Tru64 UNIX

ULTRIX to Tru64 UNIX Migration

Part Number: AA-PS3EE-TE

December 1997

Product Version: Tru64 UNIX

This manual describes how to migrate from an ULTRIX and UWS system to a Compaq Tru64 UNIX (formerly DIGITAL UNIX) system. The manual covers the migration to Tru64 UNIX Versions 3.0 to 3.2 from ULTRIX and UWS Versions 4.2 to 4.4 and to Tru64 UNIX Version 4.0B from ULTRIX and UWS Version 4.5.

The manual discusses migration issues for ULTRIX users, system and network administrators, and programmers.

Compaq Computer Corporation
Houston, Texas

Contents

2 Overview of the Tru64 UNIX User Environment

3 Migrating Your ULTRIX User Environment to a Tru64 UNIX System

Part 3 Migrating Your System and Network Administration Environment

4 Overview of Tru64 UNIX System and Network Administration

7 Migrating Your ULTRIX Application to a Tru64 UNIX System

G Migration from ULTRIX Version 4.5 to Tru64 UNIX Version 4.0B

Index

Examples

Figures

Tables

About this Manual

This manual compares the Compaq Tru64 UNIX (formerly DIGITAL UNIX) operating system to the ULTRIX operating system by describing the differences between the two systems. This manual also contains information about software components of the Tru64 UNIX product.

Note

> This manual does not contain information about software components or products that you purchase separately from the Tru64 UNIX product.

Audience

This manual is written for ULTRIX users, system and network administrators, and programmers who need information about migrating to the Tru64 UNIX system:

- Users should read this manual to determine what differences exist between using an ULTRIX system and using a Tru64 UNIX system.
- System and network administrators should read this manual to determine what differences exist between administering an ULTRIX system and network and a Tru64 UNIX system and network.
- Programmers should read this manual to determine what differences between the ULTRIX programming environment and the Tru64 UNIX programming environment affect the migration of applications.

Organization

This manual discusses the following topics:

Part I **Introduction**

Chapter 1 Is an overview of migration from the ULTRIX operating system to the Tru64 UNIX operating system.

Part II **Migrating Your User Environment**

Chapter 2 Is an overview of the Tru64 UNIX user environment that describes differences from the ULTRIX environment.

Chapter 3	Describes how to set up your Tru64 UNIX user environment so that it is similar to your ULTRIX user environment. Also, it describes how to migrate shell scripts from an ULTRIX system to a Tru64 UNIX system.
Part III	**Migrating Your System and Network Administration Environment**
Chapter 4	Is an overview of the Tru64 UNIX system and network administration environment that describes differences from the ULTRIX environment.
Chapter 5	Describes how to set up a Tru64 UNIX system for maximum compatibility with ULTRIX systems.
Part IV	**Migrating Your Applications**
Chapter 6	Is an overview of the Tru64 UNIX programming environment that describes differences from the ULTRIX environment.
Chapter 7	Describes the steps involved in migrating source applications from ULTRIX systems to Tru64 UNIX systems.
Chapter 8	Describes how to use certain features of Tru64 UNIX, such as shared libraries.
Part V	**Appendixes**
Appendix A	Describes differences between Tru64 UNIX and ULTRIX commands, including how to get the behavior of ULTRIX commands on Tru64 UNIX systems, where applicable.
Appendix B	Describes differences between Tru64 UNIX and ULTRIX header files and routines, including how these header file differences affect program portability.
Appendix C	Describes differences between Tru64 UNIX and ULTRIX system calls, including how to get the behavior of ULTRIX system calls on Tru64 UNIX systems, where applicable.
Appendix D	Contains three sample programs that show modem control.
Appendix E	Summarizes the differences between XUI and OSF/Motif terminology, windows and window managers, menus and menu items, standard message boxes, and mouse button bindings.
Appendix F	Summarizes the differences between XUI and OSF/Motif component names.
Appendix G	Summarizes issues when migrating from ULTRIX Version 4.5 to Tru64 UNIX Version 4.0B.

Related Documents

In addition to this manual, you should read the following Tru64 UNIX manuals as you move to a Tru64 UNIX system:

- General users

  - *Technical Overview*
- System and network administrators
  - *Installation Guide*
  - *System Administration*
  - *Network Administration*
  - *Security*
  - *Sharing Software on a Local Area Network*
- Programmers
  - *Programmer's Guide*
  - *Programming Support Tools*
  - *Writing Software for the International Market*
  - *Network Programmer's Guide*
  - *Guide to Realtime Programming*
  - *Guide to DECthreads*

Icons on Tru64 UNIX Printed Books

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

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

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

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

Reader’s Comments

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

You can send your comments in the following ways:

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

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

  `/usr/doc/readers_comment.txt`

- Mail:

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

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

Please include the following information along with your comments:

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

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

Conventions

`%` `$`	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.
`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.

Part 1

Introduction

1

Introduction to Migrating from ULTRIX to Tru64 UNIX Systems

This chapter is an overview of migrating from the ULTRIX and ULTRIX Worksystem Software (UWS) operating system, Version 4.2 and higher, to the Tru64 UNIX operating system, Version 3.0 and higher. It begins by describing some Tru64 UNIX features that are unavailable on ULTRIX systems. Then, it gives brief information about features that are common to both systems, followed by a list of ULTRIX features that are unavailable on Tru64 UNIX systems.

This chapter also mentions each of the migration issues that can affect you as you move to a Tru64 UNIX system. You can use this information to assess the effort involved in migration. Detailed information about migration from ULTRIX to Tru64 UNIX systems is provided in the rest of this manual.

1.1 Tru64 UNIX Features Unavailable on ULTRIX Systems

The following Tru64 UNIX system features are unavailable on ULTRIX systems:

- OSF/1 kernel
- Real-time kernel
- Improved standards conformance
- Shared libraries
- Increased number of per-process open file descriptors
- Logical storage management
- Streams
- Memory-mapped files
- Sixty-four bit data types and addresses
- Security integration architecture (SIA)
- Increased number of pseudoterminals (ptys)
- Debugging support for multithreading
- Increased file system sizes

The following sections describe these Tru64 UNIX features in more detail.

1.1.1 OSF/1 Kernel

The OSF/1 kernel is based on the Mach kernel developed at Carnegie-Mellon University. This kernel consists of a compact, extensible system kernel designed to support distributed and parallel computing services for single and multiprocessor systems. The OSF/1 kernel provides the basic operating system services, including virtual memory management and interprocess communications.

Additional code implements UNIX services as extensions to the kernel. These extensions to the kernel are designed as kernel subsystems. File systems, network protocol families and pseudodevice drivers, and STREAMS drivers are some of the supported subsystems. Each subsystem is configured into the kernel by an existing kernel-supported framework, which provides the mechanism for registering the driver entry routines. For example, the operating system supports the virtual file system framework, the network framework, the device switch framework, the interrupt dispatch framework, and the STREAMS framework.

1.1.2 Real-Time Kernel

The Tru64 UNIX real-time kernel and environment provides you with the capability of developing and running portable applications in a POSIX environment. POSIX 1003.4 Draft 11 is a set of functions and calls that can be used in the design and creation of real-time applications.

The Tru64 UNIX real-time environment offers the following POSIX features:

- Process synchronization. Processes can be synchronized by using the following methods:
  - Real-time clocks and timers
  - Priority scheduling
  - Semaphores
- Shared memory
- Process memory locking
- Asynchronous I/O

For more information on real-time programming, see the *Guide to Realtime Programming*.

1.1.3 Standards Compliance

Using programming standards enhances the portability of your application. Standard-compliant code is independent of the hardware or even the operating system on which the application runs.

Both the ULTRIX and UWS system and the Tru64 UNIX system have programming environments that allow you to develop applications that conform to the major industry standards.

The *Software Product Description* (SPD) for the ULTRIX product, the UWS product, and the Tru64 UNIX product each contain detailed lists of the standards they support. Refer to the SPDs for this information. For information about specific migration issues, see Chapter 7.

The Tru64 UNIX system provides programming interfaces that are defined in the OSF Application Environment Specification (AES) standard. Although the AES is not a formal standard, using AES-conformant library routines helps ensure the portability of your program between products based on the OSF/1 operating system. The *Application Environment Specification Operating System Programming Interfaces Volume (AES/OS)* specifies programming interfaces for the operating system portion of the OSF applications environment.

1.1.4 Shared Libraries

The Tru64 UNIX system includes dynamic, shared libraries as part of the programming environment. That is, the libraries contain no fixed base addresses. When you link your application with a shared library, the executable application does not contain the library; instead, it contains the information needed to load the shared library at startup time and to access the shared routines at execution time.

Because shared libraries allow several applications to use a single copy of a library routine, they can help save disk space and memory, and improve system performance.

For more information about shared libraries, see Section 6.6 and Section 8.1.

1.1.5 Per-Process Open File Descriptors

Both the ULTRIX and UWS and the Tru64 UNIX systems allow you to configure the number of open file descriptors a process can use. By default, the number for ULTRIX and UWS systems is 64, for Tru64 UNIX systems, 4096. For information about how to configure this feature, see the *System Administration* manual. For information about modifying an application to use a different number of open file descriptors, see Section 8.3.

1.1.6 Logical Storage Manager

The Logical Storage Manager (LSM) subsystem is a replacement for the Logical Volume Manager of previous Tru64 UNIX systems. See Chapter 4 for more information.

1.1.7 Streams Kernel Mechanism

System V Release 3.2 STREAMS is included in the Tru64 UNIX system. STREAMS is a kernel mechanism that supports development of network services and data communications drivers. The STREAMS mechanism consists of a set of system calls, kernel resources, and kernel routines that can create, use, and dismantle a stream. A stream is a full-duplex processing and data transfer path between a driver in kernel space and a process in user space.

1.1.8 Memory-Mapped Files

The Tru64 UNIX system includes the Berkeley `mmap` function, which allows an application to access files with memory operations rather than file I/O operations. See `mmap`(2) for more information.

1.1.9 Sixty-Four Bit Data Types and Addresses

The Alpha architecture is based on a 64-bit microprocessor. As such, it introduces a number of extended capabilities beyond 32-bit architectures. For example, 64-bit addressing allows the Tru64 UNIX system to support file system sizes greater than 2 gigabytes (GB). Most applications only require a recompilation in order to run on a Tru64 UNIX system. However, if you want your application to be portable (run on both 32-bit and 64-bit systems) and to interoperate with programs on other systems, you must check the C coding techniques. Chapter 7 describes specific aspects of the C language and explains certain programming techniques that will help both new program development and the migration of existing programs from ULTRIX to Tru64 UNIX systems.

1.1.10 Security Integration Architecture

The Tru64 UNIX system includes the security integration architecture (SIA). SIA is a framework that can support multiple, layered security mechanisms on a system.

SIA can be employed in base or enhanced security modes. By using the SIA routines, the security commands access a `matrix.conf` file. Which `matrix.conf` file is accessed depends on the security mode employed (basic or enhanced) and the security mechanisms that are enabled through SIA. This information is contained in the *Security* manual.

In the Tru64 UNIX system, SIA routines, through the appropriate `matrix.conf` file, also control the access and manipulation of both `passwd` and `group` entries, employing several of the security-related programming routines. See the *Security* manual for more information.

The Tru64 UNIX `login`, `su`, and `passwd` commands, and `xdm` (the workstation login box) use the SIA interfaces. The `passwd` and `group` entries in the `/etc/svc.conf` file are provided in the Tru64 UNIX system for archival library (like those in the ULTRIX system) compatibility.

SIA log information is written to the `/var/adm/sialog` file, whenever the `sialog` file is present and enabled.

If you want to manipulate password or group information, contact your Compaq Computer Corporation representative for information on obtaining SIA interface information.

1.2 Features Common to ULTRIX and Tru64 UNIX Systems

The Tru64 UNIX system includes most features that are available on typical UNIX systems, such as the ULTRIX system. In many cases, you use the Tru64 UNIX system in the same way that you use the ULTRIX system. This section briefly describes the Tru64 UNIX system features. Differences between how Tru64 UNIX features operate and how the ULTRIX equivalent features operate are described in other chapters of this manual.

1.2.1 User Applications, Commands, and Shells

The Tru64 UNIX system has most of the user commands, such as `grep`, `who`, `man`, and `more`, that are available on the ULTRIX system. In most cases, you use the same command options and arguments on ULTRIX and Tru64 UNIX systems. The Tru64 UNIX system also provides the `vi` and `ex` text editors, among others (such as Emacs). The interfaces to the editors are the same, so you need not learn new editing commands to edit files on a Tru64 UNIX system. Workstation applications that you use on the ULTRIX system are also available on the Tru64 UNIX system; for example, the Tru64 UNIX system provides the Bookreader software, the Calendar and Clock software, and the visual differences program, `dxdiff`. For complete information about user applications and commands, see Section 2.3 and Appendix A.

The Tru64 UNIX system provides three shells: the C shell (`csh`), the Bourne shell (`sh`), and the Korn shell (`ksh`). For information about how these shells compare to the ULTRIX equivalent shells, see Section 2.4. For information about migrating shell scripts, see Section 3.2.

1.2.2 Development Tools

The Tru64 UNIX development environment is similar to the ULTRIX development environment. Both the Tru64 UNIX system and the ULTRIX system (for RISC users) offer an ANSI C compliant compiler. In addition to compiling ANSI C programs, the compiler includes a compilation mode that allows you to compile programs written in traditional Kernighan and Ritchie (K&R) C.

The programming tools that are available on the Tru64 UNIX system are traditional UNIX programming tools and new tools. You can use the `dbx` debugger on Tru64 UNIX systems to find errors in your program. The system includes the `make` utility for building your application and the `sccs` utility for storing your application's source files. You can purchase the DEC FUSE software for use on the Tru64 UNIX system; DEC FUSE includes a computer-aided software engineering (CASE) environment for developing software.

Using the Tru64 UNIX programming software, you can write programs that communicate over a network, that provide windowing user interfaces, that are portable to multiple systems, and that are adaptable to multiple locales.

Once your application is fully developed, you can use the `setld` utility to package the application as a kit for installation on Tru64 UNIX systems. For more information about the Tru64 UNIX development environment, see Chapter 6. For information on building installable software kits, see the *Programming Support Tools* manual.

1.2.3 File Systems

The Tru64 UNIX system supports the following file systems:

- Advanced File System (AdvFS)
- UNIX File System (UFS)
- Network File System (NFS)

- CD-ROM (compact disc read-only memory) File System (CDFS)

  Conforms to the ISO 9660 standard. See `cdfs`(4) for more information.

- Virtual File System (VFS) interface and framework

  The VFS interface enables transparent access to UFS and NFS file systems and allows both file systems to run in parallel. Transparent access is accomplished by retaining the traditional operating system interfaces on top of the VFS layer. Therefore, the file system types are not apparent to the user.

- The /proc File System

  A debugging file system that is compatible with the System V, Release 4 specification. This file system is a development tool that allows any process to control and monitor the execution of another unrelated process. See `proc`(4) for more information.

You set up, check, and maintain Tru64 UNIX file systems in much the same way that you perform these tasks on ULTRIX systems. For information about administering file systems, see Section 4.8.

1.2.4 System and Network Administration Tools

System and network administration tasks on a Tru64 UNIX system are comparable to those on an ULTRIX system.

You can set up your system so that users can print files on local and remote printers. Users can also print PostScript files on local or remote PostScript printers. The remote printing occurs over a Transmission Control Protocol/Internet Protocol (TCP/IP) network.

The Tru64 UNIX system includes TCP/IP for use on a local area network (LAN) or wide area network (WAN). You manage the network components by using many of the same commands that you use on an ULTRIX system, such as the `arp`, `ifconfig`, and `hostid` commands. The system also includes the Simple Network Management Protocol (SNMP) Agent, which gives information to an SNMP network management station.

The Tru64 UNIX system includes many of the distributed system services available on ULTRIX. In particular, it includes the Berkeley Internet Name Domain (BIND) service and the Network Information Service (NIS, formerly called YP). For time synchronization, the Tru64 UNIX system includes the Network Time Protocol (NTP).

Like the ULTRIX system, the Tru64 UNIX system uses the `sendmail` utility as the general-purpose internet mail router. The system also includes the `uucp` utility. You can use `uucp` to copy files between UNIX systems and to execute commands on remote UNIX systems.

For more information about how the Tru64 UNIX system and network management environment compares to the same environment on ULTRIX systems, see Chapter 4.

1.2.5 Data Interoperability

In many cases, you can exchange data easily between Tru64 UNIX and ULTRIX systems. For example, you can mount an ULTRIX file system on a Tru64 UNIX system. (For information about performing this task, see Section 5.1.) In addition, you can use Tru64 UNIX commands to read tapes you create with the ULTRIX `tar` and `pxtar` commands. (For information about using these commands on a Tru64 UNIX system, see Section 5.3.) You can also use the `cpio` and `ltf` commands to read and write tape archives. You can use the `dump` command on an ULTRIX system and the `restore` command to restore the dump on a Tru64 UNIX system. In addition, you can use TCP/IP network copying facilities.

Users on an ULTRIX system can also exchange data with a Tru64 UNIX system provided that files are less than 2 gigabytes (GB) in size.

1.2.6 Symmetric Multiprocessing

The Tru64 UNIX system supports Symmetric Multiprocessing (SMP). SMP is a modification to the kernel that allows multiple processors to execute the kernel code simultaneously. SMP activity is accomplished safely by means of locks, which are used to control the concurrent access of shared data structures within the kernel.

The SMP software on Tru64 UNIX systems has a high degree of commonality with the ULTRIX SMP software. You can migrate your ULTRIX SMP applications to Tru64 UNIX systems as long as you ensure that the applications conform to the migration information in this manual. There are no specific SMP migration considerations for users or system managers, and the only programming considerations are:

- ULTRIX SMP applications can use two system calls, `startcpu` and `stopcpu`, which have no equivalent calls on Tru64 UNIX systems.
- ULTRIX SMP applications employ the `cpustat` command to display information about the use and state of each CPU in a SMP system. The Tru64 UNIX system employs the `kdbx` debugger for the same purposes. See `kdbx`(8) for more information.

