or deleted.

1.2.2 Flexibility

The `setld` utility lets the user choose subsets at installation. Users can also use the `setld` utility to delete subsets, then reinstall them as needed.

This means that users can customize their systems to perform specific types of activities. For example, after installing the mandatory ULTRIX subsets, a user might load optional subsets and products to tailor the system to serve one or more of the following purposes:

- Communications
- Documentation
- Computer Aided Software Engineering (CASE)
- Commercial database operations

1.2.3 Uniformity

The `setld` utility is an integral part of the ULTRIX installation architecture. Therefore, producing products in the form of kits that are compatible with the `setld` utility enhances compatibility with any future ULTRIX installation architecture.

In addition, kits that are compatible with the `setld` utility can be loaded on a server machine for installation over the network using the `ris` utility and for installing into a diskless environment using the `dms` utility.

1.2.4 Media Support

You can use any of the following devices to install a `setld`–compatible software product from the distribution media specified:

- An arbitrary, mountable file system on any supported data disk, for example, a file system found on an RA60 disk pack or a CDROM optical disc
- A TK50 tape on a TK50 or a TK70 tape drive
- An MT9 tape of arbitrary density

1.3 Production Requirements for setld-Compatible Kits

You need the following components to produce your own `setld` distribution kits:

- A master directory hierarchy containing the files that make up the product
- A `key` file describing product subsets in the master directory hierarchy
- An optional subset control program (SCP) for each software subset included in the product

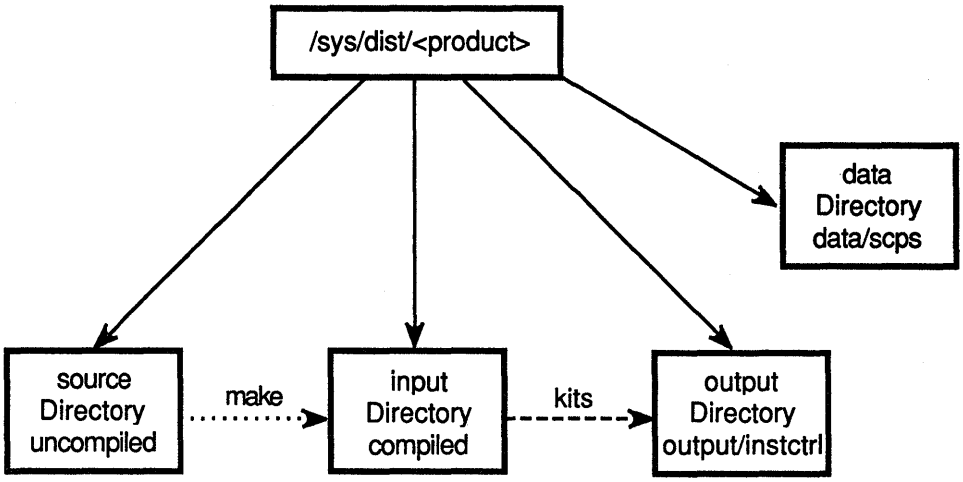
1.4 Production Process for setld-Compatible Kits

The following steps, performed in the order indicated, establish the `setld`–compatible environment for your product. This procedure assumes that the uncompiled files that make up your product reside in an existing directory structure, called the source directory:

1. Generate the master directory hierarchy for your product. This might be done by using `make` files to transfer files from the source directory to an input directory.
2. Create a `data` directory to contain the operating environment for kitting your product.
3. Create an inventory of the master hierarchy.
4. Create the `key` file.
5. Create the subdirectory `scp` under the `data` directory to contain SCP files.
6. Create the SCPs.
7. Create an output directory structure to contain output files compatible with the `setld` utility.
8. Create the `instctrl` subdirectory under the output directory to contain control files.
9. Run the `kits` utility to transfer images into the output directory.
10. Generate individual pieces of media.

Figure 1-1 illustrates a sample directory layout for a `setld`–compatible product environment. In the figure, `make` files are used to generate the master directory hierarchy. Then, the `kits` utility transfers the images from the input directory to the output directory.

Figure 1-1: Sample setld–Compatible Product Environment Layout



ZK-0165U-R

Chapter 4 describes how to write SCPs. Chapter 5 describes how to set up a `setld`–compatible environment for your product and how to create a distribution kit compatible with the `setld` utility.

How setld Utilities Work 2

This chapter describes the operations that the major `setld`-related utilities, `newinv` and `kits`, perform to produce subsets.

2.1 The newinv Utility

This section describes how the `newinv` utility processes a master inventory input file. The input file is empty if the software product is being processed for a kit for the first time. The input file is the existing master inventory file if the kit for the product is being updated. The file UWS400.mi is used as an example throughout this section.

Master inventory input filenames use the following convention:

<Product_code><Version_code>.mi

For example:

```
UWS400.mi
```

The `newinv` command line has the following syntax:

```
newinv <filename.mi> <directory>
```

For example:

```
newinv UWS400.mi ../build
```

The output of the `newinv` utility is a file containing the master inventory of a software product.

Each record in the master inventory file contains three fields which are separated by tabs. Each record ends with a newline.

Table 2-1 lists the fields that make up a master inventory record with a description of each field.

Table 2-1: Master Inventory File

Field	Description
Flags	An unsigned integer. The low two bits can be set to indicate the following: p for precedence: the user's copy of the file supercedes the developer's copy. The p bit is usually set for files like rc, rc.local, and etc/ttys. v for volativity: changes to the user's copy of the file are expected. The v bit is usually set for files like usr/spool/mqueue/syslog and usr/adm/aculog. The values of the bit settings are: neither bit = 0, p only = 1, v only = 2, p + v = 3.
Pathname	The dot-relative (./) pathname of the file described by this record.
Subset name	The name of the subset containing the file.

The example that follows shows a portion of UWS400.mi, the master inventory file for the ULTRIX Worksystem Software Version 4.0 product:

```
0     ./usr/bin/xcons UWSX11400
0     ./usr/bin/xedit UWSX11400
0     ./usr/bin/xfd   UWSX11400
0     ./usr/bin/xget  ZOMBIE
0     ./usr/bin/xhost UWSX11400
0     ./usr/bin/xload UWSX11400
0     ./usr/bin/xlsfonts      UWSX11400
0     ./usr/bin/xmh   UWSX11400
0     ./usr/bin/xrdb  UWSX11400
0     ./usr/bin/xrefresh      UWSX11400
0     ./usr/bin/xsend ZOMBIE
0     ./usr/bin/xset  UWSX11400
0     ./usr/bin/xsetroot      UWSX11400
0     ./usr/bin/xterm UWSX11400
0     ./usr/bin/xwininfo      UWSX11400
```

The newinv utility performs the following operations when it generates a master inventory file:

- Creates a backup file, UWS400.mi.bkp.
- Performs a sort operation on the UWS400.mi file.
- Performs a find operation on <directory> to get file and directory names from the input directory tree and directs the output to a temporary file.
- Performs a sort operation on the temporary file.
- Performs a relational join operation that produces the following three groups of records:
  - New records containing the pathname only

- Records continuing from the previous inventory, if any
- Records included in the previous inventory that represent files missing from the input hierarchy, if any.