1.3 ULTRIX Features Unavailable on Tru64 UNIX Systems

The Tru64 UNIX system provides many ULTRIX features, but it omits some ULTRIX features. The following list shows features that are available on ULTRIX systems but not on Tru64 UNIX systems:

- Support for VAX hardware
- Support for the MIPS RISC hardware
- Support for most terminals or printers not manufactured by Compaq
- N-buffered I/O services
- Diskless Management Services (DMS) (although there is some support for a dataless environment in Tru64 UNIX systems)
- DECwindows debugger, `dxdb` (although the DEC FUSE programming environment is supported)
- Hesiod software (although BIND software is supported)
- Kerberos software (see Section 6.9.2)
- The Remote Procedure Calling package, DEC RPC (see Section 6.9.1)
- Extended SNMP features
- ULTRIX disk partitioning `ioctl` functions
- XUI graphical user interface (GUI)

1.4 Migrating from ULTRIX to Tru64 UNIX Systems

Migrating from an ULTRIX system to a Tru64 UNIX system involves:

- Migrating a user from an ULTRIX to a Tru64 UNIX system
- Migrating an ULTRIX system and network management environment to a Tru64 UNIX system
- Migrating an application from an ULTRIX to a Tru64 UNIX system

You might be involved in one or more of these types of migration.

This section gives a brief overview of the three migration paths, including brief descriptions of the issues involved in migration.

1.4.1 Migrating as a User

As a user of the Tru64 UNIX system, you will notice few differences between the Tru64 UNIX system and the ULTRIX system. Most commands and tools are the same or similar on the two systems. For information about the differences that do exist between the Tru64 UNIX and ULTRIX user environments, see Chapter 2.

The following list describes the major tasks involved in migrating from an ULTRIX to a Tru64 UNIX system:

- Using commands

Most commands are similar on the Tru64 UNIX and ULTRIX systems. For a list of specific differences between commands, see Appendix A.

- Using shells

  You should notice few differences in how the shell you use operates. You might need to modify shell scripts to use them on a Tru64 UNIX system. For information about porting shell scripts, see Section 3.2.

- Setting environment variables

  You might need to set the `PATH`, `filec`, or international environment variables on your Tru64 UNIX system. For information about setting these environment variables, see Section 3.1.

1.4.2 Migrating Your System and Network Administration Environment

The system and network administration tasks on both systems are similar. Many of the commands, setup scripts, and utilities you use on an ULTRIX system are available on a Tru64 UNIX system. The following list describes some of the tasks you might need to perform on a Tru64 UNIX system as you migrate from an ULTRIX system:

- Mount ULTRIX file systems on a Tru64 UNIX system (as described in Section 5.1).
- Read information from tape archives by using the Tru64 UNIX `tar` command (as described in Section 5.3).
- Add devices to your system after you configure the system (as described in Section 5.4).
- Configure daemons for ULTRIX compatibility (as described in Section 5.7 and Section 5.8).

1.4.3 Migrating Your Application

You can migrate an application from an ULTRIX to a Tru64 UNIX system in one of two ways: either as source code (recommended) or as executable code, as described in the following sections. If the source code is unavailable or you need an operating version of the program while you are migrating source code, you can migrate an executable.

1.4.3.1 Migrating Source Code

To migrate source code from an ULTRIX to a Tru64 UNIX system, follow these steps:

1. Copy your program source files (and `make` files, if any) to the Tru64 UNIX system.

2. If you require additional development environment tools to build your application, migrate those tools to the Tru64 UNIX system.
3. Modify your `make` files, if necessary, so that they work on the Tru64 UNIX system.
4. Select the appropriate compilation mode and correct any C syntax errors.
5. Evaluate changes in symbols (undefined symbols, multiply defined symbols) and modify the source code appropriately.
6. Evaluate your header files (missing header files, changed header file names) and modify the source code appropriately.
7. Evaluate differences between a 32-bit and 64-bit programming environment and modify the source code appropriately.
8. Run `lint`, if possible, to identify other errors. Correct the errors as you find them.
9. Evaluate and modify references to libraries and library routines that are provided on ULTRIX systems but not on Tru64 UNIX systems.
10. Run your program and correct semantic errors.
11. Test your program thoroughly on the Tru64 UNIX system.

For more information about the work needed to complete these tasks, see Chapter 7.

1.4.3.2 Migrating Executables

You can migrate a MIPS ULTRIX executable by using the DECmigrate for DEC OSF/1 Alpha product. This product is made up of the `mx` translator and the `mxr` run-time system.

The `mx` translator translates only user mode programs. It does not:

- Translate kernel code.
- Support applications that read system memory by using `/dev/kmem` or `/dev/mem`.
- Support applications that depend on exact memory layout or file formats of system-provided files.
- Translate ULTRIX executables older than ULTRIX Version 4.0.
- Translate MIPS executables other than ULTRIX and Tru64 UNIX Version 1.0 executables.
- Translate big endian MIPS programs.
- Provide precise exception behavior.

- Emulate MIPS instruction atomicity.
- Translate MIPS II or MIPS III programs (no R4000 processor support).
- Enable translated programs to use the Tru64 UNIX shared libraries.

For more information on the mx translator and the mxr run-time environment, see the *DECmigrate for DEC OSF/1 V1.2: Translating Executables* manual.

Part 2

Migrating Your User Environment

This part gives an overview of the Tru64 UNIX user environment and describes specific differences between Tru64 UNIX and ULTRIX systems that affect users.

2

Overview of the Tru64 UNIX User Environment

Using a Tru64 UNIX operating system is similar to using an ULTRIX and UWS operating system. Like an ULTRIX and UWS system, a Tru64 UNIX system offers both a windowing graphical user interface (GUI) for workstations and a terminal interface.

Also like ULTRIX and UWS, the Tru64 UNIX workstation interface is DECwindows, based on the industry standard OSF/Motif. As a result, there are no significant differences between the workstation interfaces of both systems. ULTRIX and UWS does offer, in addition, an XUI version of the DECwindows interface, based on a DIGITAL proprietary graphical user interface.

In addition to the windowing interface, you can use the Tru64 UNIX system from a terminal or from a workstation window that emulates a terminal. With few exceptions, the commands and tools you use on an ULTRIX system are on the Tru64 UNIX system. Tru64 UNIX command and file names are case sensitive, just as they are on the ULTRIX system. You can use pipes, command input and output redirection, and background jobs in the same way that you use these features on an ULTRIX system.

This chapter gives an overview of the Tru64 UNIX user environment, including differences in the workstation environment, differences in the Tru64 UNIX directory structure, and differences in supported tools and shells, and differences in the security environment.

Note

For details about using a Tru64 UNIX system, see the *DECwindows User's Guide* and the *Command and Shell User's Guide*.

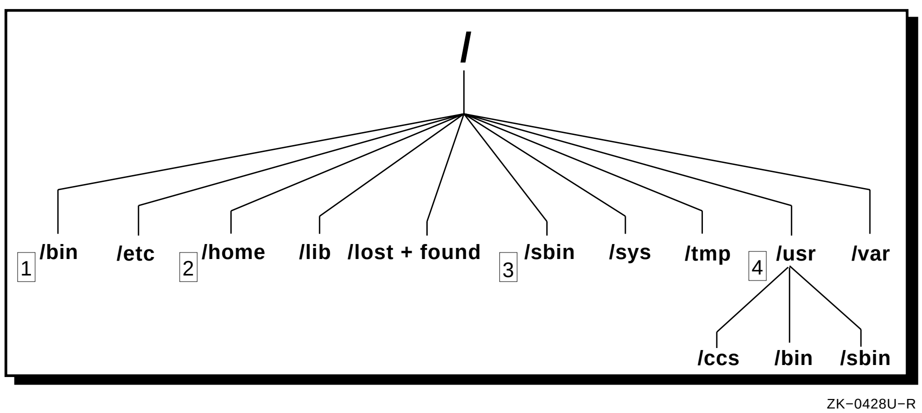
2.1 Differences in the Tru64 UNIX DECwindows Interface

The Tru64 UNIX DECwindows interface is based on OSF/Motif Version 1.2.3. By contrast, ULTRIX and UWS gives you a choice of two DECwindows interfaces: OSF/Motif and XUI. The OSF/Motif interface is almost identical to the Tru64 UNIX system interface, because the ULTRIX and UWS implementation is based on OSF/Motif Version 1.2.2. The ULTRIX and UWS XUI interface is based on the DIGITAL developed graphical user interface.

2.2 Differences in the Tru64 UNIX Directory Structure

The directory structure on your Tru64 UNIX system is different from the directory structure on an ULTRIX system. Figure 2–1 shows most of the directories in the root (/) file system.

Figure 2–1: Tru64 UNIX Directory Structure for General Users



As the figure shows, the directory structure on Tru64 UNIX is identical to the ULTRIX directory structure in many ways. (This figure does not show the complete directory structure; for example, the `/opt`, `/dev`, and `/mnt` directories, which are typically not used by general users, are omitted. See also Section 4.8.1.) The following list describes important differences:

1. Some commands that are in `/bin` on an ULTRIX system are in the `/usr/bin` or `/usr/sbin` directory on a Tru64 UNIX system. This change should not affect you because your `PATH` environment variable causes the Tru64 UNIX system to search the appropriate directories for commands. As a start, you can use the same definition for the `PATH` environment variable as you used on the ULTRIX system. However, you should remove `/bin` from your path definition and add `/usr/bin` and `/usr/sbin`.

   If you need to determine the location of a particular command that is not in your path, you can use the `whereis` command, which looks for commands in a set of standard locations. If a given command file is in more than one directory, `whereis` reports all locations of the command.

2. The Tru64 UNIX directory structure contains the `/home` directory. On Tru64 UNIX systems, this directory is intended to be used to contain the home directories for users. For example, the home directory for a

user named Ross might be /home/ross. See your system administrator for the actual location of your home directory.

3 The Tru64 UNIX directory structure contains the /sbin and /usr/sbin directories. The /sbin directory contains commands that system administrators use when the system is in single-user mode; /usr/sbin contains commands administrators use in multiuser mode.

4 The Tru64 UNIX directory structure does not contain the /usr/etc or /usr/ucb directories. Most commands that reside in these directories on an ULTRIX system are, on the Tru64 UNIX system, in the /usr/bin directory. This change should not affect you, but you should remove /usr/ucb and /usr/etc from your path definition and add /usr/bin.

Other than these differences, you should notice no difference between the directory structures on the ULTRIX and Tru64 UNIX systems during daily use.

2.3 Differences in User Applications and Commands

The following list describes the user applications that are packaged on the Tru64 UNIX system:

- Bookreader

  The Bookreader program for Tru64 UNIX workstations has a user interface based on Motif, and is similar to the Bookreader program on ULTRIX workstations. For information about using Bookreader, see dxbook(1X) or start the Bookreader program and read its online help information.

- Calculators

  The bc and dc calculators are the same on the Tru64 UNIX system as they are on the ULTRIX system. The Tru64 UNIX system does not supply the DECwindows Calculator program dxcalc. Use the xcalc program instead; a link from dxcalc to xcalc is provided. For information about using these calculator programs, see bc(1), dc(1), and xcalc(1X).

- Calendar

  The Calendar program for Tru64 UNIX workstations has a user interface based on Motif, and is similar to the one on ULTRIX workstations. For information about using the Calendar program, see dxcalendar(1X) or start the Calendar program and read its online help information.

- Cardfiler

  The Cardfiler program for Tru64 UNIX workstations has a user interface based on Motif, and is similar to the one on ULTRIX workstations. For information about using the Cardfiler program, see dxcardfiler(1X) or start the Cardfiler program and read its online help information.

- CDA Viewer

  The CDA Viewer program on Tru64 UNIX workstations has a user interface based on Motif, and is similar to the one on ULTRIX workstations. For information about using the CDA Viewer, see dxvdoc(1X). You can also start the CDA Viewer and read its online help information.

- DECterm

  The DECterm terminal emulator program for Tru64 UNIX workstations has a user interface based on Motif, and is similar to the one on ULTRIX workstations. For information about using DECterm, see dxterm(1X) or start a DECterm window and read the online help information.

- Editors and other pattern-scanning tools

  The ed, ex, sed, vi, and GNU Emacs editors are the same as the editors of the same names on the ULTRIX system. For information on using these editors, see ed(1), ex(1), sed(1), and vi(1). The GNU Emacs editor features an operational xmenu interface, and is described in the *GNU Emacs Manual*. If the Emacs editor is installed on your system, this manual is contained in the following PostScript file:

  ```
  /usr/lib/emacs/doc/emacs.ps
  ```

  The Tru64 UNIX distribution media also includes the Emacs source code as an optional item.

  The awk program is essentially the same as the ULTRIX awk program. For information on awk, see awk(1). The sed editor and the awk program are also discussed in *Programming Support Tools*. The gawk program is essentially the same as the ULTRIX nawk program. For information on gawk, see gawk(1). The DECwindows Notepad program, which is also an editor, is described later in this list.

- Examples

  The Tru64 UNIX system includes a full suite of demos and sample programs in the /usr/examples/motif library, including the xcd and periodic programs. The xcd program allows you to play music compact discs in a RRD42 CD–ROM drive attached to your system. The periodic program displays a periodic chart of Motif widgets.

- General-purpose commands

  Commands for searching files (such as grep), listing directory contents and moving between directories (ls, cd, and pwd), displaying the date and time (date), and so on are, in most cases, the same as the ULTRIX equivalent commands. Differences are noted in Appendix A. The ps command functions in either of the following two ways:

  – If you omit the minus sign before the option keywords (for example, ps x), the command functions like the BSD ps command.

– If you include the minus sign (for example, `ps -x`), the command functions like the System V `ps` command.

The two versions have different lists of options; see `ps`(1).

- Mail

  Tru64 UNIX mail commands are the same as their ULTRIX equivalents except that the command names are different. The `mail` command on an ULTRIX system invokes `/usr/ucb/mail` (the `Mail` user agent). To use this mail handler on a Tru64 UNIX system, enter the `Mail` or the `mailx` command. The Tru64 UNIX user interface for the `mailx` user agent is slightly different from that of the ULTRIX version. The `mail` command on a Tru64 UNIX system invokes the `/bin/mail` program (the `binmail` user agent). You can use the Message Handler Utility (MH) just as you use MH on an ULTRIX system, with the exception that the Tru64 UNIX MH utility does not support bulletin boards. For information about using the `binmail` and `mailx` commands and the MH utility, see `binmail`(1), `mailx`(1), and `mh`(1).

  The DECwindows Mail program on Tru64 UNIX workstations has a user interface based on Motif, and is similar to the DECwindows Mail program on ULTRIX workstations. For information on other differences and on using DECwindows Mail, see `dxmail`(1X).

- Notepad

  The Notepad program on Tru64 UNIX workstations has a user interface based on Motif, and is similar to the one on ULTRIX workstations. For information about using the Notepad program, see `dxnotepad`(1X) or start the Notepad program and read its online help information.

- Paint

  The Paint program on Tru64 UNIX workstations has a user interface based on Motif, and is similar to the one on ULTRIX workstations.

- Reference pages

  Reference pages that describe the various Tru64 UNIX commands are on line. You can read the references pages on line by using the `man` command, just as on an ULTRIX system. For example, enter the following command at your system prompt:

  ```
  % man man
  ```

  This command displays the `man` command's reference page.

  The section numbers for some reference pages have changed. For example, on ULTRIX systems, Section 4 describes special files. On Tru64 UNIX systems, Section 4 describes file formats. The following list describes the sections that compose the Tru64 UNIX reference pages:

  – Section 1 describes user commands.

  – Section 2 describes system calls.

  – Section 3 describes library routines.

  – Section 4 describes file formats.

  – Section 5 describes macro packages and conventions.

  – Section 6 describes games and unsupported programs. As supplied by DIGITAL, this section is empty.

  – Section 7 describes special files.

  – Section 8 describes system and network administration commands.

  Also, the reference pages are stored under the `/usr/share/man` directory on Tru64 UNIX systems. On ULTRIX systems, the reference pages are stored in the `/usr/man` directory.

- Remote system commands

  The `rdate`, `rlogin`, `rsh`, `rwho`, and `ruptime` remote login commands are the same as the ULTRIX equivalent commands. You can use these commands to communicate with remote Tru64 UNIX systems, ULTRIX systems, and other systems that offer BSD network support. For information about using these commands, see `rdate`(1), `rlogin`(1), `rsh`(1), `rwho`(1), and `ruptime`(1).

- Remote file transfer commands

  The `ftp`, `tftp`, and `rcp` commands are the same as the ULTRIX equivalent commands. You can use these commands to transfer files between Tru64 UNIX and ULTRIX systems, and between Tru64 UNIX and other systems that offer Internet networking support. For information about using these commands, see `ftp`(1), `tftp`(1), and `rcp`(1).

The uucp command on Tru64 UNIX systems differs from the uucp command on ULTRIX systems. The Tru64 UNIX uucp command has some features that the ULTRIX uucp command does not have. The Tru64 UNIX uucp command does not support the –W option. For more information about using the Tru64 UNIX uucp command, see uucp(1).

- talk command

  The talk command is the same as the ULTRIX talk command. For information about using the talk command, see talk(1).

- telnet command

  The telnet command is the same as the ULTRIX telnet command. For information about using telnet, see telnet(1).

- Text formatting commands

  The deroff, neqn, nroff, and tbl commands are similar to the equivalent commands from the VAX ULTRIX system. Some of these commands have different options, and VAX ULTRIX nroff drivers can be ported to a Tru64 UNIX system. (RISC ULTRIX nroff drivers cannot be ported.) The default device for Tru64 UNIX nroff is –Tlp; the ULTRIX default is –T37 (Teletype Model 37). For more information about these commands, see deroff(1), neqn(1), nroff(1), and tbl(1).

- User information commands

  Commands such as finger, w, and who are the same as the ULTRIX equivalent commands. For more information about these commands, see finger(1), w(1), and who(1).

- Visual differences program

  The dxdiff DECwindows visual differences program is the same as the ULTRIX dxdiff program. For information about dxdiff, see dxdiff(1X).

2.4 Differences in Shells

The Tru64 UNIX system supports three shells: the C shell (csh), the Korn shell (ksh), and the Bourne shell (sh). This section gives a brief overview of each shell's features and syntax, highlighting differences between it and the equivalent ULTRIX shell.

2.4.1 Differences in the C Shell

The C shell is an interactive command interpreter and a command programming language that uses a syntax similar to the C programming language. The shell carries out commands either from a shell script or interactively from a terminal keyboard.

In most respects, the Tru64 UNIX C shell is the same as the ULTRIX C shell. In the Tru64 UNIX C shell, you must set an environment variable to enable file name completion on a Tru64 UNIX system and an environment variable to enable command-line editing. (For information about enabling file name completion, see Section 3.1.1. For information about enabling command-line editing, see Section 3.1.)

The Tru64 UNIX C shell does not support the `hashstat` built-in command for debugging the shell. The `hashstat` command displays statistics that indicate how effective the internal hash table has been at locating commands.

Other than these differences, the Tru64 UNIX C shell is the same as the ULTRIX C shell. For information about porting C shell scripts, see Section 3.2.3.

For more information about the Tru64 UNIX C shell, see `csh`(1).

2.4.2 Differences in the Korn Shell

The Korn shell is an interactive command interpreter and a command programming language. The shell carries out commands either interactively or from a shell script. The Korn shell contains many of the features of the Bourne shell, as well as some C shell features.

The Tru64 UNIX Korn shell is the same as the ULTRIX Korn shell. If you use the ULTRIX Korn shell interactively, you should notice no difference when you use the Tru64 UNIX Korn shell interactively. Shell scripts written for the ULTRIX Korn shell should run without modification using the Tru64 UNIX Korn shell.

For more information about the Tru64 UNIX Korn shell, see `ksh`(1).

2.4.3 Differences in the Bourne Shell

The Bourne shell is an interactive command interpreter and a command programming language. The shell carries out commands either interactively or from a shell script. The Bourne shell is the default system shell on a Tru64 UNIX system.

The ULTRIX system has two versions of the Bourne shell, `sh` and `sh5`. The Bourne shell on the Tru64 UNIX system is most similar to `sh5`.

If you use the `sh` shell on an ULTRIX system, you might notice the following differences when you use `sh` on a Tru64 UNIX system:

- The shell determines whether the argument you specify to the built-in `cd` command is a subdirectory of any of the directories specified in the definition of the `CDPATH` environment variable. If the shell finds a subdirectory that matches the argument you specify, it changes your current directory to that subdirectory. The ULTRIX `sh` shell does not have this feature.

- The default search path for the Tru64 UNIX `sh` shell is `/usr/bin`. On the ULTRIX system, the default search path is `:/bin:/usr/bin`. On the Tru64 UNIX system, `/bin` is a link to `/usr/bin`; you do not need to add `/bin` to the definition of your Tru64 UNIX `PATH` environment variable.
- The Bourne shell on ULTRIX has one variant of the shell command, `set –`, that does not exist on Tru64 UNIX systems.
- The Tru64 UNIX Bourne shell contains a built-in `echo` command. The ULTRIX Bourne shell does not contain an `echo` command.

These differences might affect the portability of your `sh` shell scripts. For information about porting `sh` shell scripts, see Section 3.2.

The Tru64 UNIX Bourne shell (`sh`) is almost identical to the ULTRIX `sh5` shell; however, its name is different, and there are a few other minor differences. The difference in name does not affect how you use the Bourne shell interactively; however, it might affect the portability of your `sh5` shell scripts. Other differences are very minor but can cause subtle failures of ported scripts. For information about porting `sh5` shell scripts, see Section 3.2.

2.5 Differences in Security Features

Like the ULTRIX operating system, the Tru64 UNIX system includes features that allow you to control access to your account, files, and workstation. For information on using the Tru64 UNIX security features, see the *Security* manual.

The Tru64 UNIX system omits the following security features that are found on ULTRIX systems: trusted path, audit, and enhanced identification and authentication features (including the shadow password file). For example, the Tru64 UNIX system does not support the equivalent of the ULTRIX `authenticate_user` programming interface. Additionally, a Tru64 UNIX system's system administrator cannot define a Secure Attention Key that you press before you log in to the system.

3

Migrating Your ULTRIX User Environment to a Tru64 UNIX System

This chapter describes how to set up your Tru64 UNIX user environment so that it is similar to your ULTRIX user environment. This chapter also describes how to port shell scripts from an ULTRIX system to a Tru64 UNIX system.

3.1 Setting Environment Variables

In most cases, you can set environment variables on your Tru64 UNIX system the same as you set them on your ULTRIX system. You might need to set the following environment variables differently on a Tru64 UNIX system:

- `editmode`
- `filec`
- `PATH`
- `LANG`
- `LC_COLLATE`
- `LC_CTYPE`
- `LC_MESSAGES`
- `LC_MONETARY`
- `LC_NUMERIC`
- `LC_TIME`

This section describes how you set these environment variables. Note that the `CSHEDIT` environment variable is not supported on the Tru64 UNIX system. To enable command-line editing, enter the following command:

```
% set editmode {emacs|vi}
```

You can include this command in your `.login` file to have the `editmode` variable set each time you log in or in your `.cshrc` file to set the variable in all subshells.

3.1.1 Setting the C Shell filec and PATH Environment Variables

The Tru64 UNIX system C shell contains most features of the ULTRIX C shell. One difference between the two shells is that the ULTRIX C shell includes file name completion by default. On Tru64 UNIX systems, you must set the `filec` environment variable to enable file name completion.

To set the `filec` environment variable, enter the following command:

```
% set filec
```

You can include this command in your `.login` file to have the `filec` variable set each time you log in or in your `.cshrc` file to set the variable in all subshells. Once you set the variable, you can press the Escape key to request that the shell complete file names on the command line.

On Tru64 UNIX systems, the default search path for the `csh` shell is `.:/usr/bin`. On ULTRIX systems, the default search path is `.:/usr/bin:/bin`. On the Tru64 UNIX system, the `/usr/ucb` directory is a link to the `/usr/bin` directory. For information about the Tru64 UNIX C shell, see `csh`(1).

3.1.2 Setting the Bourne Shell PATH Environment Variable

On Tru64 UNIX systems, the default search path for the `sh` shell is `:/usr/bin`. On ULTRIX systems, the default search path is `:/usr/bin:/bin`. On the Tru64 UNIX system, the `/bin` directory is a link to `/usr/bin`, so there is no need to add `/bin` to your path. However, there are commands in `/usr/sbin` that you might want to access. To enable the shell to access commands in `/usr/sbin`, add that directory to the `sh` search path. The following example shows the line to include in your `.profile` file to add the `/bin` directory to the default search path:

```
PATH=:/usr/bin:/usr/sbin; export PATH
```

Including this command in your `.profile` file adds the `/usr/sbin` directory to the default `sh` search path each time you log in to the system.

3.1.3 Setting International Environment Variables

The Tru64 UNIX system has environment variables that control some aspects of how you interact with programs. The environment variables control how international programs display messages, accept input, and display data. International programs use Tru64 UNIX features to display messages in your native language, collate strings as you expect, format monetary and numeric data as you expect, and so on. The following sections describe how to set these environment variables.

3.1.3.1 Setting the Environment Variable for Messages

To display a message in your native language, a program reads the message from a message catalog. By default, your program searches the /usr/lib/nls/msg/%L/%N path for message catalogs. In the preceding pathname, %L represents the locale name specified by the LANG environment variable, and %N represents the name of the message catalog, which is usually similar to program_name.cat.

If the message catalog your program needs is not stored in one of the default directories, you must set the NLSPATH environment variable, as you did on ULTRIX systems. The NLSPATH environment variable tells the program where to find the message catalogs.

3.1.3.2 Setting the Environment Variables for Data Handling

You can set a number of environment variables that control how programs accept input, display data, and manipulate data. The international environment variables on a Tru64 UNIX system are LANG, LC_ALL, LC_COLLATE, LC_CTYPE, LC_NUMERIC, LC_MONETARY, LC_TIME, and LC_MESSAGES. For a description of these environment variables, see Section 6.10.2.1.

To define these international environment variables, you specify a string, called the locale name, that tells the system what language, territory, and codeset to use in your environment. You may also be able to specify a modifier that allows you to further refine program display and data input.

The Tru64 UNIX system uses a naming convention for locales different from the ULTRIX system. On ULTRIX systems, the language specifier is three characters long and uppercase. On Tru64 UNIX systems, the language specifier is two characters long and lowercase. In addition, the format of the codeset names differ between the ULTRIX and Tru64 UNIX systems. For example, to choose an environment that supports French as it is spoken in France, enter the following command on a Tru64 UNIX system:

```
% setenv LANG fr_FR.ISO8859-1
```

On ULTRIX systems, international environment variables have little effect on the commands on the system. For example, setting the LC_TIME variable to a French locale name does not cause the date command to display dates as you expect them to be displayed in France. However, on Tru64 UNIX systems, the setting of the LC_TIME variable does affect the operation of the date command, as well as other commands.

The Tru64 UNIX system supports more locales than the ULTRIX system. However, the ULTRIX ISO 646 and DEC Multinational character set codesets are not supported on Tru64 UNIX systems. Therefore, the following locales are unavailable on Tru64 UNIX systems:

- ENG_GB.MCS
- ENG_GB.646
- FRE_FR.MCS
- FRE_FR.646
- GER_DE.MCS
- GER_DE.646

On Tru64 UNIX systems, locales are installed in the `/usr/lib/nls/loc` directory. For a list of available locales, see the *Technical Overview*.

3.2 Migrating Shell Scripts

In most cases, your shell scripts will port from ULTRIX to Tru64 UNIX with few modifications. You might need to modify your shell script because of differences between Tru64 UNIX and ULTRIX commands or because of differences between the shells on Tru64 UNIX and ULTRIX systems.

3.2.1 Modifying Commands Used in Scripts

A number of commands are different between Tru64 UNIX and ULTRIX systems. Most differences are in the options or arguments for a given command. Some commands operate differently on Tru64 UNIX systems, and some ULTRIX commands are unavailable on Tru64 UNIX systems.

For example, the Tru64 UNIX `test` command works differently from the ULTRIX `test` command. On a Tru64 UNIX system, the `–f` option makes the `test` command determine whether a file exists and is a regular file; that is, the file is not a directory, a character-special file, a block-special file, or a named pipe. On an ULTRIX system, the `–f` option makes the `test` command determine whether a file exists and is not a directory. Because of this difference, the `test -f` command can return unexpected results on a Tru64 UNIX system. You can get the effect of the ULTRIX `test -f` command on a Tru64 UNIX system by replacing the `test -f` command with the following command:

```
(test -f file) -o (test -c file) -o (test -b file) -o (test -p file)
```

By sequentially testing for a regular file (`–f`), a character-special file (`–c`), a block-special file (`–b`), or a named pipe (`–p`), this command tests one `file` to be sure it is not a directory. The command returns status in the same way as the ULTRIX `test -f` command.

If your scripts contain explicit path references to commands that are in different directories on the Tru64 UNIX system, you must change these references to reflect the Tru64 UNIX locations.

For more information about command differences that could affect porting your shell script from ULTRIX to Tru64 UNIX, see Appendix A.

3.2.2 Migrating Korn Shell Scripts

The Korn shell (`ksh`) is the same on Tru64 UNIX and ULTRIX systems. You need not modify your shell scripts.

3.2.3 Migrating C Shell Scripts

The C shell on Tru64 UNIX systems is the same as the C shell on ULTRIX systems, with one exception. Because the C shell on Tru64 UNIX systems does not support the `hashstat` built-in command, you must remove it from the ULTRIX C shell script before you move the script to a Tru64 UNIX system. The Tru64 UNIX system does not have an equivalent for this command.

3.2.4 Migrating sh Shell Scripts

The Bourne shell on Tru64 UNIX systems is largely the same as the Bourne shell on ULTRIX systems. Some differences between the two shells do exist. The following list describes changes you should make to your ULTRIX `sh` scripts or your user environment to port `sh` scripts to a Tru64 UNIX system:

- Check any `cd` commands.

  The Tru64 UNIX `cd` command might change your current directory to one that you do not expect. To avoid this problem, specify only absolute pathnames as arguments to the `cd` command.

  On Tru64 UNIX systems, the shell determines whether the argument you specify to the `cd` command is a subdirectory of any of the directories specified in the definition of the `CDPATH` environment variable. If the shell finds a subdirectory that matches the argument you specify, it changes your current directory to that subdirectory. The ULTRIX `sh` command does not have this feature.

- Remove the `set –` command from shell scripts.

  The Tru64 UNIX system does not have the `set –` command or any equivalent.

- Modify references to the `echo` command so that they invoke the `/bin/echo` command.

  The Tru64 UNIX shell contains a built-in `echo` command. References to the `echo` command in a shell script that you run on a Tru64 UNIX system invoke the built-in `echo` command. The ULTRIX Bourne shell contains no built-in `echo` command. References to the `echo` command in your ULTRIX shell script invoke the `/bin/echo` command.

The Tru64 UNIX built-in echo command differs from the /bin/echo command. For example, the built-in echo command does not support the –n option. If you use the echo –n command in a shell script, the output from the command includes the –n, as shown:

```
% echo -n hello
-n hello
```

Modify your shell script so that it invokes the /bin/echo command, as shown in the following example:

```
/bin/echo -n hello
```

(See the information about the sh shell in Appendix A for more differences between the /bin/echo command and the built-in echo command.)

The /bin/echo command is the same on ULTRIX and Tru64 UNIX systems.

3.2.5 Migrating sh5 Shell Scripts

The first two bytes of an executable program, called a magic number, tell the system what kind of program it is. The first line of most shell scripts is a magic number consisting of the combination of a number sign and an exclamation point (#!). This magic number tells the system to execute the rest of the line as if it were a normal shell command. Most shell scripts invoke the shell for which they are written to ensure that the script is executed by the appropriate shell. The first line of most scripts written for the ULTRIX sh5 shell is:

```
#! /bin/sh5
```

Because the Tru64 UNIX system uses a different name for the Bourne shell, these scripts fail. You must modify the first line to invoke the sh shell on a Tru64 UNIX system, as shown:

```
#! /bin/sh
```

If a script must run when the system is in single-user mode, specify /sbin/sh instead of /bin/shto get the statically linked version of the shell.

One significant difference between the ULTRIX sh5 shell and the Tru64 UNIX sh shell is in their treatment of positional parameters when a function is called. The Tru64 UNIX sh shell sets the positional parameters to the function call's arguments as does the ULTRIX sh5 shell. However, the Tru64 UNIX sh shell also saves the values the positional parameters held before the function was called. Upon return from the function, the shell sets the positional parameters to the saved values. The ULTRIX sh5 shell does not restore the positional parameters in this way; it leaves them set to the values they hold when the function returns. If your scripts do not rely on the ULTRIX behavior, this difference is transparent.

The most efficient way to modify the first line in a number of sh5 scripts is to write a shell script. Example 3–1 shows a shell script that changes the first line in sh5 scripts.

Example 3–1: Shell Script to Convert sh5 Scripts into sh Scripts

```
#! /bin/sh

trap 'rm -f /tmp/conv$$ ; exit ' 0 1 2                   [1]

for i                                                     [2]
do
     sed '1s/bin\/sh5/bin\/sh/' $i > /tmp/conv$$          [3]
     [ -f /tmp/conv$$ ] && {                              [4]
          mv /tmp/conv$$ $i                               [5]
}
done
```

1. The trap command makes the shell recognize the 0, 1, or 2 signals. If the shell receives one of these signals, it removes the file /tmp/conv*$$*, where *$$* is the process number of the current process. The shell script uses this file during its processing.

   Once the /tmp/conv*$$* file is removed, the shell script exits.

2. The for command starts a loop that continues as long as there are arguments on the shell script command line. Therefore, if you invoke this shell script with three arguments, the loop executes three times. The loop executes the commands between do and done.

3. The sed command modifies the first line of its input. The command searches for the string bin/sh5 and replaces it with the string bin/sh. The sed command writes its output to the /tmp/conv*$$* file.

4. The command in brackets ([]) tests to see that the /tmp/conv*$$* file exists and has a size greater than zero.

   The brackets are an alias for the /usr/bin/test command.

   The && separator specifies that the command in braces ({ }) is executed only if the test is true.

5. The mv command moves the /tmp/conv*$$* file to the location of the original input file. In effect, this command writes the converted shell script over the input file.

The shell script in Example 3–1 modifies only the first line in its input. You cannot use it to replace any sh5 invocation commands that appear on lines other than the first line of a shell script. You must either modify those invocation commands by hand or modify this shell script so that it replaces all sh5 invocation commands.

To use the shell script in Example 3–1, use the vi editor or some other editor to create a file on your Tru64 UNIX system that contains the script. Then, use the chmod command to set the file permissions on the script so that you can execute it. For example, if you name the script convert, enter the following chmod command:

```
% chmod u+x convert
```

Invoke the shell script by typing its name, followed by the names of sh5 scripts you want to convert. You can name as many shell scripts as you want on the command line, up to the maximum command-line length.

For example, suppose you want to convert three shell scripts: setup, modify, and remove. To convert the three shell scripts, enter the following command:

```
% convert setup modify remove
```

The convert script reads each file, one at a time, and changes its first line, if necessary. The converted shell script is stored in the same file as the input shell script; in this case, the converted shell scripts are named setup, modify, and remove.

Be sure to test the converted shell scripts for other possible incompatibilities before placing them into daily use.

Part 3

Migrating Your System and Network Administration Environment

This part gives an overview of the Tru64 UNIX system and network administration environment, and describes specific differences between Tru64 UNIX and ULTRIX systems that affect system and network administrators.

4

Overview of Tru64 UNIX System and Network Administration

The Tru64 UNIX system and network administration environment is similar to the ULTRIX administration environment. You can use most administration tools on a Tru64 UNIX system in the same way as on an ULTRIX system. However, some differences do exist. This chapter is an overview of the Tru64 UNIX system and network administration environment, describing the differences from the ULTRIX environment.

This chapter does not give detailed information about administering a Tru64 UNIX system or using Tru64 UNIX system administration tools. Administering a Tru64 UNIX system is described in the *System Administration* manual and the *Network Administration* manual.

4.1 Installation and System Setup

Installation and system setup are similar on Tru64 UNIX and ULTRIX systems. The Tru64 UNIX installation procedure, like the ULTRIX installation procedure, can use both the `setld` software and Remote Installation Services (RIS) software to install a bootable system from media. Both systems have setup scripts that you use in similar ways to set up systems after an installation.