- Drops any group for which there are no records.
- Places the user in an editor, either the editor specified by the setting of the environment variable `EDITOR` or in the default editor `/usr/ucb/vi`.
- Prompts the user to delete records from the group representing files missing from the input hierarchy if they are no longer part of the subset.
- Merges with the continuing records any records remaining in the group representing files missing from the input hierarchy after editing.
- Prompts the user to edit new records by adding the flag and subset fields.
- Merges the new records with the continuing records.
- Performs a `sort` operation on the edited file, which now matches the input hierarchy, and generates a new `UWS400.mi` file.

Refer to `environ`(7) in the *ULTRIX Reference Pages* for information about environment variables.

2.2 The kits Utility

This section describes how the `kits` utility produces subset images, inventories, and control files from input files that have been transferred from the source directory for the product. The `kits` utility generates data files that make up the media master in the output directory.

The following example shows the `kits` command line syntax with a sample command:

```
kits <filename.k> ../input ../output
kits UWS400.k ../input ../output
```

The data files the `kits` utility generates and uses, the contents of each, and a description of its purpose are listed in Table 2-2.

Table 2-2: kits Utility Data Files

File	Contents	Description
SPACE	10240-byte dummy file	Created by user; reserved by Digital
instctrl/*	Directory	Generated; contains `setld` control information
INSTCTRL	tar image	Generated; contains `setld` control information
<P-code>.image	Record for each subset	Generated; size and checksum information for each subset

The `kits` utility performs the following operations when it generates subset images, inventories, and control files:

- Creates the `output/instctrl` directory if none exists.
- Performs the following operations for each subset:
  - Generates the subset inventory file, which has a `.inv` extension, and places it in the `output/instctrl` directory.
  - Generates the subset control file, which has a `.ctrl` extension, and places it in the `output/instctrl` directory.
  - Generates the subset image and places it in the `output` directory.
  - Generates an empty `.scp` file if none exists in the `./scps` directory.
  - Transfers the `.scp` file to the `output/instctrl` directory.
  - Computes the subset image size and checksum and creates a record in the `.image` file.
- Creates the `output/INSTCTRL` file from the contents of the `output/instctrl` directory.

Files Used by setld 3

This chapter describes files which the `setld` utility uses during subset installation. Some of the files are generated by the `kits` utility and some are generated by the `setld` utility.

3.1 File Format

The files that the `setld` utility uses are variable length ASCII text files. The field separator is a tab (CTRL/I), and the record separator is a newline (CTRL/J).

3.2 File Types

Table 3-1 shows the types of files that the `setld` utility uses, with the extension and contents of each file.

Table 3-1: Files Used by the setld Utility

File Type	Extension	Contents
Control	.ctrl	Subset control information from key file, media information, lists of dependencies, flags, and descriptions.
Inventory	.inv	File attribute information, size, and dot–relative path name of each file in subset.
Lock	.lk	Marks subset as installed, lists dependent subsets. Generated by `setld` only if subset installed successfully.
Lock	.dw	Empty. Generated by `setld` only if subset failed to install.
Image data	.image	Record containing size and checksum of each subset in kit.
Compression data	.comp	Empty. Generated only if the subset images for the product are compressed.
SCP	.scp	Subset control program.

The sections that follow contain descriptions of all but the SCP files. Subset control programs are described in Chapter 4.

3.2.1 Control Files

The `setld` utility uses control files generated by the `kits` utility to get descriptive information about subsets.

There is a control file for each subset and it contains the following information from the key file for the subset:

- Product name
- Subset description
- RA60 disk volume number
- Tape volume number and location on tape
- List of dependencies
- Subset control bit flags

The entries can be in any order, but all of them must be included in each control file.

The following example shows `UWSX11400.ctrl`, the control file generated for the UWSX11400 subset, in the order produced by the `kits` utility:

```
NAME='ULTRIX Worksystem Software V4.0 UWSX11400'
DESC='X11/DECwindows User Environment'
NVOLS=1:28
MTLOC=1:8
DEPS="ULTINET400"
FLAGS=0
```

3.2.2 Inventory Files

The `setld` utility uses inventory files generated by the `kits` utility to verify the contents of subsets when they are installed.

There is an inventory file for each subset. The inventory file contains a record describing each file included in the subset.

Each record in the inventory file contains 12 fields. Each field and its contents are described in Table 3-2.

Table 3-2: Inventory File Record Contents

Field Number	Name	Contents
1	Flags	An unsigned integer. The low two bits can be set to indicate the following: `p` for precedence: the user's copy of the file supercedes the developer's copy. The `p` bit is usually set for files like `rc`, `rc.local`, and `etc/ttys`. `v` for volativity: changes to the user's copy of the file are expected. The `v` bit is usually set for files like `usr/spool/mqueue/syslog` and `usr/adm/aculog`. The values of the bit settings are: neither bit = `0`, `p` only = 1, `v` only = 2, `p + v` = 3.
2	Size	Number of bytes.
3	Sum	Checksum (mod 2^16).
4	uid	User id.
5	gid	Group id.
6	Mode	Octal representation of mode.
7	Date	Last modification date.
8	Revision	Revision code of the product that includes the file.
9	Type	A code describing the file that can be: d - directory containing one or more files c - character device b - block device p - named pipe (FIFO) f - regular file l - hard link s - symbolic link
10	Path	The dot-relative (./) pathname.
11	Link-to	For file types `l` and `s`, the path to which the file is linked. For types `c` and `b`, an integer representing the major and minor numbers for the device. For all other file types, `none`.
12	Subset name	The name of the subset containing the file.

The following example shows a portion of `UWSX11400.inv`, the inventory file for the UWSX11400 subset:

```
0     1864    24893   0       0       100644  4/4/88  400     f
./usr/sk
el/.XtActions   none    UWSX11400
0     1225    15886   0       0       100644  4/20/88 400     f
./usr/sk
el/.Xdefaults   none    UWSX11400
0     512     00000   0       0        40755  5/17/88 400     d
./usr/li
b/lpdfilters    none    UWSX11400
ddd     28713   34313   0       0       100755  5/4/88  400     f
./usr/li
b/dwcmsg        none    UWSX11400
0     7680    00000   0       10       40755  5/17/88 400     d
./usr/li
b/decwin/fonts  none    UWSX11400
0     512     00000   0       10       40755  5/17/88 400     d
./usr/li
b/decwin        none    UWSX11400
0     3826    03999   0       0       100644  4/4/88  400     f
./usr/li
b/XErrorDB      none    UWSX11400
0     3119    54414   0       0       100755  4/4/88  400     f
./usr/li
b/X11/rgb.txt   none    UWSX11400
0     4096    59456   0       0       100755  4/4/88  400     f
./usr/li
b/X11/rgb.pag   none    UWSX11400
0     4096    00014   0       0       100755  4/4/88  400     f
./usr/li
b/X11/rgb.dir   none    UWSX11400
```

3.2.3 Lock Files

There are two types of lock files that the `setld` utility can generate. One, with a `.lk` extension, indicates a subset is installed. The second type of lock file, with a `.dw` extension, indicates a subset failed to install.

3.2.3.1 Subset-Installed Lock Files – A subset-installed lock file, that has a `.lk` extension, marks a subset as installed. It contains the names of any dependent subsets. Subset-installed lock files exist only for subsets that are currently installed. They appear only on the destination system.

Subset-installed lock files are removed when a subset is deleted. However, if there are dependent subsets, their names are displayed and the user is asked to confirm that deletion should take place.