The Tru64 UNIX installation supports configuring a system after installation. This feature allows you to install software on several system disks at one machine. You can then move each system disk to its own machine and configure it for use there. Take note of cabling inconsistencies and possible logical unit address changes (which affect the `/etc/fstab` file) when moving disk devices between systems.

Unlike an ULTRIX and UWS system, where you choose whether to install UWS, when you install a Tru64 UNIX system, the mandatory windowing software is automatically installed. The *Installation Guide* lists the subset names. If you do not need the windowing software, you can use the `setld -d` command to remove its subsets after the installation is complete.

Like the ULTRIX system, the Tru64 UNIX system is organized into software subsets. Some subsets are required at installation time, while others are optional. The contents of various Tru64 UNIX subsets might be different from ULTRIX subsets. For information about the Tru64 UNIX subsets, see the *Installation Guide*.

The Tru64 UNIX installation procedure creates log files that record the result of the installation. These log files are created in the `/var/adm/smlogs` directory. On ULTRIX systems, the log files are created in the `/var/adm` directory.

4.2 Available System Setup Scripts

Like the ULTRIX system, the Tru64 UNIX system includes setup scripts that you can use to complete the installation and configuration of your system's environment. You should use these setup scripts to set up various Tru64 UNIX utilities. The scripts are similar to the ULTRIX scripts that have the same name, but some differences might exist. For information about using the setup scripts, see the *Network Administration* manual.

Table 4–1 lists the scripts available on a Tru64 UNIX system.

Table 4–1: Setup Scripts Available on Tru64 UNIX Systems

Setup Script	Purpose
`addgroup`	Adding groups to your system
`adduser`	Adding users and creating users' home directories
`bindsetup`	Setting up the Berkeley Internet Name Domain (BIND) service
`latsetup`	Setting up the local area transport (LAT) service
`lprsetup`	Adding local and remote printers to your system
`mailsetup`	Setting up mail
`MAKEDEV`	Installing device-special files
`netsetup`	Establishing and adding nodes to a local area network (LAN)
`nfssetup`	Setting up a Network File System (NFS) file system
`nissetup`	Setting up the Network Information Services (NIS, formerly called YP)
`ntpsetup`	Configuring the Network Time Protocol (NTP) daemon
`snmpsetup`	Setting up the Simple Network Management Protocol (SNMP) Agent
`strsetup`	Configuring STREAMS special device files
`svcsetup`	Modifying the name service configuration file, `/etc/svc.conf`
`uucpsetup`	Configuring your system for `uucp` connections

4.3 System Customization Files

Both the Tru64 UNIX and ULTRIX systems have files that you use to customize your system. You can use some of your ULTRIX customization files on your Tru64 UNIX system with little or no modification. Typically, the only changes you must make are to remove references to ULTRIX specific features. The following are some of these files:

- From the root directory (/, the superuser's home directory):
  - `.cshrc`
  - `.login`
  - `.mailrc`
  - `.profile`
  - `.rhosts`
  - `.Xdefaults`
  - `.X11Startup`
- From the `/etc` directory:
  - `acucap`
  - `automount.master`
  - `exports`
  - `hosts`
  - `hosts.equiv`
  - `phones`
  - `remote`
  - `resolv.conf`
  - `svcorder`

In addition, a number of configuration files are the same on ULTRIX and Tru64 UNIX systems, except that the Tru64 UNIX system does not support the Hesiod naming service. Once you remove references to Hesiod from the following files, you can use them on your Tru64 UNIX system:

- `netgroup`
- `networks`
- `protocols`
- `rpc`
- `services`

Other configuration files are different on ULTRIX and Tru64 UNIX systems. For example, the Tru64 UNIX system does not have the following configuration files:

- `crontab`

  Instead of using an `/etc/crontab` file, the directory `/usr/var/spool/cron/crontabs` contains a number of files that the `cron` daemon uses to start facilities. For more information, see `cron`(8).

- `rc.local`

  On a Tru64 UNIX system, the system initialization functions performed by the ULTRIX `/etc/rc.local` file are provided by the `/etc/inittab` file and the shell scripts in the `/sbin/init.d` directory. For more information about system initialization, see the *System Administration* manual.

- `gettytab`

  The `/etc/gettytab` file is obsolete and has been replaced by `/etc/gettydefs`. To allow communication with systems using nonstandard parameters, copy one of the existing `gettydefs` entries and edit the copy as required to provide the parameters you need. See `gettydefs`(4) for specific file format information.

- `ttys`

  The function of the `/etc/ttys` file is changed. Tru64 UNIX systems use the `/etc/ttys` file to control root access by marking which lines are secure. The `/etc/inittab` file is used to configure terminal lines. You might want to save your ULTRIX `/etc/ttys` file for information on the configurations of specific terminal lines, but the format of the `/etc/inittab` file is very different. See `inittab`(4) for specific file format information.

Information about the differences between most other ULTRIX and Tru64 UNIX customization files is in this chapter. For information about creating and modifying those files, see the *Network Administration* manual and the *System Administration* manual.

4.4 System Configuration

When you install the Tru64 UNIX system, the distribution software includes the files that the system needs to create and build the core kernel and the kernel subsystems. You might need to reconfigure your system, on occasion, to align and tune it to meet the changing conditions of your site.

The Tru64 UNIX configuration procedure is similar in many ways to the ULTRIX procedure. The procedure consists of the Berkeley Standard Distribution Version 4.3 (BSD 4.3) configuration scheme, which includes the mechanism for configuring a kernel according to the definitions found in the static system configuration file, /sys/conf/*NAME*, where *NAME* is the name of your system, in uppercase letters. The kernel calls the autoconfig routine at startup time to configure physical devices that are defined in the configuration file and are connected to the system. Devices that are defined in the configuration file, but are not connected to the system, are not configured and cannot be used. Other subsystems (file systems and network protocol families, for example) are initialized and configured if they are defined in the /sys/conf/*NAME* file, and if the corresponding subsystem framework is present and activated.

Like the ULTRIX configuration file, the Tru64 UNIX configuration file contains a number of parameters that you can use to tune your system. The parameters on the Tru64 UNIX system differ from the ULTRIX parameters. For information about using the Tru64 UNIX parameters, see the *System Administration* manual.

As with ULTRIX, you build a new kernel on the Tru64 UNIX system automatically by using the doconfig program. You can also build a new kernel manually by using the config program. The only difference is that the config program on Tru64 UNIX systems is in the /sys/bin directory. On ULTRIX systems, the program is in the /etc directory. When you build a kernel on the Tru64 UNIX system, the doconfig or config program places the newly built kernel in the directory /sys/*NAME*, where *NAME* is your system name. For more information about building a new kernel, see the *System Administration* manual.

4.5 System Security Features

The Tru64 UNIX system has elementary features that allow you to control access to your system. For example, you can create and remove accounts and set permissions for files and directories. These system security features included in the Tru64 UNIX system are the traditional UNIX security features. For information about using these security features, see the *System Administration* manual.

The Tru64 UNIX system also contains more sophisticated security features. These features are described in the *Security* manual.

4.6 Print Services

The Tru64 UNIX system includes the traditional BSD UNIX capabilities for printing files. The system supports a print spooler for queuing print jobs to one or more printers. The /etc/printcap file describes the printers available, including their characteristics. You can print files on a remote Tru64 UNIX system over the TCP/IP network, just as you can on an ULTRIX system. You can print files on a local or remote PostScript printer, files on a printer connected to a LAT port, and files that contain the appropriate PostScript prologue print without modification.

Although the Tru64 UNIX system supports basic print capabilities, it does not support the PrintServer for ULTRIX software to print files on the Compaq family of PrintServer network laser printers. Compaq offers an optional software package for supporting PrintServer printers on Tru64 UNIX systems; licenses for this software are bundled with the printers themselves, and the software is available separately. Contact your local Compaq salesperson for further information about PrintServer support. See the *System Administration* manual and *Network Administration* manual for information on setting up printers.

The following list compares the basic printing capabilities of the Tru64 UNIX system and the same capabilities on an ULTRIX system:

- The print management and use commands are the same.

  The `lprsetup` utility is available and performs the same tasks on a Tru64 UNIX system as on an ULTRIX system; namely, creating entries in the `/etc/printcap` database, creating spool directories, creating accounting files, and so on. Other commands, such as `lpq`, `lprm`, `lpc`, `lp`, and `pac` are the same as the equivalent commands on an ULTRIX system.

- The line printer daemon has moved to a new directory.

  The print services on Tru64 UNIX and ULTRIX systems are controlled by the line printer daemon (`lpd`). On Tru64 UNIX systems, `lpd` is stored in the `/usr/lbin/lpd` directory by default. On ULTRIX systems, `lpd` is stored in the `/usr/lib` directory.

- The script that starts `lpd` has moved to a new directory.

  When you reboot a Tru64 UNIX system, the system runs the `/sbin/rc3.d/S65lpd` script file to start `lpd`. On an ULTRIX system, `lpd` is started by the `/etc/rc` file at boot time.

- The name of the spooling directory has changed.

  On a Tru64 UNIX system, files to be printed are stored in a spooling directory. By default, the directory is named `/var/spool/lpd/`*`printername`*, where *`printername`* is the name of the printer. You can change the default spooling directory on a Tru64 UNIX system by using the `lprsetup` utility.

- Most ULTRIX print filters are available on Tru64 UNIX systems.

  On ULTRIX systems, print filters are stored in the `/usr/lib/lpdfilters` directory. On Tru64 UNIX systems, they are stored in the `/usr/lbin` directory. The Tru64 UNIX system supports the following print filters:

la75of	LA75 dot matrix printer filter
lg02of	LG02 matrix line printer filter (serial only)
lg031f	LG31 matrix line printer filter
lg06of	LG06 matrix line printer filter (serial only)
lj250of	LF250 companion color printer filter
ln03of	LN03 (S) laser printer filter
ln03rof	LN03R ASCII to PostScript translation filter
ln03rof_isolatin1	LN03R ASCII to PostScript translation filter with ISO Latin/1 encoding vectors
ln03rof_decmcs	LN03R ASCII to PostScript translation filter with the DEC Multinational character set encoding vectors
ln05of	LN05 (S) laser printer filter
ln05rof	LN05R ASCII to PostScript translation filter
ln05rof_isolatin1	LN05R ASCII to PostScript translation filter with ISO Latin/1 encoding vectors
ln05rof_decmcs	LN05R ASCII to PostScript translation filter with the DEC Multinational character set encoding vectors
ln06of	LN06 (S) laser printer filter
ln06rof	LN06R ASCII to PostScript translation filter
ln06rof_isolatin1	LN06R ASCII to PostScript translation filter with ISO Latin/1 encoding vectors
ln06rof_decmcs	LN06R ASCII to PostScript translation filter with the DEC Multinational character set encoding vectors
ln07of	LN07 (S) laser printer filter
ln07rof	LN07R ASCII to PostScript translation filter
ln07rof_isolatin1	LN07R ASCII to PostScript translation filter with ISO Latin/1 encoding vectors
ln07rof_decmcs	LN07R ASCII to PostScript translation filter with the DEC Multinational character set encoding vectors
ln08of	LN08 (S) laser printer filter
ln08rof	LN08R ASCII to PostScript translation filter
ln08rof_isolatin1	LN08R ASCII to PostScript translation filter with ISO Latin/1 encoding vectors

`ln08rof_decmcs`	LN08R ASCII to PostScript translation filter with the DEC Multinational character set encoding vectors
`lpf`	General-purpose line printer filter for the LA75, LA100, LA120, and LA210 printers
`lqf`	Letter-quality printer filter

- The following `printcap` options are available in ULTRIX and UWS, but are not available on Tru64 UNIX:
  - `ps`, printer type
  - `Tr`, Postscript trailer page
- The following DEClaser PostScript printer options are available on ULTRIX and UWS, but are not available on Tru64 UNIX:
  - `–N`, number up
  - `–X`, number of copies
  - `–Z`, print selected pages
- The following DEClaser non-PostScript printer options are available on ULTRIX and UWS, but are not available on Tru64 UNIX:
  - `–X`, number of copies
  - `–Z`, print selected pages

4.7 Terminal Capability Handling

The Tru64 UNIX system supports the `termcap` and `terminfo` mechanisms for describing terminal capabilities in essentially the same manner as on the ULTRIX system. These generic terminal-handling mechanisms are broken down into the following two parts:

- A database that describes the capabilities of each supported terminal
- A subroutine library that allows programs to query that database and make use of the capability values it contains

This section describes database capabilities. Section 7.7 discusses using the `curses` and `termcap` libraries.

The `termcap` capabilities in Tru64 UNIX are comparable to those in BSD 4.3-5. The `terminfo` capabilities are comparable to those in System V Release 3.0 (SVID 2). Tru64 UNIX `termcap` and `terminfo` databases support the following terminals:

VT52	VT220	VT330
VT100	VT240	VT340

VT102	VT241	VT400
VT125	VT300	VT420
VT200	VT320	Xterm

In addition, these databases support a number of common generic devices, including:

ansi	lpr	plugboard
arpanet	network	pmconsole*
bussiplexer	minansi*	printer
dialup	mransi*	switch
dumb	patchboard	unknown
ethernet		

All entries contain only 7-bit control codes. Names marked with an asterisk (*) are in the terminfo database only.

The termcap file is located in the /usr/share/lib directory; the /etc/termcap file is a link included for ULTRIX compatibility. The terminfo database is located in the /usr/share/lib/terminfo directory instead of in /usr/lib/terminfo as on ULTRIX systems.

The terminfo database sources are also located in /usr/share/lib/terminfo instead of in /usr/src/usr.lib/terminfo.

4.8 Disk and File System Maintenance Features

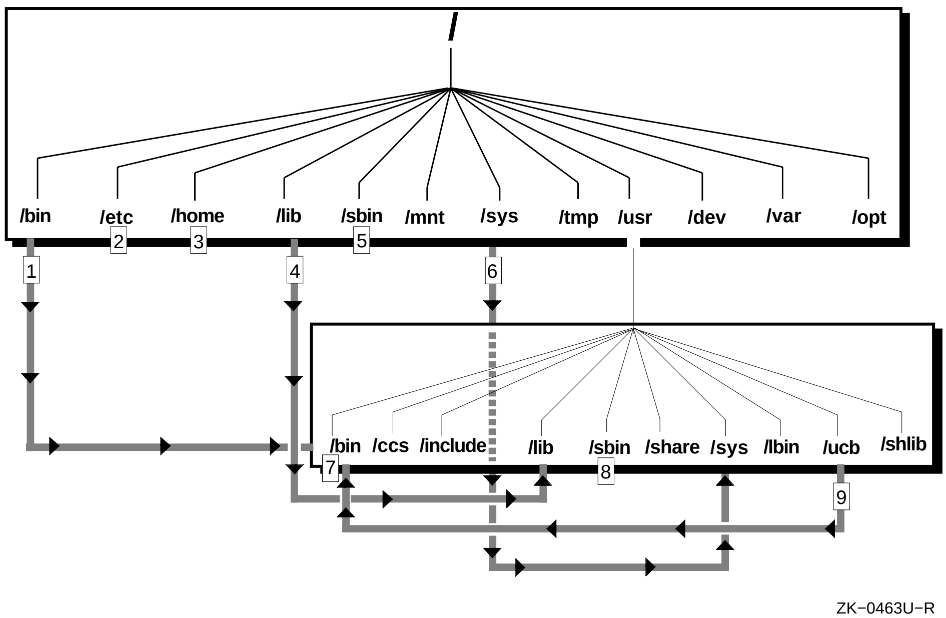
Basic maintenance of disks is similar on a Tru64 UNIX system and an ULTRIX system. Both systems support the UNIX File System (UFS) and the Network File System (NFS). For information about configuring your type of file system (UFS or NFS), see the *System Administration* manual.

Most commands you use to manage disks are the same on a Tru64 UNIX system as they are on an ULTRIX system. This section compares disk and file system maintenance on the two systems, and points out differences.

4.8.1 Tru64 UNIX Directory Structure

The directory hierarchy on a Tru64 UNIX system is different from that on an ULTRIX system. Figure 4–1 shows many of the directories in the Tru64 UNIX directory structure.

Figure 4–1: Tru64 UNIX Directory Structure for System Administrators



As Figure 4–1 shows, many of the directories in the Tru64 UNIX file system structure are identical to the ULTRIX file system structure. The following list points out important differences:

1. On Tru64 UNIX systems, the `/bin` directory is a link to the `/usr/bin` directory.
2. Many system administration commands have moved out of the `/etc` directory and into either the `/sbin` or `/usr/sbin` directory.

   The `/etc/ifconfig` command is linked symbolically to `../sbin/ifconfig`.
3. The Tru64 UNIX directory structure contains the `/home` directory, intended as a root for users' home directories. However, on Tru64 UNIX systems, the home directories for most users are subdirectories of the `/usr/users` directory, which is the default location for adding a user (typically, with the `adduser` command). The actual location of user subdirectories is at the discretion of the system administrator.
4. The `/lib` directory is a link to the `/usr/lib` directory. In addition, the `/usr/lib` directory contains links to libraries stored in the `/usr/ccs/lib` directory.
5. The `/sbin` directory contains the set of executables required to boot and initialize the system successfully in single-user mode. When you are in single-user mode, you can use only the commands in the `/sbin`

directory because shared libraries are unavailable. The commands in the /sbin directory are not linked with shared libraries.

Note

The /sbin directory contains only a subset of the commands that are available on an ULTRIX single-user mode system. You can do less on a Tru64 UNIX system from single-user mode than you can on an ULTRIX system.

6 The /sys directory is a link to the /usr/sys directory.

7 The /usr/bin directory contains binaries and links to binaries in other directories, such as /usr/ccs/bin.

8 The /usr/sbin directory contains commonly used system administration commands. The commands in this directory are linked with shared libraries. When the system is in multiuser mode, you should use the commands in /usr/sbin directory, rather than the commands in the /sbin directory.

9 The /usr/ucb directory is a link to the /usr/bin directory on Tru64 UNIX systems.

4.8.2 Differences in Creating a UNIX File System

To create a UNIX File System (UFS) on a Tru64 UNIX system, you use the newfs command. This command builds a new file system on a specific device, using information in the disk label as its default values. If there is no disk label, newfs uses information from the /etc/disktab file. Compaq recommends that you create disk labels with the disklabel command before running the newfs command. (See disklabel(8) for more information.) You can specify options to redefine the standard sizes for the disk geometry.

The newfs command is similar on Tru64 UNIX and ULTRIX systems. The Tru64 UNIX command omits the –v option. For more information about newfs, see newfs(8).

4.8.3 Differences in Checking a UNIX File System

To check the integrity of a UNIX File System (UFS), use the fsck command. The fsck command checks the integrity of UFS file systems. This command can determine the type of a particular file system by using information in the /etc/fstab file. Alternatively, you can specify options on the fsck command line to indicate what type of file system you are checking. The following table describes differences between the ULTRIX and Tru64 UNIX fsck command:

ULTRIX fsck Command	Tru64 UNIX fsck Command
Repeats the checking operation if it makes repairs to the file system.	Does not perform this rescanning operation.
Has file system clean byte aging, which forces the file system to be checked with fsck periodically.	Does not have clean byte aging; you should run fsck on all file systems periodically, even though fsck says the file system is clean. Use fsck –o to force checking.

For more information about fsck, see fsck(8).

4.8.4 Differences in Mounting and Unmounting a File System

You mount and unmount file systems on a Tru64 UNIX system by using the mount and umount commands. Like the ULTRIX mount command, the Tru64 UNIX mount command mounts the file system you specify or file systems described in the fstab file. The mount and umount commands are similar on Tru64 UNIX systems and ULTRIX systems. For more information, see mount(8). You can mount an ULTRIX file system on a Tru64 UNIX system as described in Section 5.1.

Note

You cannot mount a file system with a 4 kB block size on a Tru64 UNIX system. If you have any data that you need to access and the data is on auxiliary disks in a file system with a 4 kB block size, you must dump the disk to tape or to a disk that has a file system created with an 8 kB block size.

The format of the Tru64 UNIX fstab file is different from the format of the ULTRIX file. Like the ULTRIX fstab file, information about each Tru64 UNIX file system is contained on a separate line in the fstab file. The contents and field ordering of the line are different between Tru64 UNIX and ULTRIX systems. On Tru64 UNIX systems, you separate fields on a line with spaces or tabs. On ULTRIX systems, you separate fields by using a colon. See fstab(4) for more information.

4.8.5 Differences in Monitoring File System Use

Use the df and du commands to monitor file systems use. The Tru64 UNIX df command is similar to the ULTRIX df command, except that by default the Tru64 UNIX command displays statistics in 512-byte blocks while the ULTRIX command displays them in units of 1024 bytes. Use the –k option to display statistics in 1024-byte units. The Tru64 UNIX command supports options that are unavailable on an ULTRIX system, including a –t option that allows you to specify that statistics be displayed for a particular file system type. The Tru64 UNIX `du` command is the same as the ULTRIX `du` command, except that the Tru64 UNIX command supports options that are unavailable on ULTRIX systems. For more information about these commands, see `df`(1) and `du`(1).

4.8.6 Specifying Disk Quotas

You can specify file system disk quotas on a Tru64 UNIX system. The steps you take to activate file system disk quotas on a Tru64 UNIX system are similar to those on an ULTRIX system. For information about activating disk quotas, see the *System Administration* manual.

4.8.7 Differences in Setting Up and Maintaining NFS Software

The Tru64 UNIX Network File System (NFS) software is a facility for sharing files in a heterogeneous environment of processors, operating systems, and networks. The NFS software on a Tru64 UNIX system is similar to the NFS software on an ULTRIX system.

Sharing on a Tru64 UNIX system is accomplished by mounting a remote file system or directory on a local system and then reading or writing the files as though they are local. You can use the Tru64 UNIX NFS software to mount remote ULTRIX file systems. You can also use NFS software to mount Tru64 UNIX file systems on an ULTRIX system. However, if there are files greater than 2 gigabytes (GB) in size, the ULTRIX users will be able to perform file operations only on the first 2 GB.

The Tru64 UNIX NFS software supports two versions of the NFS protocol: Version 2 and Version 3. NFS Version 2 protocol limits remote file access to 2 GB, because of the 32-bit file size and offset fields in the protocol. NFS Version 3 protocol does not have this file access limitation. NFS Version 3 protocol supports 64-bit remote file access. Therefore, the maximum file offset that can be accessed by Version 3 clients is 16 exabytes (2**64-1 bytes).

Whether NFS Version 3 or Version 2 protocol is used is transparent to the client: no action needs to be taken. When a Tru64 UNIX Version 3.0 client mounts a file system from a server, it will use the Version 3 protocol if the server supports it. However, the client will use the Version 2 protocol when it mounts a file system from a Tru64 UNIX Version 2.0 (or earlier) server, or is mounting an ULTRIX file system.

To set up the NFS software, you use the `nfssetup` command. This command operates the same on Tru64 UNIX systems as it does on ULTRIX systems.

Like an ULTRIX system, you list the files that you want to export to remote systems in the `/etc/exports` file. This file has the same general format on a Tru64 UNIX system as it does on an ULTRIX system, with some changes in the export options. However, the old ULTRIX export options are accepted. See `exports`(4) for more information.

If you want to have certain NFS file systems mounted automatically when you boot your Tru64 UNIX system, list those file systems in the `/etc/fstab` file. The format of the Tru64 UNIX `fstab` file is slightly different from the format of the ULTRIX file. As in the ULTRIX `fstab` file, information about each Tru64 UNIX file system is contained on a separate line in the `fstab` file. The contents and order of the line are the same on Tru64 UNIX and ULTRIX systems. The difference is that on Tru64 UNIX systems you separate fields on a line with spaces or tabs. On ULTRIX systems, you separate fields by using a colon.

To mount an NFS file system, you enter the Tru64 UNIX `mount` command. You also use this command to display the list of file systems that are currently mounted on the local system. This command is the same as the ULTRIX `mount` command. For more information about this command, see `mount`(8).

You can display information about NFS servers by using the `showmount` command. This command lists all mount points on the remote server, displays the remote hosts current export list, and so on. This command is the same on Tru64 UNIX and ULTRIX systems. For more information about the command, see `showmount`(8).

To get the status of NFS activity, use the `nfsstat` command as you do on an ULTRIX system. For more information about this command, see `nfsstat`(8).

As on ULTRIX systems, the following four daemons implement the Tru64 UNIX NFS service:

- `portmap`

  The `portmap` daemon maps the remote procedure call (RPC) program numbers of network services to their Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) port numbers. This daemon is similar on Tru64 UNIX and ULTRIX systems.

  Like the ULTRIX `portmap` daemon, the Tru64 UNIX `portmap` daemon supports port checking. Port checking ensures that file access requests were generated by an authorized client kernel, rather than by an unauthorized application program.

- `mountd`

  The `mountd` daemon checks the access permission of the client and returns a pointer to the file system or directory that is to be mounted. The `mountd` daemon is similar between Tru64 UNIX and ULTRIX systems. The difference is that, by default, on a Tru64 UNIX system, the daemon services requests only from the superuser of a remote system.

The ULTRIX mountd daemon services requests from any user on the remote system.

Section 5.8 describes how to configure the mountd daemon so that it runs like the ULTRIX daemon.

- nfsd

  The nfsd daemon allows access to the NFS mounted file system. This daemon is the same on Tru64 UNIX and ULTRIX systems.

- nfsiod

  The nfsiod daemon allows clients to read ahead and write behind to NFS mounted file systems. This daemon is the same as the ULTRIX biod daemon.

4.8.8 Differences in Partitioning Disks

Like ULTRIX disks, Tru64 UNIX disks are divided into partitions. Disk partitions are logical divisions of a disk that allow you to put files of the same type into separate areas of varying sizes. Partitions have default sizes that depend on the type of disk; the installation process uses these default sizes unless it finds an ULTRIX partition table on the disk. To specify alternative partition sizes as part of the installation, you must boot the system into standalone mode and use the disklabel command to create a partition table before running the normal installation procedure. After the system is installed, you can change partition sizes with the Tru64 UNIX disklabel command.

The disklabel command reads and writes the disk pack label. The disk pack label contains the partition table for the disk and information about the geometry of the disk. The disk label is located on one of the first sectors of the disk, usually in block 0.

You use the disklabel command to create, modify, and display the label on a disk. This command is the equivalent of the chpt command on ULTRIX systems. For more information about the disklabel command, see disklabel(8).

4.9 Event-Logging Features

The Tru64 UNIX system event-logging and binary event-logging facilities both record information about system events. On Tru64 UNIX systems, the system event-logging facility uses the syslogd daemon to collect the information logged and distribute it; the binary event-logging facility uses the binlogd daemon to collect information. (On ULTRIX systems, the system log daemon is syslogd daemon and the error-logging daemon is the elcsd daemon.) The syslogd daemon can collect and report the messages logged by the various kernel, command, utility, and application programs.

The system logs messages as specified in the /etc/syslog.conf file. This file is different from the ULTRIX error-logging configuration file, /etc/elcsd.conf. You use the Tru64 UNIX /etc/syslog.conf file to specify the parts of the system, or facilities, for which event logging is enabled. Examples of facilities are the kernel, a user process, and the Mail system. The file also specifies the event message severity level, and the location of the log file to which messages are written.

On the Tru64 UNIX system, the system event-logging facility writes its output to a number of log files, often one file for each facility being logged. You can specify that the system event-logging system create the log file on the local system or a remote system. In most cases, the remote system can be any system that runs the syslogd daemon, including an ULTRIX system. However, the log file for the binary event-logging facility, which logs binary errors, must reside on a local or remote Tru64 UNIX system. Also, you cannot log errors from an ULTRIX system on a Tru64 UNIX system.

For more information about error logging, see the *System Administration* manual.

4.10 Disk Shadowing Facilities

Although the ULTRIX system has no mechanism for replicating data, Compaq offers a separately licensed ULTRIX product that replicates data, called ULTRIX Disk Shadowing. The ULTRIX Disk Shadowing product is not available on Tru64 UNIX systems. To replicate data on a Tru64 UNIX system, use the Logical Storage Manager (LSM) subsystem.

4.10.1 Logical Storage Manager

The Tru64 UNIX Logical Storage Manager (LSM) is an integrated, host-based disk storage management tool that protects against data loss and improves disk I/O performance. System administrators use LSM to perform disk management functions including disk concatenation, data mirroring or shadowing, and striping.

LSM builds virtual disks, called volumes, on top of UNIX system disks. LSM permits dynamic reconfiguration of its disk volumes, making it easy to adapt to changes in I/O load and application needs, and to maximize system availability. LSM features a high degree of flexibility in the way volumes can be mapped to disk and partition devices. This flexibility allows you to optimize performance, change volume size, add mirrors, and perform backups or other administrative tasks without interrupting system applications and users.

LSM includes a command-line interface, a menu interface, and a windows-based interface that a system administrator can use to transparently optimize I/O performance, change volume size, add plexes, and perform backups or other administrative tasks.

Migration information is contained in the *Logical Storage Manager* manual.

4.10.2 Logical Volume Manager

To replicate data on earlier versions of Tru64 UNIX systems, the Logical Volume Manager (LVM) subsystem was used. This subsystem has been retired in favor of the LSM subsystem.

Note

The LVM subsystem and the ULTRIX Disk Shadowing product use incompatible on-disk metadata formats. Consequently, you cannot mount an existing ULTRIX shadowed file system on an LVM mirrored logical device without converting.

Table 4–2 shows some of the differences between the ULTRIX Disk Shadowing product and the LVM subsystem.

Table 4–2: Differences in Disk Shadowing Facilities

	ULTRIX Disk Shadowing	LVM
Description	A layered product that enables you to replicate data on disk partitions.	A kernel subsystem that enables you to create and manage logical volumes. Additionally, you can replicate data and create logical volumes that span multiple disks.
Partitions	Supports `root`, `swap`, and data partitions	Supports data partitions, but does not support `root` and `swap` partitions
Metadata Size	4kB	70kB–4MB
Terminology	Disk shadowing	Data mirroring
	Shadow device	Logical volume
	Disk partition	Physical volume
	Metadata	Metadata
	Shadow set	Set of physical volumes used in a mirrored logical volume
	Two-member shadow set	Single mirrored data

Table 4–2: Differences in Disk Shadowing Facilities (cont.)

ULTRIX Disk Shadowing	LVM
Three-member shadow set	Double mirrored data
None	Physical extent (contiguous disk region)
None	Logical extent (contiguous logical region that maps to 1, 2, or 3 physical extents)
None	Volume group (set of physical and logical volumes)

For information on migrating shadowed data from an ULTRIX system to a Tru64 UNIX system, see Section 5.2.

4.11 Networking Support

The Tru64 UNIX system includes the following networking support:

- Transmission Control Protocol/Internet Protocol (TCP/IP) software and associated applications, such as `telnet`, Berkeley remote commands and utilities, and Simple Network Management Protocol (SNMP) Agent software
- DECnet software

  You can install and use the DECnet software and any of its related software on a Tru64 UNIX system.
- Socket interface (both BSD 4.3 and BSD 4.4) and X/Open Transport Interface (XTI) to TCP/IP
- STREAMS mechanism to support development of network services and data communications drivers

The Tru64 UNIX system does not support the packet filter pseudodevice driver.

The following section gives an overview of the Tru64 UNIX Internet network environment by describing similarities and differences from the ULTRIX environment.

4.11.1 TCP/IP Network Management Commands

When you manage a TCP/IP host, you use a number of commands to set up, determine the status of, and modify network parameters. This section gives an overview of some of the commonly used commands and explains how the commands differ from their ULTRIX equivalent commands. For a list of all command differences between ULTRIX and Tru64 UNIX, see Appendix A.

The following list describes commonly used network management commands:

- `arp`

  The `arp` command displays and modifies Address Resolution Protocol tables.

  This command is the same on Tru64 UNIX and ULTRIX systems, except that on a Tru64 UNIX system the `arp` command does not support the reading of a core file.

- `ifconfig`

  The `ifconfig` command displays and configures network interface parameters.

  This command is the same on Tru64 UNIX and ULTRIX systems.

- `hostid`

  The `hostid` command displays the identifier of the local host.

  This command is the same on Tru64 UNIX and ULTRIX systems.

- `MAKEHOSTS`

  The `MAKEHOSTS` command is unavailable on Tru64 UNIX systems.

- `netsetup`

  You use the `netsetup` command to add your system to a local area network (LAN).

  This command is the same as the ULTRIX `netsetup` command, except that on a Tru64 UNIX system, the `netsetup` command has additional features and a different interface.

- `netstat`

  The `netstat` command displays network statistics, such as interface counters, protocol counters, and routing information.

- `netx`

  This command is unavailable on Tru64 UNIX systems. The Tru64 UNIX system does not supply network exerciser software.

- `ping`

  The `ping` command sends Internet Control Message Protocol (ICMP) ECHO_REQUEST packets to network hosts.

  The `ping` command is the same on Tru64 UNIX and ULTRIX systems, except that on a Tru64 UNIX system, the `–l` option causes the command to send a specified number of packets. On an ULTRIX system, this option causes the `ping` command to display long output. The Tru64 UNIX `ping` command displays long output by default (verbose mode is on).

You can use the `ping` command on a Tru64 UNIX system to get information about an ULTRIX system; also, you can enter the command on an ULTRIX system to get information about a Tru64 UNIX system.

- `.rhosts` and `/etc/hosts.equiv`

  The Tru64 UNIX system does not support `–host`, `+@group`, or `–@group` syntax.

- `rdate`

  The `rdate` command sets the current system date and time to the network date and time.

  The `rdate` command is the same on Tru64 UNIX and ULTRIX systems. (You can also use the Network Time Protocol (NTP) and Time Synchronization Protocol (TSP) to synchronize your system time. For information about NTP and TSP, see Section 4.15.3.)

- `screend`

  The `screend` daemon is the same on Tru64 UNIX and ULTRIX systems. This daemon instructs the kernel to accept or reject IP packets during forwarding, depending on how it is configured.

- `screenmode`

  The `screenmode` command is the same on Tru64 UNIX and ULTRIX systems. This command enables or disables packet screening by the kernel.

- `screenstat`

  The `screenstat` command is the same on Tru64 UNIX and ULTRIX systems. This command displays statistics about kernel packet screening.

4.11.2 Simple Network Management Protocol Agent

The Tru64 UNIX system employs the `snmpd` daemon as a Simple Network Management Protocol (SNMP) Agent. Like an ULTRIX system, the Tru64 UNIX system can be managed by a Network Management Station (NMS) using SNMP. No changes are required for the NMS software to manage a Tru64 UNIX system.

The Tru64 UNIX system does not include the ULTRIX Extended SNMP Agent. Because this software is unavailable, you cannot define a private Management Information Base (MIB) on a Tru64 UNIX system.

In addition, the MIB on the Tru64 UNIX system is an extended version of the MIB on an ULTRIX system. The MIB contains all the variables that are defined on an ULTRIX system and some new variables. The Tru64 UNIX

MIB implements the Internet MIB-II standard and the proposed Fiber Distributed Data Interface (FDDI) MIB Version 1.1 standard.

You set up SNMP on a Tru64 UNIX system by using the `snmpsetup` command, just as you do on an ULTRIX system. This command creates the `snmpd.conf` and the `inet_momd.conf` files. The contents of these files differ between Tru64 UNIX and ULTRIX systems. The Tru64 UNIX `/etc/netman/snmpd.conf` file contains only community information. SNMP reads other information, such as interface speed and interface type, from the Tru64 UNIX kernel. The `/etc/netman/inet_momd.conf` file is a new file that contains the system location and system contact variables for the Internet MIB-II standard.

4.12 Local Area Transport

Like an ULTRIX system, the Tru64 UNIX system uses the local area transport (LAT) protocol.

The following list compares the LAT on the Tru64 UNIX system and LAT on an ULTRIX system:

- The configuration file entry has changed.

  On Tru64 UNIX systems, the following is the configuration file entry for LAT: `options LAT`.

- A setup script is available.

  You set up LAT on a Tru64 UNIX system by using the `latsetup` command. This command creates the LAT terminal devices, adds entries into the `/etc/inittab` file, and starts LAT on your system. For more information on setting up LAT, see the *Network Administration* manual.