The following example shows `ULTPGMR400.lk`, the subset-installed lock file for the ULTPGMR400 subset:

```
ULTINTL400
ULXF77400
```

3.2.3.2 Subset-Failed Lock Files – The presence of a subset-failed lock file, that has a `.dw` extension, indicates that the last attempt to install a subset failed. The file is empty.

3.2.4 Image Data Files

The `setld` utility uses image data files generated by the `kits` utility to verify kit images for a Remote Installation Service (`ris`) area installation.

Image data files contain one record for each subset in the kit.

Each record in the image data file contains 3 fields. Each field and its contents are described in Table 3-3.

Table 3-3: Image Data File Record Contents

Field	Contents
Checksum	Mod 2^16
Size	Total kbytes in subset
Subset name	Product code, subset mnemonic, version number

The following example shows `UWS400.image`, the image data file for the ULTRIX Worksystem Software kit:

```
33528  2720 ROOT
00444  6080 ULTBASE400
10570  2200 ULTBIN400
19953   910 ULTINET400
61885   320 ULTEXER400
61066   450 ULTNFS400
47131   240 ULTUMAIL400
25177  2220 ULTMH400
58658  4700 UWSX11400
52864   890 UWSFONT400
03166  1020 UWSFONT3D400
21560   630 ULTDCMT400
58793  1520 ULTPGMR400
57868   170 ULTINTL400
07452   330 ULTSCCS400
41240   880 UWSXDEV400
13489  1430 UWSXDEV3D400
43810   420 ULTPASCAL400
60226   410 ULTVAXC400
61839  1300 ULTMAN400
04788    70 ULTACCT400
29772   270 ULTCOMM400
14187   100 ULTBSC400
13299  1170 ULTMOP400
24884   490 ULTUUCP400
```

3.2.5 Compression Data File

The compression data file is generated by the `kits` utility only if the product is compressed when it is produced for processing by the `setld` utility. The file is empty.

Compression file names have the following syntax:

```
<Product_code><Version_code>.comp
```

For example:

```
UWS400.comp
```

Writing setld Subset Control Programs 4

This chapter describes how to write Subset Control Programs (SCP) to install and manage subsets.

If an SCP is not needed for a subset, the `kits` utility creates an empty SCP file for that subset.

Each SCP must be executable by the superuser. The mode set for SCPs should be `read only` for the world.

You can write SCPs in any programming language. An SCP is text to be interpreted by a command interpreter. For example:

```
bourne shell    #!/bin/sh #!/usr/ucp/lisp
korn shell      #!/bin/ksh
c shell         #!/bin/csh#!/bin/make
```

Note

Use of the c shell, `csh`, is not recommended.

SCPs run from the top level directory of the subset hierarchy. All file references must be dot-relative (./) to files located above the directory in which the SCP program runs.

4.1 SCP Format

SCP names have the following format:

<product_code><subset_mnemonic><version_code>.scp

For example, the SCP for the UWSX11400 subset has the following name:

`UWSX11400.scp`

4.2 SCP Environment

The `setld` utility executes SCPs which install and manage software as needed. The `setld` utility controls SCP action according to the setting of the environment variable, ACT.

The possible settings for the ACT variable are listed in Table 4-1.

Table 4-1: Possible ACT Settings

Setting	Description
C	Before deleting and after adding a subset to a diskless environment. Provides subset-specific preparation commands. The command line includes an argument describing the requested action, for example: INSTALL - node–specific installation configuration DELETE - clean up node–specific configuration information on deletion
PRE_D	Before deleting a subset.
POST_D	After deleting a subset.
PRE_L	Before loading a subset.
POST_L	After loading a subset.
M	Before presenting the subset menu.
PRE_U	Before updating a subset.
POST_U	After updating a subset.
V	Verify a subset installation.

Write your SCP programs to get the ACT tag from the environment. Then, set up, take down, or perform all generic configuring for the subset, depending on the value of the ACT tag.

In a shell program, `$ACT` is a defined variable. If there is a validation suite for the subset, place it in the SCP program so that the suite is activated when the ACT variable is set to V.

The following example shows the SCP for the UWSX11400 subset:

```
NUL=/dev/null

XCONSDATA="#xcons \"/usr/bin/xcons 10 ttyv0\" none on nomodem"

SM0DATA="#ttyv0 \"/usr/bin/xterm -L -sb -rv =80x24+195+275 unix:0\"
xterm on secure window=\"/usr/bin/Xsm 0\"
:0 \"/usr/bin/login -P /usr/bin/Xprompter -C /usr/bin/dxsession\" none
on secure window=\"/usr/bin/Xsm\""

QV0DATA="#ttyv0 \"/usr/bin/xterm -L -sb -rv =80x24+195+275 unix:0\"
xterm on secure window=\"/usr/bin/Xqvss 0\"
:0 \"/usr/bin/login -P /usr/bin/Xprompter -C /usr/bin/dxsession\" none
on secure window=\"/usr/bin/Xqvss\""

QD0DATA="#ttyv0 \"/usr/bin/xterm -L -sb -rv =80x24+195+275 unix:0\"
xterm on secure window=\"/usr/bin/Xqdss :0\"
:0 \"/usr/bin/login -P /usr/bin/Xprompter -C /usr/bin/dxsession\" none
on secure window=\"/usr/bin/Xqdss -bp #000080 c 70\""

QD1DATA="#ttyv1 \"/usr/bin/xterm -L -sb -rv =80x24+195+275 unix:1\"
xterm on secure window=\"/usr/bin/Xqdss :1\"
:1 \"/usr/bin/login -P /usr/bin/Xprompter -C /usr/bin/dxsession\" none
on secure window=\"/usr/bin/Xqdss -bp #000080 c 70\""

NL="

umask 022
case $ACT in
POST_[AL])
```

```
        # flat load, hit qv.o
        [ -f usr/sys/BINARY.vax/qv.o ] &&
                echo '
qv_def_scrn?W 2
$q

        ;;
C)
        # configure.
        case "$1" in
        INSTALL)
                case "`pwd`" in
/)      # straight install, read config data from dev/kmem
                        TYPE=`echo "ws_display_type/d" | adb /vmunix
/dev/kmem |
                                awk 'NR == 2 {print $2}'`
                        case "$TYPE" in
                        42 | 49 | 50 | 35)
                                UNITS=`echo "ws_display_units/d" |
                                        adb /vmunix /dev/kmem |
                                        awk 'NR == 2 {print $2}'`
                                ;;
                        *)      # no graphics device, bale out
                                        exit 0
                        esac
                        ;;
                *)      # dot relatively.
                        # GDEV and WS_UNITS are exported by DMS
                        TYPE=$GDEV
                        case "$WS_UNITS" in
                        "")     # not called from DMS, bale out.
                                exit 0
                                ;;
                        *)      # read UNITS as provided by DMS
                                UNITS="$WS_UNITS"
                                ;;
                                        esac
                        ;;
                esac

                # rip apart UNIT code to see which heads to set up.
                BIT0=`expr $UNITS % 2`
                UNITS=`expr $UNITS / 2`
                BIT1=`expr $UNITS % 2`
                case "$BIT0" in
                1)
                        # set up head 0
                        mv dev/ttypf dev/ttyv0 &
                        mv dev/ptypf dev/ptyv0 &
                        egrep -v "ttyv0" etc/ttys |
                                sed 's/^console/#console/' > tmp/ttys
                        case "$TYPE" in
                        35 | QV)
                                echo "$QV0DATA$NL$XCONSDATA" >> tmp/ttys
                               ;;
                        49 | SM)
                                echo "$SM0DATA$NL$XCONSDATA" >> tmp/ttys
                               ;;
                        42 | 50 | SG | QD*)
                                echo "$QD0DATA$NL$XCONSDATA" >> tmp/ttys
                              ;;
                        esac
                        mv tmp/ttys etc/ttys
                        ln -s ../tmp/X0 dev/X0 &
                        ;;
                esac
```

```
                case "$BIT1" in
                1)
                        # set up head 1 - QDSS only
                        wait
                        mv dev/ttype dev/ttyv1 &
                        mv dev/ptype dev/ptyv1 &
                        egrep -v "ttyv1" etc/ttys > tmp/ttys
                        echo "$QD1DATA" >> tmp/ttys
                        mv tmp/ttys etc/ttys &
                        ln -s ../tmp/X1 dev/X1 &
                        ;;
                esac
                  ;;
        DELETE)
                # put the pty's back and clean up the ttys file.
                [ -f dev/ttyv0 ] && mv dev/ttyv0 dev/ttypf &
                [ -f dev/ptyv0 ] && mv dev/ptyv0 dev/ptypf &
                [ -f dev/ttyv1 ] && mv dev/ttyv1 dev/ttype &
                [ -f dev/ptyv1 ] && mv dev/ptyv1 dev/ptyv1 &
                egrep -v "ttyv0|ttyv1" etc/ttys |
                        sed 's/^#console/console/' > tmp/ttys
                mv tmp/ttys etc/ttys
                ;;
        esac
                egrep -v "ttyv0|ttyv1" etc/ttys |
                        sed 's/^#console/console/' > tmp/ttys
                mv tmp/ttys etc/ttys
                ;;
        esac
        ;;
POST_D)
                ;;
esac

# let everybody die off...
wait
exit 0
```