- The name of the control program has changed.

  On Tru64 UNIX systems, the control program is named `latcp`. In addition to the functions of the ULTRIX control program, `latcp` allows you to delete a service definition or an application port mapping; to specify a static rating, and switch between static and dynamic; to specify different display options; to specify specific adapters for LAT; and to initialize counter information to zero.

- The script that starts LAT has moved to a new directory.

  On Tru64 UNIX systems, the script that starts LAT is in the `/sbin/init.d` directory. For more information, see `lat`(8).

4.13 Diskless Management Services

The Tru64 UNIX system does not include the Diskless Management Services (DMS) software. You cannot configure a diskless Tru64 UNIX system. However, dataless clients are supported starting with Tru64 UNIX Version 3.0 systems. See the *Tru64 UNIX Software Product Description* for more information.

4.14 Remote Installation Services

Like an ULTRIX system, the Tru64 UNIX system includes the Remote Installation Services (RIS) software. However, the RIS software on the Tru64 UNIX system uses the `bootp` protocol instead of the Maintenance Operations Protocol (MOP). This means that Tru64 UNIX systems can only be servers for other Tru64 UNIX systems, but ULTRIX systems can be servers for ULTRIX systems and Tru64 UNIX systems. See the *Sharing Software on a Local Area Network* manual for more information.

4.15 Distributed System Services

The Tru64 UNIX system has many of the distributed system services you are used to using with your ULTRIX TCP/IP network. In particular, the system supports the Berkeley Internet Name Domain (BIND) service, the

Network Information Service (NIS), and the Network Time Protocol (NTP) Time Synchronization Protocol (TSP) time services.

The Tru64 UNIX system does not support the Kerberos authentication service. You cannot use Kerberos for password security, data encryption, or authentication services. It also does not support the Hesiod naming service.

This section gives an overview of the BIND, NIS, and NTP services available on Tru64 UNIX systems.

As does the ULTRIX system, the Tru64 UNIX system has an `/etc/svc.conf` file that determines how your system uses the BIND and NIS services to find host information. You can use the `svcsetup` command to maintain the `svc.conf` file. Because the Tru64 UNIX system does not include the Hesiod name server, you can specify `bind` only in the hosts database entry.

4.15.1 Berkeley Internet Domain Service

Like an ULTRIX system, the Tru64 UNIX system has the Berkeley Internet Name Domain (BIND) service. However, the BIND service on Tru64 UNIX does not include the Hesiod name server. Because Tru64 UNIX systems do not support the Hesiod naming service, you cannot use the BIND service to distribute the following databases on a Tru64 UNIX system:

- `aliases`
- `group`
- `networks`
- `passwd`
- `protocols`
- `rpc`
- `services`

You can use the Network Information Service (NIS) to distribute these databases. See Section 4.15.2 for information about NIS.

Like the ULTRIX BIND service, the Tru64 UNIX BIND service is based on a server/client model. Servers maintain databases of host names and addresses. When client systems require information about a host, they query the resolver file, `resolv.conf`, for the IP address of a BIND server to service their request. The BIND server runs a daemon, `named`, that services the client's requests.

The Tru64 UNIX system has the `bindsetup` command, which allows you to configure your system as a BIND client or server.

The Tru64 UNIX system has the nslookup and nsquery commands to allow you to get host information from BIND. For information about these commands, see nslookup(8) and nsquery(8).

4.15.2 Network Information Services

The Network Informtion Service (NIS) is a distributed database lookup service for sharing information between systems on a network. The Tru64 UNIX NIS supports the network distribution of the following databases:

- aliases
- group
- hosts
- netgroup
- networks
- passwd
- protocols
- rpc (from ONC RPC)
- services

These databases have the same format on a Tru64 UNIX system as they do on an ULTRIX system, with one exception. On a Tru64 UNIX system, only the root account is allowed to have a user identification (UID) of 0. On ULTRIX, other accounts can also have a UID of 0.

You can use the nissetup command on a Tru64 UNIX system to set up NIS interactively. On ULTRIX systems, you used the ypsetup command. This command operates the same on Tru64 UNIX systems as it does on ULTRIX systems, but it has some additional features. You can also set up NIS by manually using the following commands:

- domainname, which sets the name of the current NIS domain
- makedbm, which creates a NIS servers map
- ypxfr, which transfers an NIS map from a server to a local host

You must also start the NIS daemons, such as the ypserv, ypbind, and yppasswdd daemons. The steps you take, daemons you start, and commands you use to set up NIS manually are different on a Tru64 UNIX system. For example, on the Tru64 UNIX system, you edit the /etc/rc.config file by using the /usr/sbin/rcmgr utility to automatically start the NIS daemons when the system boots. On ULTRIX systems, this is done by editing the /etc/rc.local file.

The Tru64 UNIX system also has commands, such as ypcat, ypmatch, and ypwhich, that allow you to get information from NIS. In addition, the system has commands, such as yppasswd and yppush, that allow you to maintain your NIS databases. These commands are the same on Tru64 UNIX systems as they are on ULTRIX systems.

Like an ULTRIX system, the Tru64 UNIX system has a /etc/svc.conf file that determines how your system uses NIS to find information. You can use the svcsetup command to maintain the svc.conf file.

See the *Network Administration* manual for more information on NIS configuration.

4.15.3 Time Services

The Tru64 UNIX system includes the Network Time Protocol (NTP) and Time Synchronization Protocol (TSP) for time synchronization.

NTP allows accurate, dependable, and synchronized time for hosts on both wide area networks (WANs) (like the Internet) and local area networks (LANs). In particular, NTP provides synchronization traceable to clocks of high absolute accuracy, and avoids synchronization of clocks keeping incorrect time.

The time daemon for the Tru64 UNIX NTP is xntpd. This daemon is an implementation of the NTP Version 2 standard as defined by the Internet Request For Comment (RFC) 1119, omitting authentication. The daemon is compatible with Version 1 servers, including the ntpd daemon available on ULTRIX systems. For more information about the daemon, see xntpd(8).

You normally use two commands to set and monitor time for the xntpd daemon. The ntpdate command sets the locale date and time by polling the NTP server you specify to determine the correct time. The ntpq command monitors NTP servers that are running the xntpd daemon. For more information about these commands, see ntpdate(8) and ntpq(8).

The Tru64 UNIX system has the ntpsetup command to help you configure and run the xnptd daemon on a Tru64 UNIX system. For information about setting up NTP, see the *Network Administration* manual.

The Tru64 UNIX system also includes the ntp and ntpdc commands to allow you to monitor ULTRIX systems that run the ntpd daemon. For more information, see ntp(8) and ntpdc(8.)

TSP is the protocol used by the /usr/sbin/timed daemon. In its simplest application, the TSP servers on a broadcast network (for example, an Ethernet) periodically broadcast TSP packets. The hosts on the network elect one of the hosts on the network running TSP as a master. The master then controls further operation until it fails and a new master is elected.

The master collects time values from the other hosts and computes an average. Each host then synchronizes its clock with the master host.

TSP quickly synchronizes all participating hosts, but it does not trace time back to its sources to determine how accurate the time is. Therefore, the time distributed by a TSP host can be incorrect.

The Tru64 UNIX /usr/sbin/timed daemon is the same as the ULTRIX /etc/timed daemon, with one exception. The Tru64 UNIX daemon does not support the –E option. On ULTRIX systems, this option allows you to force the master time server to distribute its local time to the network, while the network time is controlled by an outside agent, such as NTP.

4.16 The sendmail Utility

The sendmail utility is a general-purpose internetwork mail router. It enables you to send mail to other users on the system and to users on other systems. In most cases, the mail, mailx, and mh commands rely on the sendmail utility to parse mail addresses and to resolve system aliases. The Tru64 UNIX sendmail utility is the same as the ULTRIX sendmail utility, except for the following differences:

- The location of the local aliases file has changed.

  You can specify local aliases on a Tru64 UNIX system, just as you did on an ULTRIX system. The aliases file on a Tru64 UNIX system is /var/adm/aliases; on an ULTRIX system it is in /etc/aliases.

  You can copy your ULTRIX aliases file to a Tru64 UNIX system. For example, enter a command like the following on a Tru64 UNIX system to copy an ULTRIX aliases file:

  ```
  # rcp ultsys:/etc/aliases /var/adm/sendmail/aliases
  ```

  Once you copy the aliases file to the Tru64 UNIX system, enter the newalises command as shown:

  ```
  # newaliases
  ```

  This command builds a new copy of the alias database.

4.17 The uucp Utility

The Tru64 UNIX system has the uucp utility for copying between UNIX systems. The uucp utility allows you to transfer data from one system to another, and to execute commands on a remote system. Connections using the uucp utility can handle data communication over a wider geographic area than a LAN and usually transmit the data through telephone connections.

The uucp utility on Tru64 UNIX systems is different in some ways from the uucp on ULTRIX systems. On Tru64 UNIX systems, the uucp utility is the

HoneyDanBer uucp. (The name HoneyDanBer is derived from the names of the authors of this version of uucp, Peter Honeyman, David A. Nowitz, and Brian E. Redman.) Also, uucp communications is supported over the TCP/IP protocol.

On both systems, you use the uucpsetup command to set up the uucp utility. The Tru64 UNIX command is similar to the ULTRIX command, except that it has been modified to be consistent with the Tru64 UNIX version of uucp. For information about using the uucpsetup utility, see the *Network Administration* manual.

The files that store uucp information and the scripts that control uucp on a Tru64 UNIX system are in different locations and, in some cases, have a different format from the files and scripts on an ULTRIX system. The following list details the differences:

- System information file

  Information about which systems uucp calls out to is stored in the /usr/lib/uucp/Systems file on Tru64 UNIX systems, rather than the /usr/lib/uucp/L.sys file. The format of the Tru64 UNIX Systems file is different from the L.sys file on ULTRIX systems.

- Device information file

  On Tru64 UNIX systems, the file that stores device information is the /usr/lib/uucp/Devices file. On ULTRIX systems, device information is stored in the /usr/lib/uucp/L-devices file. The format of the Devices file is somewhat different from the format of the L-devices file.

- Security information file

  On Tru64 UNIX systems, the /usr/lib/uucp/Permissions file stores information about which systems can access the local system and about which commands can be executed locally. The Permissions file allows you greater control (than you had on an ULTRIX system) over how individual systems can access the local system.

  On an ULTRIX system, information about systems that can access the local system is stored in the /usr/lib/uucp/USERFILE file, and information about which commands can be executed remotely is stored in the /usr/lib/uucp/L.cmds file.

- System polling script

  The Tru64 UNIX system has the Poll file, a script that polls named systems at certain intervals. This file is similar to the LIST.DAY, LIST.HOUR, LIST.LONGHALL, LIST.NIGHT, and LIST.NOON files in the /usr/lib/uucp directory on an ULTRIX system.

- Daemon startup script

The Tru64 UNIX system has the /var/spool/cron/crontabs/uucp file to start up uucp daemons. This file starts the uudemon.admin, uudemon.cleanu, uudemon.hour, and uudemon.poll daemons.

On ULTRIX systems, uucp daemons are started by the /etc/crontab file.

- Log files

  The Tru64 UNIX system has the /usr/spool/uucp/.Admin/errors file, which is equivalent to the /usr/var/spool/uucp/ERRLOG file on ULTRIX systems.

  The Tru64 UNIX system has log files for the uucp, uucico, uux, and uuxqt utilities. Each utility maintains a separate log for each system with which you communicate. The file names are:

  – /usr/spool/uucp/.Log/uucp/*system_name*
  – /usr/spool/uucp/.Log/uucico/*system_name*
  – /usr/spool/uucp/.Log/uux/*system_name*
  – /usr/spool/uucp/.Log/uuxqt/*system_name*

  The log files are equivalent to the /usr/var/spool/uucp/LOGFILE file on ULTRIX systems.

  The Tru64 UNIX system has the /usr/spool/uucp/.Admin/xfer-stats file, which is equivalent to the /usr/var/spool/uucp/SYSLOG file on ULTRIX systems.

- Directories

  The Tru64 UNIX system has the /usr/spool/uucp/.Xqtdir directory, which is equivalent to the /usr/var/spool/uucp/.XQTDIR directory on ULTRIX systems.

  The Tru64 UNIX system has the /usr/spool/uucp/.Status/*system_name* directory, which is equivalent to the /usr/var/spool/uucp/STST directory on ULTRIX systems.

  The Tru64 UNIX system has the /usr/spool/uucp/.Workplace directory, which is equivalent to the /usr/var/spool/uucp/TM directory on ULTRIX systems.

  The Tru64 UNIX system has the /usr/spool/uucp/*system_name* directory, which is equivalent to the /usr/var/spool/uucp/sys directory on ULTRIX systems.

For information about managing the uucp utility on Tru64 UNIX systems, see the *Network Administration* manual. Also, see Appendix A for a list of commands not supported by the Tru64 UNIX system.

4.18 The tip and cu Utilities

In the Tru64 UNIX system, `tip` and `cu` are separate utilities, using separate configuration files. In ULTRIX systems, `cu` is a front end to the `tip` utility.

The `tip` utility enables you to connect to a remote system. This allows you to work on the remote system as if you logged in directly. In addition, you can transfer files by using the `tip` utility. To configure the `tip` utility, you modify the `/etc/remote`, `/etc/phones`, and `/etc/acucap` files. The `cu` utility enables you to connect directly or indirectly to a remote system. This gives you capabilities similar to the `tip` utility, including the ability to transfer files. To configure the `cu` utility, you modify the `uucp` configuration files in the `/usr/lib/uucp` directory.

5 Migrating Your ULTRIX System and Network Environment

This chapter describes how to set up a Tru64 UNIX system for maximum compatibility with ULTRIX systems, and how to migrate file systems from an ULTRIX system to a Tru64 UNIX system. This chapter also discusses the following topics:

- Using the `tar` and `pxtar` commands
- Configuring Small Computer System Interconnect (SCSI) devices
- Setting up internationalization databases
- Configuring the `inetd` daemon for ULTRIX compatibility
- Configuring the `mountd` daemon for ULTRIX compatibility

Note

For information on migrating shadowed data from an ULTRIX system to a Tru64 UNIX Version 3.0 or later system, see the *Logical Storage Manager* manual.

5.1 Mounting an ULTRIX File System on a Tru64 UNIX System

You can mount an ULTRIX File System (UFS) on a Tru64 UNIX system, provided the file system is created with an 8 kB block size and there are partition tables on the disk. The Tru64 UNIX system can read the partition table created by the ULTRIX `chpt` command. Once you mount the ULTRIX file system, you can use it as you normally would. Using an ULTRIX file system on a Tru64 UNIX system does not affect its usability on an ULTRIX system.

To move an ULTRIX file system to a Tru64 UNIX system, follow these steps:

1. If the file system was created with a 4 kB block size, you must dump the disk to tape or to a disk that has a file system created with an 8 kB block size.

2. Install the disk containing the ULTRIX file system onto the Tru64 UNIX system.

3. Check the ULTRIX file system by using the `fsck` command:

```
# /usr/sbin/fsck /dev/rrz0h
** /dev/rz0h
** Last Mounted On
IMPOSSIBLE INTERLEAVE = 0 IN SUPERBLOCK
SET TO DEFAULT ?
```

   The `IMPOSSIBLE INTERLEAVE` message indicates that the Tru64 UNIX system cannot use certain information on the ULTRIX disk. Answer the `SET TO DEFAULT` prompt by typing `yes`, as shown:

```
SET TO DEFAULT ? yes
IMPOSSIBLE NPSECT = 0 IN SUPERBLOCK
SET TO DEFAULT ?
```

   The `IMPOSSIBLE NPSECT` message indicates that the Tru64 UNIX system cannot use certain information on the ULTRIX disk. Answer the `SET TO DEFAULT` prompt by typing `yes`, as shown:

```
SET TO DEFAULT ? yes
** Phase 1 -- Check Blocks and Sizes
** Phase 2 -- Check Pathnames
  .
  .
  .
#
```

   The `fsck` command continues.

   Note

   You receive these messages from the Tru64 UNIX `fsck` command the first time you use the command on an ULTRIX disk. If you use the `fsck` command to check the disk later, these messages do not appear.

4. Create a directory on which to mount the ULTRIX data. The following command creates a directory named `ultrixdata`:

```
# mkdir /ultrixdata
```

5. Mount the file system:

```
# mount /dev/rz0h /ultrixdata
```

Each time you move an ULTRIX disk from an ULTRIX system to a Tru64 UNIX system or from a Tru64 UNIX system to an ULTRIX system, run the `fsck` command. Then, mount the disk. For mounting UFS CD–ROM discs, use the `−d` option to the `mount` command. See `mount`(8) for more information.

5.2 Migrating Shadowed Data

This section describes migration from the ULTRIX Disk Shadowing product to the Tru64 UNIX Logical Volume Manager (LVM) software.

Note

This section does not discuss migration to the Logical Storage Manager (LSM) software on Tru64 UNIX systems. For migration information about LSM, see the *Logical Storage Manager* manual.

Before migrating ULTRIX shadowed data to a Tru64 UNIX system, review the following guidelines:

- The LVM subsystem has a broader management scope than the ULTRIX Disk Shadowing product. Nevertheless, the migration strategy presented in this section only focuses on the disk mirroring aspects of the LVM subsystem. For a complete description of the LVM subsystem, see the *System Administration* manual.
- You must have `root` privilege on the Tru64 UNIX system to mirror data using the LVM subsystem.
- Creating physical volumes, which is an LVM concept, on a raw partition overwrites the existing data on that partition.
- An ULTRIX shadow device can only consist of corresponding partitions on physical disks of the same type. Logical volumes do not have this restriction.
- You cannot migrate shadowed `root` and `swap` partitions to the LVM subsystem.
- Consider the user data size and the metadata size when allocating partitions for LVM physical volumes:
  - User data

    If the existing ULTRIX shadowed partition is nearly full, migrate the data to a larger partition.
  - Metadata

    A Tru64 UNIX system requires more physical space to replicate data than an ULTRIX system requires because the LVM metadata uses more disk space. Use the default LVM parameters for maximum logical volumes, maximum physical volumes, and maximum physical extents in a volume group, which requires approximately 4 MB of additional disk space.

5.2.1 Migration Summary

The following steps summarize the procedure for migrating shadowed data from an ULTRIX system to a Tru64 UNIX system:

1. Dump the ULTRIX shadowed file system to tape. (This is the only step performed on an ULTRIX system.)
2. Label the disks that you intend to use for disk mirroring. If you plan to migrate the shadowed disks, install the disks on the Tru64 UNIX system before labeling.
3. Create and extend a nonmirrored logical volume.
4. Mirror the logical volume.
5. Create a Tru64 UNIX file system on the mirrored logical device.
6. Mount the Tru64 UNIX file system and restore the ULTRIX file system from tape.

Repeat this procedure for each ULTRIX shadowed file system.

5.2.2 Migration Example

The following example demonstrates how to migrate an ULTRIX shadowed file system to corresponding partitions on a Tru64 UNIX system. The resulting migration automatically mirrors data on the Tru64 UNIX system in the same manner that data was shadowed on the ULTRIX system. The elements of this example include:

ULTRIX Disk Shadowing example elements before migration:

File system: `/fs`
Shadow device: `/dev/shd14g`
Disk partitions: `/dev/rz1g` and `/dev/rz2g`
Shadow set: two-member
Disk type: `rz56`

LVM example elements after migration:

File system: `/fs`
Logical volume: `logvolmir`
Volume group: `/dev/vg01`
Physical volumes: `/dev/rz1g` and `/dev/rz2g`
Mirror capacity: single mirrored
Disk type: `rz56`

Use the following example as a guide for migrating your ULTRIX shadowed data:

1. Dump the ULTRIX shadowed file system to tape by entering the following command on your ULTRIX system:

   ```
   # dump 0uf /dev/rmt0h /fs
   ```

   This command copies the entire contents of the `/fs` file system to the `/dev/rmt0h` tape. The command also records the date of the dump in the file `/etc/dumpdates` when the dump is successful.

2. On the Tru64 UNIX system, create a label on the disks you will use for mirroring:

   ```
   # disklabel -r -w rz1 rz56
   # disklabel -r -w rz2 rz56
   ```

   These commands install the standard label on the designated drive. (For more information about initializing disks, see the *System Administration* manual.)

   You can omit this step if you have already installed a label on your disks.

3. On the Tru64 UNIX system, create and extend a nonmirrored logical volume using the following steps:

   a. Create the physical volumes you will use for disk mirroring by entering the LVM `pvcreate` command:

      ```
      # pvcreate /dev/rrz1g
      Physical volume /dev/rrz1g has been successfully created.
      # pvcreate /dev/rrz2g
      Physical volume /dev/rrz2g has been successfully created.
      ```

      This command initializes your direct access storage device for use as a physical volume in a volume group.

   b. Create a volume group directory in the `/dev` directory:

      ```
      # mkdir /dev/vg01
      ```

      Volumes that are mirrored must be in the same volume group. This command creates the directory that identifies the volume group `vg01` for the LVM subsystem.

   c. Create the volume group device file:

      ```
      # mknod /dev/vg01/group c 16 0
      ```

      This command creates the volume group special device file, which is a direct connection between the volume group and the LVM driver code. The volume group special device file must be a character (`c`) device; it must use one of three predefined major device numbers, in this case `16`; and it must have a minor device number of `0`.

d. Create the volume group and populate it with the physical volumes you created with the pvcreate commands:

```
# vgcreate /dev/vg01 /dev/rz1g /dev/rz2g
Creating /etc/lvmtab.
Volume group /dev/vg01 has been successfully created.
```

This command creates the /dev/vg01 volume group that has the members /dev/rz1g and /dev/rz2g. The /etc/lvmtab file contains information that allows the LVM software to access the physical volumes that compose its volume groups after a system reboot.

e. Create the logical volume:

```
# lvcreate -s y -n logvolmir /dev/vg01
A logical volume with name "logvolmir" will be created.
Logical volume "/dev/vg01/logvolmir" has been successfully
created with minor number 1.
```

The lvcreate command creates a logical volume name, logvolmir.

f. Extend the logical volume to encompass all the physical extents of one physical volume. In this example, 63 is the total number of physical extents in the physical volume /dev/rz1g. The vgdisplay command lists the number of physical extents available on each volume.

Specify a logical extent for the logical volume by using the lvextend command:

```
# lvextend -l 63 /dev/vg01/logvolmir /dev/rz1g
Logical volume "/dev/vg01/logvolmir" has been
successfully extended.
```

The -l option extends the logical volume so that it encompasses 63 physical extents. The first argument to the command, /dev/vg01/logvolmir, names the logical volume. The second argument, /dev/rz1g, specifies that the logical extents are assigned to the physical extents on the /dev/rz1g physical device.

4. Mirror the logical volume on the /dev/rz2g device:

```
# lvextend -m 1 /dev/vg01/logvolmir /dev/rz2g
The newly allocated mirror is now being synchronized.
This operation will take some time.
Please wait...
Logical volume "/dev/vg01/logvolmir" has been
successfully extended.
```

The -m option specifies that the system maintains one mirror of the data in logical volume /dev/vg01/logvolmir. The /dev/rz2g argument specifies that the system maintain the mirror using physical extents on the /dev/rz2g physical device.

5. Create a file system on the logvolmir volume by using the newfs command:

```
# newfs /dev/vg01/logvolmir rz56
```

6. Mount the ULTRIX file system on the LVM mirrored logical device and restore the file system from tape:

```
# mount /dev/vg01/logvolmir /fs
# cd /fs
# restore -r
```

The mount command mounts the /dev/vg01/logvolmir logical volume on the /fs directory. The cd command changes the current directory to /fs, and the restore -r command restores the ULTRIX data from tape to the current working directory.

The /fs file system is now converted to Tru64 UNIX LVM disk mirroring.

5.3 Using the tar and pxtar Commands

The ULTRIX system supports two commands for maintaining tape archives: pxtar and tar. The pxtar command is POSIX-compliant; the tar command is not.

The Tru64 UNIX system has one tape archive command, tar. The Tru64 UNIX tar command is POSIX-compliant.

If you use the ULTRIX pxtar command to create a tape archive, you can read that tape archive by using the Tru64 UNIX tar command. In addition, if you use the ULTRIX tar command to create archives that fit on a single volume, you can read those single-volume archives with the Tru64 UNIX tar command.

However, the ULTRIX tar command allows you to create and read an archive that can span multiple tapes. The ULTRIX tar command writes a file header at the start of each continuation tape. By default, the Tru64 UNIX tar command does not expect the ULTRIX header information. The header information is treated as data, resulting in an incorrectly extracted file and the Tru64 UNIX tar command reporting a checksum error. To read an ULTRIX tar archive spanning multiple tapes using the Tru64 UNIX tar command, use the –U option on the Tru64 UNIX system. This option allows the Tru64 UNIX tar command to read tapes and to ignore the header information specific to ULTRIX.

5.4 Configuring Small Computer System Interconnect Devices

During the `doconfig` portion of the installation, the `sizer` program determines what hardware (such as disks and tapes) is attached to your system and reports its findings in the system configuration file.

On ULTRIX systems, `sizer` automatically places 16 Small Computer System Interface (SCSI) device entries (`rz0-rz7` for disks and `tz0-tz7` for tapes) in the system configuration file. This behavior enables you to attach additional SCSI devices at any time without having to rebuild your kernel.

On Tru64 UNIX systems, `sizer` finds only the SCSI devices physically attached to your system at the time of installation and specifies those devices in the system configuration file. For example, if you have an RZ56 as unit 0, a TLZ04 as unit 1, and an RZ24 as unit 2 on your system, `sizer` places only these three devices in your configuration file, as `rz0`, `tz1`, and `rz2`, respectively. If you later add new devices to your system, you must edit the configuration file to include the new devices and rebuild the kernel.

You can save yourself the need to repeat this process by using the `/sys/conf/GENERIC` file as a guide to edit the configuration file to add all possible `rz`*n* and `tz`*n* devices the first time you rebuild the kernel. If you are performing an advanced installation, you can edit the configuration file before the first kernel is built. For information about editing the Tru64 UNIX configuration file and rebuilding the kernel, see the *System Administration* manual.

The RZ57 SCSI disk and TZK10 SCSI tape units are not supported on a Tru64 UNIX system.

5.5 Configuring Tru64 UNIX Shared Memory

Some applications can require you to configure shared memory. Configuring shared memory on a Tru64 UNIX system is done in the same way as on an ULTRIX system, by editing the configuration file and rebuilding the kernel. However, the configuration parameters are slightly different, as shown in the following table:

Parameter on ULTRIX	Parameter on Tru64 UNIX	Remarks
smmax	shmmax	Defines the maximum number of bytes of virtual memory at which a shared memory segment can be sized. The default value is 4 MB on Tru64 UNIX systems. This value is expressed in pages on ULTRIX systems, and expressed in bytes on Tru64 UNIX systems.
smmin	shmmin	Defines the minimum number of bytes of virtual memory at which a shared memory segment might be sized. The default value is 1 MB on Tru64 UNIX systems. This value is expressed in pages on ULTRIX systems, and expressed in bytes on Tru64 UNIX systems.
smseg	shmseg	Defines the maximum number of shared memory segments per process. The default value is 32 on Tru64 UNIX systems.

These Tru64 UNIX defaults are set to values that are common to most layered products. See the *System Administration* manual for information about modifying the configuration file and rebuilding the kernel.

5.6 Setting Up Internationalization Databases

The Tru64 UNIX internationalization features allow you to receive messages and give input in your native language, even when you are in single-user mode. For this feature to operate correctly, you must store message catalogs and locale databases for the /sbin commands in the /etc directory. You must also be sure that the LANG environment variable is defined correctly.

To store message catalogs and locale databases for the /sbin commands in the /etc directory, follow these steps:

1. Translate the message catalogs to the appropriate language, if necessary.

   The message catalogs are stored in the /usr/lib/nls/msg/en_US.88591 directory. Other message catalogs might also be available in subdirectories of the /usr/lib/nls/msg directory if someone has, for example, translated the system catalogs.

2. Create subdirectories in the /etc/nls directory.

   Programs search for the message catalogs in the /etc/nls/msg/*%L* directory, where *%L* represents the currently defined locale. You must create the msg/*%L* subdirectories. For example, suppose you want to use message catalogs for French as it is spoken in Canada. Enter the following commands to create subdirectories:

   ```
   % cd /etc/nls
   % mkdir -p msg/fr_CA.88591
   ```

3. Copy to the /etc directory the message catalogs and locale databases for the language and commands you want to use.

   For example, suppose you want to use French as it is spoken in Canada when you are in single-user mode. Suppose that someone has translated the system-supplied message catalogs and has stored them in the /usr/lib/nls/msg/fr_CA.88591 directory. In this case, you would enter the following cp commands:

   ```
   % cp /usr/lib/nls/loc/fr_CA.88591 /etc/nls/loc/fr_CA.88591
   % cp /usr/lib/nls/loc/fr_CA.88591.en \
   /etc/nls/loc/fr_CA.88591.en
   % cp /usr/lib/nls/msg/fr_CA.8859/* /etc/nls/msg/fr_CA.8859/.
   ```

   The first cp command copies the French-Canadian character database, the second command copies the environment database, and the third command copies the message catalogs. Delete any message catalogs from the /etc/nls/msg/fr_CA.8859 directory that do not correspond to an /sbin command. This frees up space in the root partition.

4. Announce to the system that you want to use the French-Canadian locale when you are in single-user mode. To do this, define the LANG environment variable as follows:

   ```
   % setenv LANG fr_CA.88591
   ```

   You can also set the LANG variable in root's .profile file or shell resource file.

5.7 Configuring the inetd Daemon for ULTRIX Compatibility

Both Tru64 UNIX and ULTRIX systems include the /etc/inetd.conf file, which contains information for the inetd daemon. The inetd daemon is the Internet service daemon.

The Tru64 UNIX inetd.conf file contains a new field. The following list describes the fields in the Tru64 UNIX inetd.conf file:

- *ServiceName*, which names one of the services in the /etc/services file.
- *SocketType*, which is either a stream value or a datagram value.

- *ProtocolName*, which is one of the protocols in the /etc/protocols file.
- *Wait/NoWait*, which determines whether the inetd daemon waits for a datagram server to release the socket. (Stream sockets are always NoWait.)
- *UserName*, which specifies the user name that the inetd daemon should use to start the server.
- *ServerPath*, which specifies the full pathname of the server the inetd daemon should execute.
- *ServerArguments*, which are the command-line arguments passed to the server.

The new *UserName* field allows you to specify what user name inetd should assign to a server when it starts. On ULTRIX systems, servers were automatically started with the root user name. For compatibility, specify root in this field for each service. However, if your server does not need root privileges, consider specifying another user name in this field. As long as your server does not need root privileges, you should not notice a difference between the operation of an ULTRIX server and the operation of a Tru64 UNIX server that is started under a user name other than root.

5.8 Configuring the mountd Daemon for ULTRIX Compatibility

The mountd daemon works with other daemons to provide the NFS service. This daemon checks the access permission of the client and returns a pointer to the file system or directory that is to be mounted by the NFS service.

By default, the mountd daemon on Tru64 UNIX systems accepts requests only from the superuser of a remote system. By contrast, the ULTRIX daemon accepts mount requests from any user.

You can configure the mountd daemon on a Tru64 UNIX system to accept requests from users other than the superuser. To do so, start the daemon with the -n option, as shown:

```
# /usr/sbin/mountd -n
```

This command starts the daemon so that it operates the same as the ULTRIX mountd daemon.

Part 4

Migrating Your Applications

This part gives an overview of the Tru64 UNIX programming environment and describes specific differences between Tru64 UNIX and ULTRIX systems that affect the ways application programs are migrated to a Tru64 UNIX system. This part also gives an overview of how to use shared libraries on a Tru64 UNIX system.

6

Overview of the Tru64 UNIX Programming Environment

The Tru64 UNIX and ULTRIX programming environments are similar and most of the tools in the Tru64 UNIX programming environment are the same as the ULTRIX equivalent tools. However, some differences exist. This chapter is an overview of those differences, in the following categories:

- Alpha architecture
- Graphical programming environment
- Software development tools
- Other programming tools
- Source file control
- Product installation tools
- Shared libraries
- Standard application programming interfaces (APIs)
- Network programming software
- Distributed services programming software
- Internationalization features
- Event-logging software
- Security
- Curses libraries

6.1 Alpha Architecture

To take advantage of the Alpha architecture, the Tru64 UNIX programming environment differs from ULTRIX in the following areas:

- Data representation
- Data access
- Data alignment
- File system

These changes, described in the following sections, can affect how a program accesses and manipulates data.

6.1.1 Data Representation

The Tru64 UNIX C data types have been modified and extended to include a 64-bit type. Table 6–1 shows the differences in data types between the ULTRIX and Tru64 UNIX environments.

Table 6–1: C Language Data Types

Data Type	32-Bit MIPS or VAX System (Size in Bits)	64-Bit Tru64 UNIX System (Size in Bits)
`char`	8	8
`short`	16	16
`int`	32	32
`long`	32	64
`long long`	Not available	64
`float`	32 (MIPS: IEEE single precision)(VAX: F_floating)	32 (IEEE single precision)
`double`	64 (MIPS: IEEE double precision)(VAX: G_floating or D_floating)	64 (IEEE double precision)
`pointer`	32	64

The major differences are that `long` is defined to be 64 bits; `pointer` is defined to be 64 bits, extending the address space; and `long long`, a new data type, is defined to be 64 bits. The `long long` data type offers the unique name for a 64-bit data type that might give additional interoperability between 32-bit and 64-bit systems.

Like the VAX and MIPS systems, the Tru64 UNIX system uses right-to-left byte ordering (little endian) for integer types.

6.1.2 Data Access

Unlike the VAX and MIPS architectures, which allowed byte and word memory accesses, the Alpha architecture supports only memory accesses of longword (32 bits) or quadword (64 bits). Byte and word accesses are accomplished by multiple instructions, which load a longword or quadword, mask, and shift to get the desired entity. The lack of a single operation for byte and word access might produce incorrect results in cases where you are accessing adjacent byte or word entities in shared memory segments.

For instance, a multithreaded application or multiple processes that have access to adjacent byte data through shared memory or shared memory-mapped files will have to use thread mutual exclusion locking functions or semaphore locks, respectively, to avoid conflicts with accesses to adjacent byte or word data items.

Also, the order in which write operations occur can be different from what the programmer intended. If it is important to guarantee the order in which data is written to memory, use memory barrier instructions.

6.1.3 Data Alignment

On both MIPS and Alpha systems the data alignment is implied by the data type. For instance, an `int` (32 bits) is aligned on a 4-byte boundary. On MIPS systems, a `long` (32 bits) is also aligned on a 4-byte boundary. But on Alpha systems, a `long` (64 bits) is aligned on 8-byte boundaries. If you are using assembly language, you will need to understand and code according to these alignment restrictions. If you are using a high-level language, such as C, the compiler will take care of this alignment for you. However, you need to understand these alignment differences when using `long` and `pointer` types in structure definitions that are shared between 32-bit and 64-bit systems.

6.1.4 File Systems

On the 32-bit MIPS and VAX systems, the maximum size of files and file systems was 2 gigabytes (GB). This limit was imposed by the programming interface and file system, which used a 32-bit integer to represent the file offset in bytes (`off_t`) when navigating within a file or file system.

On a 64-bit Alpha system, you can now build much larger files and file systems. The `off_t` file offset is defined to be a `long` on Alpha systems, which is 64 bits in length. Given this extended capability, it is possible to build files and file systems that cannot be fully accessed by 32-bit systems. You need to keep this in mind when working in a distributed environment where file systems are shared between 32- and 64-bit systems.

6.2 Graphical Programming Environment

The Tru64 UNIX DECwindows Motif windowing environment is based on the industry-standard OSF/Motif Version 1.2.3 graphical user interface, featuring three-dimensional visuals and consistent operation and style. The ULTRIX and UWS OSF/Motif graphical interface is based on Version 1.2.2 of OSF/MOtif. Because OSF/Motif Version 1.2.3 does not include new features, there should be no migration issues between the ULTRIX and UWS and Tru64 UNIX system programming environments. However, the

DECwindows XUI interface available on ULTRIX systems is different from the Tru64 UNIX interface. The following sections discuss these differences.