Producing setld-Compatible Kits 5

This chapter describes how to produce a `setld`-compatible software distribution kit. The uncompiled files that make up your software product are assumed to exist in a directory structure.

The sections that follow tell you how to:

- Generate the master directory hierarchy.
- Create the `data` directory.
- Generate the master inventory.
- Create the `key` file.
- Create the SCPs.
- Create an output directory.
- Generate the kit images.
- Create the `SPACE` file.
- Build the `/etc/kitcap` data base.
- Make the distribution media.

The UWSX11400 subset is used as the main example throughout this chapter. The ULTINET400 subset is used as an example of a dependent subset.

5.1 Generating the Master Directory Hierarchy

This section describes how to generate the master directory hierarchy by transferring files from the source directory to an input directory. The source directory contains all of the uncompiled files and directories that make up the software product. These files are compiled and transferred to the input directory using `make` files.

5.1.1 Create the Input Directory Tree

Follow this procedure to create the input directory tree:

1. Create an input directory structure that mirrors the directory structure you require on the destination system. The top of this hierarchy is `/`.
2. Do not include any files in your input directory structure that match existing ULTRIX files.

5.1.2 Manage the Transfer

One method for managing the transfer is to use `make` files to create file attributes and to create directories on the input directory tree. If you use `make`, write the files to compile the files in your source directory and to transfer the compiled files to the

appropriate place in the input directory hierarchy. Write your make files so that they set owner, group, and mode permissions for each file.

Follow this procedure if you write make files:

1. Create a make file in each source directory.
2. Write each make file to process the contents of the directory in which it runs, and to be responsible for directories immediately below the directory in which it runs.
3. Use the install command in the make files to install each file in the source directory to the desired place in the input directory. Refer to the *ULTRIX Reference Pages* for information about the install(1) command.
4. Create a master make file at the top of the source directory hierarchy. This make file calls the make file in each source directory below.
5. Run make to transfer files from the source directory to the input directory.

Appendix A contains a sample series of make files.

5.2 Building the Data Directory

The data directory tree contains the operating environment for kitting the subsets.

Follow these steps to build the data directory:

1. Create a directory with the name data.
2. Create a subdirectory with the name scps under the data directory.
3. Transfer your SCPs to data/scps.

5.3 Generating the Master Inventory

The master inventory file contains a record describing each file in the input hierarchy of a software product. The file is the ouput that results when you use the newinv utility to produce a new or updated master inventory of a software product. See Chapter 2 for descriptions of the format of the newinv output file and of the operations that the utility performs.

Follow this procedure to generate a master inventory file:

1. Change to the data directory.
2. If you are creating a new master inventory, use the touch command to create an empty file. For example:

   ```
   # touch UWS400.mi
   ```

   If you are updating the master inventory file for a product, use the existing master inventory file as your input file.
3. Generate the master inventory using a command like the following:

   ```
   # newinv UWS400.mi ../input
   ```

   Informational messages appear during the processing.
4. If you are updating a master inventory and files that were included previously are missing from the input hierarchy, a message appears. The message tells you

that the program will display the records representing those files for editing. The message tells you to press the RETURN key to continue or CTRL/C to leave the program.

5. When you press the RETURN key, the records representing missing files appear. For example:

```
0      ./usr/notneeded  UWSX11400
```

6. Delete those records that are no longer part of the product. Records that you do not delete are merged with the updated master inventory.

7. If you are updating a master inventory and files that were not part of the product previously are included in your hierarchy, a message appears. The message tells you that the program will display the records representing those files for editing. The message tells you to press the RETURN key to continue or CTRL/C to leave the program.

8. When you press the RETURN key, the new records appear. For example:

```
./usr/lib/myfile
```

9. Assign flag values and a subset name for those records that are now part of the subset. For example:

```
0 ./usr/lib/myfile UWSX11400
```

   Records that you edit are merged with the updated master inventory. Delete any records of files that do not belong in the updated product.

10. Do not include ULTRIX directories in any of your subsets. Retain all ULTRIX directory records in the inventory. Use the subset name RESERVED. For example:

```
0 ./usr     RESERVED
0 ./usr/lib RESERVED
```

5.4 Creating the key File

The `key` file stores the attributes of a software product. The `key` file is required by the `kits` procedure.

Create your `key` file using an editor, like `vi`. A common convention is to use the 3-character product code and the 3-digit version code with a `.k` extension for the filename. For example, the `key` file for the ULTRIX Worksystem Software Version 4.0 product has the name `UWS400.k`.

The `key` file is divided into two sections, product attributes and subset descriptors. Two percent characters (%%) separate the sections.

The product attributes section of the `key` file contains definitions identifying the software product. The `key` file product attributes section definitions and a description of each are listed in Table 5-1.

Table 5-1: Key File Control Section

Field	Description
NAME	The product name, for example, ULTRIX Worksystem Software.
CODE	The unique, 3-character, product code, for example, UWS. The following codes are reserved: DNP, DNU, EPI, FOR, LSP, SNA, UDC, UDT, UDW, UDX, ULC, ULT, ULX, UWS
VERS	The 3-digit version code, for example, 400, which `setld` interprets as 4.0.0.
MI	The name of the master inventory file for the product. This is the name used with the `newinv` utility.
ROOT	An optional flag that is set to 1 if you are building the ULTRIX operating system distribution.
COMPRESS	An optional flag that is set to 1 if you want to create compressed subset images. Compressed subset images use significantly less space in the output directory and on distribution media than do uncompressed subset images, but compressed subset images take longer to install.

The format for entering the `key` file product attributes section definitions follows:

- Separate the field name and its value with an equal sign (=), for example:

  ```
  CODE=ULT
  ```

- Enclose strings containing white space or shell meta characters in single quote characters ('), for example:

  ```
  'ULTRIX Worksystem Software'.
  ```

- Begin comment lines with a pound sign (#).

The subset descriptors section of the `key` file contains attribute descriptors for each subset that is part of the product.

There is one record for each subset. Each record contains four fields which are separated by tabs. Each record ends with a newline.

The fields that make up each record in the data section and a description of each are listed in the Table 5-2.

Table 5-2: Key File Data Section

Field	Description
Subset name	A character string, up to 15 characters, composed of the following: - The 3-character product code (for example, UWS) - A mnemonic identifying the subset (for example, X11) - The 3-digit version code (for example, 400) The examples above combine to form the subset name UWSX11400.
Dependency list	One of the following: - A dot (.) indicating no dependencies - A subset name (for example, ULTINET400) - Subset names separated by a logical OR symbol (|)
Flags	An unsigned integer. The bottom 8 bits are used by Digital to convey information to `setld`: - Bit 0, the sticky bit, when set indicates the subset cannot be removed; - Bit 1 when set indicates the subset is optional; - Bit 2 when set indicates the subset is updatable; - Bits 3 through 7 are reserved. The top 8 bits are undefined and available.
Subset description	Short description of functionality delimited by single quote characters ('), for example, 'X11 DECwindows User Environment'.

The example that follows shows the file `UWS400.k`, which is the `key` file for the ULTRIX Worksystem Software Version 4.0 product.

```
NAME='ULTRIX Worksystem Software V4.0'
CODE=ULT
VERS=400
MI=ULTRIX.mi
ROOT=1
RXMAKE=0
COMPRESS=1
# subset definitions follow
%%
ULTBASE400      .          1       'Base System'
ULTBIN400       .          0       'Kernel Configuration Files'
ULTINET400      .          0       'TCP/IP Networking Utilities'
ULTEXER400      .          0       'System Exerciser Package'
ULTNFS400       ULTINET400 0       'Network File System Utilities'
ULTUMAIL400     ULTINET400 0       'Extended (Berkeley) Mailer'
ULTMH400        ULTUMAIL400 0      'The RAND Mail Handler'
UWSX11400       ULTINET400 0       'X11/DECwindows User Environment'
UWSFONT400      .          0       'X11/DECwindows Fonts'
UWSFONT3D400    .          2       'X11 Fonts for 3D'
ULTDCMT400      .          2       'Document Preparation Software'
ULTPGMR400      .          2       'Software Development Utilities'
ULTINTL400      ULTPGMR400 2       'Internationalization Tools'
ULTSCCS400      .          2       'Source Code Control System'
UWSXDEV400      ULTPGMR400 2       'Worksystem Development Software'
UWSXDEV3D400    UWSXDEV400 2       'VAXstation 8000 Development'
ULTPASCAL400    .          2       'Pascal Development Package'
ULTVAXC400      .          2       'VAX C/ULTRIX'
ULTMAN400       ULTDCMT400 2       'On Line Manual Pages'
ULTACCT400      .          2       'Accounting Software'
ULTCOMM400      ULTINET400 2       'Communications Utilities'
ULTBSC400       .          2       'Bisynchronous Communications'
ULTMOP400       .          2       'Maintenance Operations Protocol'
ULTUUCP400      ULTCOMM400 2       'Unix-to-Unix Copy Facility'
```

5.5 Building the Output Directory

The output directory tree contains the `setld`-compatible files that are transferred to the destination system.

To build the output directory, create a directory with the name `output`.

5.6 Creating the SPACE File

The SPACE file is a place holder for tape records that is reserved by Digital.

Follow these steps to create the SPACE file:

1. Use the `cd` command to change to the output directory.
2. Enter the following command sequence:

```
touch space
tar cf SPACE space
```

5.7 Generating the kit Images

Run the `kits` program to generate data files that produce kit images for transfer to the output directory tree. The `kits` program resides in `/sys/dist`. See Chapter

2 for a description of the operations that the kits utility performs.

The kits command line syntax, with an example, follows:

```
kits <key_file> <input_directory> <output_directory>
kits UWS400.k ../input ../output
```

5.8 Building /etc/kitcap

The file /etc/kitcap contains an entry for each software product kit a node can generate. An /etc/kitcap file must exist on each node that can generate a kit for that software product. You can keep the kitcap file in the data directory and link the file /etc/kitcap to it.

The format of an /etc/kitcap entry for tape media, with an example, follows:

```
<kitcodeTK>:<kit_dir>:SPACE:SPACE:SPACE:INSTCTRL:<subset>[:<subset>]
UWS400TK:/sys/dist/<prod>/output:SPACE:SPACE:SPACE:INSTCTRL:UWSXRT400 \
:UWSMH400
```

The subsets are listed in the order in which they are named in the key file.

The format of an /etc/kitcap entry for RA60 disk media, with an example, follows:

```
<kitcodeRA>:<kit_dir>:<disk_part>:insctrl:INSTCTRL:<subset>[:<subset>]
UWS400RA:...:a:insctrl:INSTCTRL:UWSXRT400:UWSXDEV400
```

The subsets are listed in the order in which they appear on the tape.

5.9 Making the Distribution Media

This section tells you how to make distribution media containing the images to be installed using the setld utility.

5.9.1 Making Tape Media

Use the gentapes utility to make tape media. The command line syntax, with an example, follows:

```
gentapes [-wv] [node:]:<kitcode> <tape_drive>
gentapes mysystem:UWS400 /dev/nrmt0h
```

The -w option indicates write only; the -v option indicates verify only. If neither option is specified, the utility writes, rewinds the tape, then verifies.

If you specify a node, the gentapes utility looks for the output directory on the node you specify. You can use the Network File System (NFS) to remotely mount the kit on a machine with the correct drive.

5.9.2 Making RA60 Disk Media

Use the genra utility to make RA60 disk media. The command line syntax, with an example, follows:

```
genra [node:]:<kitcode> <directory>
genra mysystem:UWS400 /mnt
```

Reproducing setld-Compatible Kits 6

Versions 2.0 and later of ULTRIX include programs to reproduce `setld`-compatible software distribution kits. This chapter describes how to reproduce these kits for ULTRIX, ULTRIX Worksystem Software, ULTRIX Layered Products, and third party software packages distributed for ULTRIX.

Be certain that your software developer has been granted the right to copy the distribution kit before performing any of the procedures described in this chapter.

To reproduce copies of `setld`-compatible software distribution kits:

1. Create a product output directory.
2. Copy the files from your media.
3. Prepare the product output directory and database.
4. Produce individual pieces of media.

6.1 Creating a Product Output Directory

Follow these steps to create a product output directory:

1. Create a directory with enough space to contain the product you plan to copy.
2. Change to this directory using the `cd` command.

6.2 Copying from Your Media

Follow the instructions that apply to the type of media from which you are copying.

6.2.1 Copying from Tape

Follow this procedure to copy from tape media:

1. Mount the tape on the tape drive.
2. Issue a sequence of commands like the following:

```
dd if=/dev/nrmtnh of=TK50.1 bs=512
dd if=/dev/nrmtnh of=TK50.2 bs=10k
dd if=/dev/nrmtnh of=ROOT bs=10k
setld -x /dev/nrmtnh
```

   The letter *n* in the commands above represents the unit number of the drive you are using.

When the commands have executed, the copy procedure is complete.

6.2.2 Copying from an RA60 Disk

Follow this procedure to copy from an RA60 disk:

1. Mount the partition containing the product you want to copy on `/mnt`.
2. Type a command with the following syntax:

```
setld -x /mnt/<product_dir>
```

When the command has executed, the copy procedure is complete.

6.3 Preparing the Product Output Directory and Database

This procedure modifies the directory containing the product you copied to make the contents compatible with the `setld` utilities you use to reproduce the media.

1. Use the `cd` command to change to the directory `/sys/dist`.
2. Place the following command procedure in a file named, for example, `rtc:`

```
#!/bin/sh5
#

KITCAP=/tmp/kitcap
CURVOL=1

cd instctrl
I=`expr *.image : ').image'`
echo "Your product code is $I"

tar cf ../INSTCTRL *

echo "${I}TK:`pwd`:TK50.1|1:TK50.2|1:ROOT:INSTCTRL

for S in `awk '{print $3}' *.image`
do
     [ $S = ROOT ] && continue
     . $S.ctrl
     set -- `(IFS=:;echo $MTLOC)`
     VOL=$1
     LOC=$2
     [ $VOL -ne $CURVOL ] &&
     {
          echo ":%%$VOL          CURVOL=$VOL
     }
     echo ":$Sdone
echo >> $KITCAP

mv /tmp/kitcap /etc
```

3. Use the `cd` command to change to the output directory.
4. Execute the command procedure.

A product code appears at the end of the procedure. Retain this code which is part of your product identification.

6.4 Generating Individual Pieces of Media

You can reproduce distribution media from the output directory. You can reproduce tape and RA60 disk copies for distribution of the ULTRIX supported and unsupported kits.

6.4.1 Generating RA60 Disk Media

You can use the RA60 disk pack to distribute several products simultaneously, because each product is in a separate partition. The partition used for each product is specified in the command procedure you execute to prepare the output directory, so you do not need to specify it.

The `genra` command, which is in the `/sys/dist` directory, reproduces RA60 disk media. This command requires two arguments: a product code and a device special file name for the disk to which you are writing. The product code appears in a message when the command procedure used to prepare the output directory executes. The product code ULT is used in the example that follows.

Issue a command like the following to copy the product from the output directory to the correct partition of your RA60 disk:

```
/sys/dist/genra ULT /dev/ranc
```

Substitute the unit number of the drive on which your RA60 disk is ready and on line for the '*n*'.

6.4.2 Generating Tape Media

The `gentapes` command, which can be found in the `/sys/dist` directory, reproduces tape media. Products with the product code UWS cannot be written to 9-track tapes.

The `gentapes` command requires two arguments: a product code and a device special file name for the tape drive to which you are writing. The product code appears in a message when the command procedure used to prepare the output directory executes. The product code ULT is used in the example that follows.

Issue a command like the following to copy the product from the output directory to tape:

```
/sys/dist/gentapes ULT /dev/rmtnl
```

Substitute the unit number of the drive on which your TK50 tape cartridge is ready and on line for the '*n*'.

How setld Options Work 7

This chapter describes the operations that each `setld` option performs.

Except when using the `-i` option, you must be the superuser to run the `setld` utility. Therefore, the first operation for each option is to validate your superuser access. When a `setld` command line includes a pathname, the next operation for each option is to check access to that path after validating superuser access.

The sections that follow describe the remaining operations that occur when you execute `setld` commands using each of the `setld` options. The option descriptions are given in the order in which a system manager would probably use each option. Each command specifying an individual subset refers to the UWSX11400 subset.

When an option requires specification of a `location`, the location is the device special file, mount point, or server name followed by a colon (:) from which the subset or product is to be transferred.

7.1 The Load Option

Use the `-l` option when you load a software product. The command line syntax, with an example, follows:

```
setld -D [path] -l <location> [subset]
# setld -D /var/adm/ris -l /dev/rmt0h
```

The `setld` utility performs the following operations:

1. Checks access to the location specified in `location`
2. Loads `instctrl` information from the location specified in `location` to a temporary directory
3. Determines which subsets to load by calling SCP with the ACT variable set to `M` and `$1` (the first argument) set to `-l`. Those subsets for which SCP exits with a `0` status, indicating that the subset is available for installation, are divided into mandatory and optional, according to the FLAGS setting. If there is no SCP for a subset, `setld` assumes an exit status of `0`. Optional subsets are offered on the `setld` menu from which users select subsets.
4. Performs the following operations for each subset selected from the menu:
   a. Checks dependencies
   b. Checks size
   c. Runs `scp PRE_L`
   d. Loads subset
   e. Verifies subset
   f. Flags subset as verified
5. After all the subsets selected have been loaded, performs the following operations for each subset flagged as verified:
   a. Runs `scp POST_L`

b. If no `path` was specified on the command line, runs `scp C INSTALL`
c. Locks subset
d. Adds dependency lock records

7.2 The Configure Option

Use the `-c` option to call the SCP to perform node-specific, "online" configuration operations for a subset. The command line syntax, with an example, follows:

```
setld -D [path] -c subset message
```

```
# setld -D /var/adm/ris -c UWSX11400 INSTALL
```

The `setld` utility performs the following operations:

1. Validates existence of subset
2. Sets the ACT variable to C
3. Calls `<subset>.scp` with `message` as `$1` (the first argument)

7.3 The Delete Option

Use the `-d` option to delete subsets. The command line syntax, with an example, follows:

```
setld -D [path] -d subset[subset...]
```

```
# setld -D /var/adm/ris -d UWSX11400
```

The `setld` utility performs the following operations:

1. Checks that the subset is installed
2. Checks to see if the sticky bit is set, indicating the subset cannot be deleted
3. Checks dependencies. If any dependencies exist, displays the names of the dependent subsets and prompts the user to confirm the deletion
4. Sets ACT to `scp PRE_D_` and executes SCP
5. Deletes all files in the subset
6. Sets ACT to `scp POST_D` and executes SCP
7. If no `path` was specified, sets ACT to `C` and executes the SCP with `DELETE` as `$1` (the first argument)

7.4 The Inventory Option

Use the `-i` option to see either a list of subsets or a list of files in a subset. The command line syntax to list subsets with the status of each and its description, with an example, follows:

```
setld -D [path] -i
```

```
# setld -D /var/adm/ris -i
```

The command line syntax to list files in a subset, with an example, follows:

```
setld -D [path] -i [subset]
```

```
# setld -D /var/adm/ris -i UWSX11400
```

The `setld` utility checks the inventory of all the subsets installed if no subset name is given, or of individual subsets named.

7.5 The Extract Option

Use the `-x` option to extract files to set up a `ris` area. The command line syntax, with an example, follows:

```
setld [-D path] -x <location>
```

```
# setld -x /dev/rmt0h
```

The `setld` utility performs the following operations:

1. Checks access to the location specified in `location`
2. Loads `instctrl` information from the location specified in `location` to a temporary directory
3. Determines which subsets to load by calling SCP with the ACT variable set to `M` and `$1` (the first argument) set to `-x`. Those subsets for which SCP exits with a `0` status, indicating that the subset is available for installation, are divided into mandatory and optional, according to the FLAGS setting. If there is no SCP for a subset, `setld` assumes an exit status of `0`. Optional subsets are offered on the `setld` menu from which users select subsets.
4. Reads `instctrl` information from the temporary directory to `./instctrl`
5. Performs the following operations for each subset selected from the menu:
    a. Extracts subset image
    b. Runs `sum` on subset image
    c. Reports errors

7.6 The Verify Option

Use the `-v` option to verify a subset installation. The command line syntax, with an example, follows:

```
setld -v [subset]
```

```
# setld -v UWSX11400
```

The `setld` utility runs `scp V` to verify the installation of the subset specified.

7.7 The Update Option

Subsets on a user's system can be updated if the developer set the update flag (bit 2) in the data section of the `key` file for the subset when preparing the product for distribution. Use the `-u` option to update subsets that have the update flag set that match subsets on update distribution media.

The command line syntax, with an example, follows:

```
setld [-D path] -u <location>
```

```
# setld -u /dev/rmt0h
```

The `setld` utility performs the following operations:

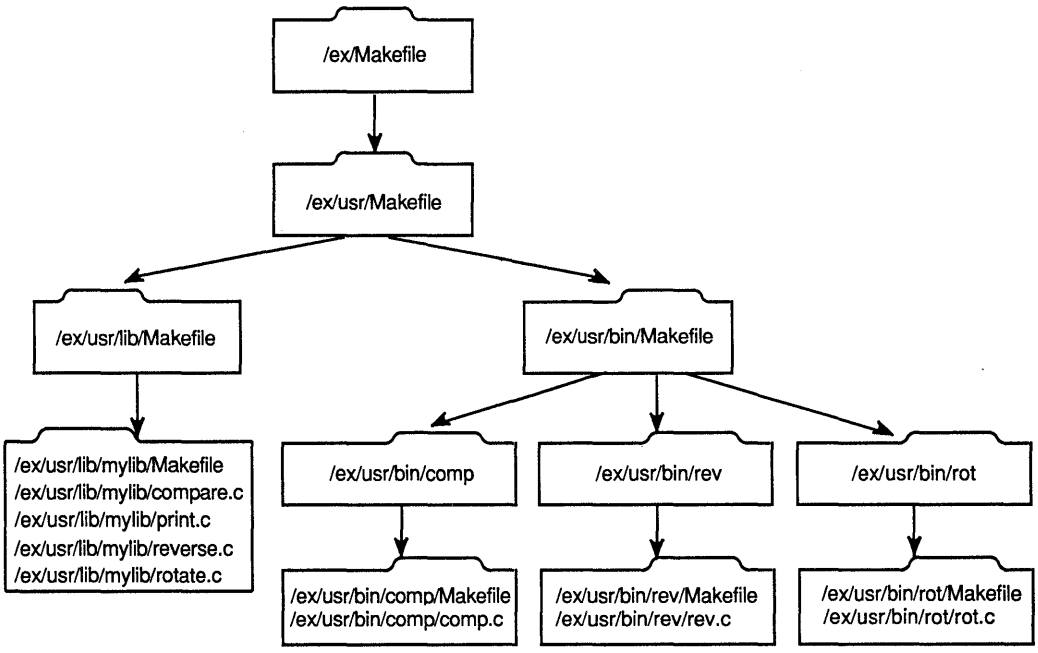
1. Checks access to the location specified in `location`
2. Loads `instctrl` information from the location specified in `location` to a temporary directory
3. Performs the following operations for each subset to be updated:
   - Merges the system subset inventory with the update subset inventory, creating a a synchronization file with the extension `.syn`.
   - Creates a lock file with a `.dw` extension.
   - Sets ACT to `scp PRE_U` and executes SCP.
   - Creates the directory `usr/adm/install/archive` if it does not exist.
   - Determines which files the user has changed and archives those files to the `usr/adm/install/archive` directory.
   - Loads the replacement files from the media.
   - Verifies the subset.
   - Moves the contents of the lock file with a `.dw` extension to a lock file with a `.lk` extension.
   - Sets ACT to `scp POST_U` and executes SCP.
   - Creates the directory `usr/adm/install/reference` if it does not exist.
   - Archives reference copies of all the files archived to the `usr/adm/install/archive` directory to the `usr/adm/install/reference` directory.
   - Checks the `p` (precedence) bit on the distribution media inventory `.inv` file and restores those files with the `p` bit set with the corresponding files in the `usr/adm/install/archive` directory.
   - Deletes files in the user's system subset that are no longer included in the update subset inventory. If a file that is no longer included in the update inventory has been modified, the file is saved in its present location, but is not attached to the subset.
   - Removes old inventory information from the `/usr/etc/subsets` directory.

Sample make Files A

This appendix contains a sample series of `make` files for managing the transfer of a software product from source directories to the input directory hierarchy from which the files will be processed using the `setld` utility. The appendix also contains the output generated when the sample `make` files execute.

Figure A-1 illustrates the directory hierarchy containing the `make` files.

Figure A-1: Sample make File Directory Hierarchy



ZK-0166U-R

The contents of each `make` file follows, in the order in which they would run. Each file contains a comment explaining its specific purpose.

A.1 /ex/Makefile

```
#
#       SCCSID  %W% (MYPRODUCT) %G%
#
#       Example Makefile, building a product, /
#
DESTROOT=
DIRS=usr
install:        $(DIRS)
        -for K in $(DIRS);\
        do\
                mkdir $(DESTROOT)/$$K;\
                (cd $$K; make DESTROOT=$(DESTROOT) install);\
        done
print:
        lpr Makefile
        for K in $(DIRS);\
        do\
                (cd $$K;make print);\
        done
```

A.2 /ex/usr/Makefile

```
#
#       SCCSID  %W% (MYPRODUCT) %G%
#
#       Example Makefile, building a product, /usr
#
DESTROOT=
DIRS=lib bin
TARGETS=dolib dobin
install:        $(TARGETS)
dolib:
        -[ -f $(DESTROOT)/usr/lib ] || mkdir $(DESTROOT)/usr/lib
        (cd lib;make DESTROOT=$(DESTROOT))
dobin:
        -[ -f $(DESTROOT)/usr/bin ] || mkdir $(DESTROOT)/usr/bin
        (cd bin;make DESTROOT=$(DESTROOT))
print:
        lpr Makefile
        for K in $(DIRS);\
        do\
                (cd $$K;make print);\
        done
```

A.3 /ex/usr/lib/Makefile

```
#
#       SCCSID  %W% (MYPRODUCT) %G%
#
#       Example Makefile, building a product, /usr
#
DESTROOT=
DIRS=mylib
install:
        for DIR in $(DIRS);\
        do\
                (cd $$DIR;make DESTROOT=$(DESTROOT) install);\
        done
print:
        lpr Makefile
        for DIR in $(DIRS);\
        do\
                (cd $$DIR;make print);\
        done
```

A.4 /ex/usr/lib/mylib/Makefile

```
# Makefile
#
#       example makefile for usr/lib/libmylib.a
#
DESTROOT=
LIBDIR=$(DESTROOT)/usr/lib
CFLAGS=-O
OBJS=compare.o print.o reverse.o rotate.o
install:        libmylib.a
        install -m 755 -o root -g system libmylib.a $(LIBDIR)
        ranlib $(LIBDIR)/libmylib.a
libmylib.a:     $(OBJS)
        ar cr libmylib.a $(OBJS)
print:
        lpr Makefile *.c
```

A.5 /ex/usr/lib/mylib/compare.c

```
/*      compare.c -
 *              example library module
 *
 *      compare() - compare two strings
*/
#ifndef lint
static char     *sccsid = "%W% (MYPRODUCT) %G%";
#endif
compare(s,t)
register char *s, *t;
{
        while( *s )
        {
                if( *s != *t )
                        break;
                ++s;
                ++t;
        }
        return(*s);
}
```

A.6 /ex/usr/lib/mylib/print.c

```
/*      print.c
 *              example library routine
 *
 *      print() - write string to stdout
*/
print(s)
char *s;
{
        write(0,s,strlen(s));
}
```

A.7 /ex/usr/lib/mylib/reverse.c

```
/*      reverse.c -
 *              example library module
 *
 *      reverse() - reverse a string
*/
#ifndef lint
static char *sccsid = "%W% (MYPRODUCT) %G%";
#endif
#define MAXBUF  1024
char *reverse(s)
register char *s;
{
        static char     buf[MAXBUF+1];
        char            *p;
        int             i;
        if( (i = strlen(s)) >= MAXBUF )
                return( (char *) 0 );
        p = buf + i;
        for( *p-- = '\0'; *s; )
                *p-- = *s++;
        return(p+1);
}
```

A.8 /ex/usr/lib/mylib/rotate.c

```
/*      rotate.c -
 *              example library module
 *
 *      rotate() - rotate each character in a string a given number
 *                      of places in the alphabet.
 *
 *              ASCII only.
*/
#ifndef lint
static char *sccsid = "%W% (MYPRODUCT) %G%";
#endif
#include        <ctype.h>
char *rotate(s,i)
register char *s;
register int i;
{
        char    *t;
        t = s;
        while( *s )
        {
                if( isupper(*s) )
                        *s = ((*s - 'A' + i) % 26) + 'A';
                else if( islower(*s) )
                        *s = ((*s - 'a' + i) % 26) + 'a';
                ++s;
        }
        return(t);
}
```

A.9 /ex/usr/bin/Makefile

```
#
#       SCCSID  %W% (MYPRODUCT) %G%
#
#       Example Makefile, building a product, /usr
#
DESTROOT=
DIRS=$(DESTROOT)/usr/bin
PROGS=comp rev rot
install:
        -mkdir $(DIRS)
        for K in $(PROGS);\
        do\
                (cd $$K;make DESTROOT=$(DESTROOT) install);\
        done
print:
        lpr Makefile
        for K in $(PROGS);\
        do\
                (cd $$K; make print);\
        done
```

A.10 /ex/usr/bin/comp/Makefile

```
#       Makefile
#
LIBS=$(DESTROOT)/usr/lib
CFLAGS=-O -s
install:
        cc -o comp comp.c -L$(LIBS) -lmylib
        install -m 755 -o root -g system comp $(DESTROOT)/usr/bin
print:
        lpr Makefile *.[ch]
```

A.11 /ex/usr/bin/comp/comp.c

```
/*      comp.c
 *              example string compare program
*/
#ifndef lint
static char *sccsid = "%W% (MYPRODUCT) %G%";
#endif
main(argc,argv)
int argc;
char *argv[];
{
        if( argc != 3 )
        {
                print("Argument Count Error\n");
                exit(1);
        }
        if( compare(argv[1],argv[2]) )
                print("Different\n");
        else
                print("Same\n");
}
```

A.12 /ex/usr/bin/rev/Makefile

```
#       Makefile
#
LIBS=$(DESTROOT)/usr/lib
CFLAGS=-O -s
install:
        cc  -o rev rev.c -L$(LIBS) -lmylib
        install -m 755 -o root -g system rev $(DESTROOT)/usr/bin
print:
        lpr Makefile *.[ch]
```

A.13 /ex/usr/bin/rev/rev.c

```
/*      rev.c -
 *              example program, reverses its argument and prints.
*/
#ifndef lint
static char *sccsid = "%W% (MYPRODUCT) %G%";
#endif
main(argc,argv)
int argc;
char *argv[];
{
        if( argc != 2 )
        {
                print("Argument Count Error\n");
                exit(1);
        }
        print(reverse(argv[1]));
        print("\n");
        exit(0);
}
```

A.14 /ex/usr/bin/rot/Makefile

```
#       Makefile
#
LIBS=$(DESTROOT)/usr/lib
CFLAGS=-O -s
install:
        cc -o rot rot.c -L$(LIBS) -lmylib
        install -m 755 -o root -g system rot $(DESTROOT)/usr/bin
print:
        lpr Makefile *.[ch]
```

A.15 /ex/usr/bin/rot/rot.c

```
/*      rot.c -
 *              example, rotate argv[1] by argv[2] in the ascii
 *              collating sequence.
*/
#ifndef lint
static char *sccsid = "%W% (MYPRODUCT) %G%";
#endif
main(argc,argv)
int argc;
char *argv[];
{
        if(argc != 3)
        {
                print("Argument Count Error\n");
                exit(1);
        }
        print( rotate(argv[1], atoi(argv[2])) );
        print("\n");
        exit(0);
}
```

A.16 Output From Sample Makefiles

```
Script started on Wed Aug 31 15:15:19 1988
csh:1  make DESTROOT=/example/root install
for K in usr; do mkdir /example/root/$K; (cd $K; make
DESTROOT=/example/root install); done
mkdir: /example/root/usr: File exists
[ -f /example/root/usr/lib ] || mkdir /example/root/usr/lib
mkdir: /example/root/usr/lib: File exists
*** Error code 1 (ignored)
(cd lib;make DESTROOT=/example/root)
for DIR in mylib; do (cd $DIR;make DESTROOT=/example/root install); done
ar cr libmylib.a compare.o print.o reverse.o rotate.o
install -m 755 -o root -g system libmylib.a /example/root/usr/lib
ranlib /example/root/usr/lib/libmylib.a
[ -f /example/root/usr/bin ] || mkdir /example/root/usr/bin
mkdir: /example/root/usr/bin: File exists
*** Error code 1 (ignored)
(cd bin;make DESTROOT=/example/root)
mkdir /example/root/usr/bin
mkdir: /example/root/usr/bin: File exists
*** Error code 1 (ignored)
for K in comp rev rot; do (cd $K;make DESTROOT=/example/root install);
done
cc -o comp comp.c -L/example/root/usr/lib -lmylib
install -m 755 -o root -g system comp /example/root/usr/bin
cc  -o rev rev.c -L/example/root/usr/lib -lmylib
install -m 755 -o root -g system rev /example/root/usr/bin
cc -o rot rot.c -L/example/root/usr/lib -lmylib
install -m 755 -o root -g system rot /example/root/usr/bin
csh:2 exit
script done on Wed Aug 31 15:23:06 1988
```

Index

R

S

T

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

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

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal*	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX
Guide to Preparing Software for
Distribution on ULTRIX Systems
AA-MG62B-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

What do you like best about this manual? _____

What do you like least about this manual? _____

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 _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

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

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line