The DECwindows Motif programming environment provides libraries and tools for developing graphical applications for workstations. This graphical programming environment includes:

- Xlib

  This is the Compaq implementation of the Massachusetts Institute of Technology's X Window System, Version 11, Release 5, library provides low-level routines for performing basic windowing functions such as display; graphics; event handling; and text, font, and cursor manipulation. Compaq has extended Xlib to provide routines for the Display PostScript System. This extension allows applications to display images by calling functions that send PostScript code.

  For more information on programming with Xlib, see the book *X Window System*, published by Digital Press. The Display PostScript System documents are contained in the `/usr/share/doclib` directory.

- X Toolkit (also called the Intrinsics)

  A library of routines that creates and manipulates interface objects called widgets.

  For more information on programming with the X Toolkit, see the book *X Window System*, published by Digital Press.

- Motif Toolkit

  A collection of widgets and gadgets for building Motif applications; similar to the XUI Toolkit for building XUI applications on ULTRIX systems. Includes the User Interface Language (UIL), a presentation description language that simplifies the creation and customization of an applications's user interface. Compaq extends the Motif Toolkit by providing additional widgets for help, color mixing, printing, compound strings, and structural visual navigation.

  For more information about programming with the Motif Toolkit, see the *OSF/Motif Programmer's Guide* manual and the *DECwindows Motif Guide to Application Programming* manual. For more information about Compaq extensions to the Motif Toolkit, see the *DECwindows Extensions to Motif* manual.

Creating graphical applications for the DECwindows Motif environment is similar to creating applications for the XUI environment. Programmers who are experienced in developing XUI applications or who are porting existing XUI applications to Motif should note the following differences:

- Name changes—For widget classes, functions, resources, enumeration literals, callback reasons, compound strings, and fontlists.

See Appendix F for a list of the names for these components.

- Window managers—Motif uses the Motif Window Manager (`mwm`); XUI uses the DECwindows Window Manager (`dxwm`). The window manager provides functions for moving and resizing windows on the workspace. The Motif Window Manager works with the toolkit to manage the operations of windows on the screen.

  Terminology differences exist between XUI and Motif. Window functions such as changing the size, shape, or location of a window can be done as Window menu items in Motif. In XUI, these functions are activated by clicking on window manager buttons or borders.

  For a summary of terminology and windowing differences, see Appendix E.

- Style changes—Menu items that appear in the File, Edit, and Help menus are different in the Motif interface. Motif also provides accelerators for each menu item and provides different default mouse button bindings.

  For a summary of these differences, see Appendix E.

For information about how to design Motif compliant applications, see the *OSF/Motif Style Guide* manual and the *DECwindows Companion to the OSF/Motif Style Guide* manual.

For more complete information about porting XUI applications to Motif, see the *Porting XUI Applications to Motif* manual.

6.3 Software Development Tools

Like ULTRIX systems, Tru64 UNIX systems have a variety of software development tools. You can use these tools as you port applications to a Tru64 UNIX system, as well as when you develop new applications on a Tru64 UNIX system.

This section gives an overview of the following Tru64 UNIX software development tools, highlighting differences from the ULTRIX software development tools:

- C preprocessor
- C compiler
- Linker
- Debugging tools
- Other programming tools, including `ar`, `cflow`, `ctags`, `cxref`, `dis`, `file`, `lex`, `lint`, `make`, `nm`, `odump`, `pixie`, `prof`, `pixstats`, `size`, `stdump`, `strip`, and `yacc`

This section gives an overview of only the Tru64 UNIX C preprocessor and C compiler, because they are part of the Tru64 UNIX product. In addition, other compilers, such as Fortran and Pascal, are available for use on the Tru64 UNIX system.

Note

The ULTRIX RISC programming environment for Version 4.3A and higher systems use the MIPS Version 3.0 compilation system, as does the Tru64 UNIX system. Earlier versions of ULTRIX RISC programming environments were based on the MIPS Version 2.10 compiler.

6.3.1 The C Preprocessor

The C preprocessor (`cpp`) on Tru64 UNIX systems is similar to the preprocessor (`cpp`) on ULTRIX systems. Like the ULTRIX preprocessor, the Tru64 UNIX preprocessor interprets directives, such as `#include` and `#define`. The syntax for specifying directives is the same as the syntax on ULTRIX systems.

The Tru64 UNIX system defines a number of preprocessor symbols. Some of these symbols are different from the equivalent symbol on an ULTRIX system. For information about Tru64 UNIX predefined symbols, see Section 7.4.1.

6.3.2 The C Compiler

Like the ULTRIX C compiler driver, the Tru64 UNIX C driver performs several tasks. You can enter the `cc` command to run the C preprocessor, the C compiler, or the linker. Normally, you use the `cc` command to run all three tools and to compile and link your application. Like most ULTRIX C compilers, the Tru64 UNIX C compiler supports optimizing code. In addition, the compiler supports Tru64 UNIX features, such as linking with shared libraries and creating function prototypes. (For more information about the features and general use of the compiler, see the *Programmer's Guide*.)

For compatibility with ULTRIX compilers, the Tru64 UNIX compiler supports several modes for compiling applications. You choose which mode the compiler operates in by using one of the following command-line options:

Option	Description
–std0	Invokes a mode that compiles C applications as defined by Kernighan and Ritchie (K&R), with some ANSI extensions such as function prototypes. This mode is the default mode.
–std	Invokes a mode that compiles applications according to the ANSI standard. The mode allows certain extensions to the ANSI standard, such as C++ style comments and casting of the left-hand side of an assignment operator.
–std1	Invokes a mode that compiles applications in strict accordance with the ANSI standard.

For information about using these options to compile ULTRIX programs on Tru64 UNIX systems, see Section 7.4.

Many applications written for the ULTRIX programming environment will compile with no changes. However, there are certain behaviors that are present in the ULTRIX system that are not in the default Tru64 UNIX system. Most of these behaviors will not be caught at compile time but will instead cause an application to fail when it is run.

The Tru64 UNIX system is written using a hierarchy of interfaces and definitions. Using the default interface, –D_OSF_SOURCE, applications will be able to make use of all the features specified by the OSF Application Environment Specification (AES). If other specific operating system environments are needed, you can use the following symbols:

- -D_OSF_SOURCE
- -D_AES_SOURCE
- -D_XOPEN_SOURCE
- -D_POSIX_SOURCE (for maximum portability of your application)
- -D_ANSI_C_SOURCE
- -D_BSD

For example, applications needing a fully POSIX-conforming environment should be compiled with the -D_POSIX_SOURCE compiler switch. Applications needing a strict ANSI-conforming environment should be compiled with the -D_ANSI_SOURCE and -std1 compiler switches.

6.3.3 The Linker

In most instances, you can use the compiler to link separate application object files into a single executable application.

As part of the compilation process, compiler drivers call the linker, ld, to combine one or more object files into a single application object file.

The linker's operation is essentially similar on the two systems; the most important difference is that by default the Tru64 UNIX linker links with shared libraries; the ULTRIX system does not support shared libraries. The Tru64 UNIX linker resolves external references, searches libraries, and performs all other processing required to create object files that are ready for execution. The resulting object module can either be executed or can serve as input to a separate `ld` command. (You can invoke the linker separately from the compiler by entering the `ld` command.)

The Tru64 UNIX linker also supports C++ automatic constructors and destructors, and new options.

On Tru64 UNIX systems, you normally use the linker to create shared libraries. For information about using and creating shared libraries, see Chapter 8. To link your application with shared libraries, use the appropriate compiler driver. To inhibit linking with shared libraries, use the driver's `-non_shared` option.

Because the Tru64 UNIX environment is a 64-bit environment, the linker, by default, loads the program text and data in the high 64-bit virtual address space of the process (between 0xFFFFFFFFFFFFFFFF and 0x0000000100000000). As a result, there are no addresses accessible with a 32-bit address. If your source code contains any unintended pointer truncations, they will trap into the kernel and cause a run-time error. You can change this default behavior by using the `–T` or `–D` options to change the text and data segment origin, respectively.

6.3.4 The Debugger

The primary debugging tool on Tru64 UNIX systems is `dbx`, which is a source-level debugger. This debugger is the same tool that is available on ULTRIX systems, and you can use it the same as you used the ULTRIX `dbx`. The differences between the Tru64 UNIX and ULTRIX versions of `dbx` are that the Tru64 UNIX debugger has been enhanced to support debugging applications that are linked with shared libraries.

The ULTRIX window interface to `dbx`, which is `dxdb`, is not supplied on Tru64 UNIX systems. If you develop software in a window environment, you can purchase and install the DEC FUSE product. DEC FUSE is a software development, analysis, and maintenance environment for programmers. DEC FUSE offers a set of tools with a DEC OSF/Motif user interface and graphics options in an integrated setting. DEC FUSE tools include an editor, a code manager, a program builder, a debugger, a cross-referencer, and a call graph browser.

Tru64 UNIX systems also include another debugging tool, `kdbx`. The `kdbx` utility is an interactive, crash analysis and kernel debugging tool that replaces the ULTRIX `crash` program. As a kernel debugging tool, `kdbx` serves as a front end to the `dbx` debugger, and enables you to examine the running kernel or dump files created by `savecore`. The `kdbx` utility is also insensitive to version numbers, and can be customized and extended. For more information on the `kdbx` utility, see the *Kernel Debugging* manual.

6.3.5 Other Programming Tools

Tru64 UNIX systems have other programming tools that are available on ULTRIX. These tools have been modified to support the ANSI C language dialect, shared libraries, and 64-bit data types. Otherwise, their use is the same as their ULTRIX equivalents.

The following list gives a brief description of each tool. (For more information about the tools, see the reference page for each individual tool.)

- `ar`

  Creates and maintains archive libraries. (You cannot use the `ar` command to create shared libraries. To create shared libraries, use the `ld` command as described in Section 8.1.4 and Section 8.1.5.)

- `cflow`

  Analyzes C application files (as well as `yacc`, `lex`, and assembler files) and builds a graph that charts the external references made in the application.

- `ctags`

  Creates a tags file that you can use with the `ex` editor. The tags file specifies the location of functions and `typedef` declarations in the specified set of C application files.

- `cxref`

  Analyzes a set of C application files and builds a cross-reference table. The table lists all the symbols used in the application.

- `dis`

  Disassembles object files into machine instructions.

- `file`

  Reads one or more files as input, performs a series of tests on the files, and determines their types.

- `lex`

  Generates a C language source file that matches patterns for simple lexical analysis of an input stream.

- `lint`

  Checks C application files for coding that is inefficient, not portable, or might cause errors. For example, this command finds unreachable statements, automatic variables that are declared and not used, and logical expressions that have a constant value.

- `make`

  Builds up-to-date versions of application programs. The `make` command updates the application program depending on whether the files used to build the program have changed. The `make` command updates the program only if the files used to build it have changed.

  Tru64 UNIX also includes the ULTRIX `make` command. See Section 7.1 for information on using the ULTRIX `make` command.

- `nm`

  Displays symbol table information for object files and archive files.

- `odump`

  Displays information about an object file, archive file, or executable file. For example, you can use `odump` options to display an object file's header, defined symbols, or program regions.

- `pixie` and `prof` and `pixstats`

  The `pixie` command reads applications, partitions them into basic blocks, and counts the execution of the basic blocks. Use the `prof` command with the `-pixie` option to display `pixie` data. The `pixstats` command analyzes the output from `pixie`. These profiling tools are only supported with archive libraries. They cannot be used with shared libraries. Note that the `pixstats` command produces incorrect results on Tru64 UNIX systems.

- `size`

  Displays the number of bytes required by each section of an object file, as well as the total number of bytes required by the object file.

- `stdump`

  Displays detailed symbol table information for an application or object.

- `strip`

Strips the symbolic debugger information from an executable file.

- `yacc`

  Converts a context-free grammar specification into a set of tables that can be used by a simple parsing program.

6.4 Source File Control

Like the ULTRIX system, the Tru64 UNIX system supports the Source Code Control System (SCCS). The Tru64 UNIX system also supports the Revision Control System (RCS), which is an unsupported subset on the ULTRIX system. The SCCS and RCS utilities allow you to store application modules in a directory, track changes made to those module files, and monitor access to the files. The SCCS and RCS utilities on Tru64 UNIX systems are the same as the SCCS and RCS you use on ULTRIX systems. For more information about SCCS and RCS, see the *Programming Support Tools* manual.

6.5 Product Installation Tools

Once you port your application to Tru64 UNIX, you might want to create a software package for it, for distribution to other users. Like ULTRIX systems, the Tru64 UNIX system has utilities that you can use to install, remove, combine, validate, and configure applications.

To create a software package, you use the following utilities:

- `newinv`

  Processes a master inventory input file. The output of the `newinv` utility is a file that contains a list of all the files that compose your application. The file also contains information about the subset in which each file belongs. The `newinv` utility operates the same on Tru64 UNIX systems as it does on ULTRIX systems.

- `gentapes`

  Produces magnetic tape distribution media (MT9 or TK50). This utility has the same features on Tru64 UNIX and ULTRIX systems. The location of this utility has changed from `/usr/sys/dist` to `/usr/bin` on Tru64 UNIX systems.

- `gendisk`

  Produces disk distribution media. On ULTRIX systems, the name of this utility is `genra`. The features of these utilities are the same. The location of the utility has changed from `/usr/sys/dist` to `/usr/bin`.

- `kits`

Produces subset images, inventories, and control files from the input files that have been transferred from your source directory. The utility also generates data files that make up the media master in the output directory. This utility is the same on Tru64 UNIX and ULTRIX systems.

- `setld`

  Installs software on the user's system. The `setld` command can install software from the following distribution media:

  – Data disks, including CD–ROM optical discs

  – TK50 tapes

  – MT9 tapes

  On a Tru64 UNIX system, the `setld` command resides in the `/usr/sbin` directory. On an ULTRIX system, the command is in the `/etc` directory.

Unlike the ULTRIX `setld` command, the Tru64 UNIX `setld` command does not install software into a Diskless Management Services (DMS) area. The DMS software is not provided on Tru64 UNIX systems.

See the *Programming Support Tools* manual for descriptions of the program installation tools and the process of building `setld`-compatible kits.

6.6 Shared Libraries

The Tru64 UNIX system provides shared libraries as part of the programming environment. Shared libraries are libraries linked in a file organized like a demand-paged executable program. Like other programs, the libraries contain data and text sections and export entry points or data objects. Multiple processes can use the entry points simultaneously or use the data objects (each process has a private copy of the data objects).

Unlike most programs, shared libraries contain no fixed-base address. Shared libraries contain symbol and relocation information. When you link your application with a shared library, the executable application does not contain the library routines; instead, the application contains the information it needs to load the shared library at startup time and to access the shared routines and private data at execution time.

The following shared libraries are elements of all Tru64 UNIX systems:

libDXm.so	libc.so	libids.so
libMrm.so	libc_r.so	libids_nox.so
libX11.so	libchf.so	libimg.so
libXaw.so	libcda.so	libips.so
libXext.so	libdnet_stub.so	liblkwdxm.so

libXie.so	libdl.so	libm.so
libXm.so	libdps.so	libmach.so
libXmu.so	libdpstk.so	libpsres.so
libXt.so	libdvr.so	libpthreads.so
libbkr.so	libdvs.so	libsys5.so

The following shared libraries are also elements of all systems, but were not documented earlier in this manual:

libXimp.so	libXv.so	libaud.so
libcdrom.so	libcmalib.so	libcurses.so
libiconv.so	libmxr.so	libproplist.so
libsecurity.so	libtli.so	libxti.so

Starting with Tru64 UNIX Version 3.0, the following shared libraries are elements of all systems:

libDXterm.so	libXIE.so	libXi.so

These libraries are located in the /usr/shlib directory.

In addition to shared libraries, the Tru64 UNIX system provides archive libraries. Archive libraries are traditional ULTRIX libraries. When you link your application with them, the image for library routines you call is included in your application image. You can link Tru64 UNIX applications to either the new shared libraries or the traditional archive libraries. To help you decide which libraries to use, this section describes some advantages of using shared libraries and some restrictions on using them.

For information on how to link your application with Tru64 UNIX shared libraries and how to create shared libraries, see Chapter 8.

6.6.1 Using Shared Libraries

The following list details the advantages of using shared libraries on Tru64 UNIX systems:

- Disk space savings

  When multiple applications use a shared library, you save disk space. If five applications use the same library image, the library image occurs only once on the disk. By contrast, if you link each process statically with a set of library routines, the image of the library routines occurs five separate times on the disk.

- System memory savings

When multiple processes run applications that are linked with a shared library, you save physical memory. As with disk space, you see the memory savings when multiple applications use the same shared library.

- Reduced paging

  Like other routines, a shared library routine is read into memory the first time a process needs it. Because more than one process can use the image of the shared library routine, the second process that calls it might find the routine already in memory. If several processes are using the same routine, that routine tends to remain in memory. Thus, processes that use shared libraries often require less paging than processes that use archive libraries.

- Better application and system performance

  Using shared libraries improves the performance of your application when multiple applications use the same shared library routines. This situation often occurs on a typical multiuser system when multiple applications are using shared libraries. In addition to improving the performance of individual applications, this situation improves the overall performance of your system.

  However, benchmark applications linked with shared libraries might show a degradation in performance when compared to the same application linked with archive libraries. Benchmark applications normally run on an unloaded system, so your benchmark loses the opportunity to benefit from sharing library routines with other applications. In addition, on an unloaded system, the startup time for an application linked with shared libraries is somewhat slower than the startup time for an application linked with archive libraries. Run-time performance of your benchmark might be slower because references to symbols exported from a shared library are made indirectly. References to symbols in an archive library are made directly. Indirect references are somewhat slower than direct references.

6.6.2 Changing from Archive Libraries to Shared Libraries

Normally, you can use shared libraries in any application and create any library as a shared library. In most cases, the effect of using shared libraries instead of archive libraries should be transparent; however, a few restrictions on using and creating shared libraries do exist. The following list describes these restrictions:

- The `/usr` directory must be mounted when you run an application that is linked with shared libraries.

  If your application is designed to run when the `/usr` directory is not mounted, do not use shared libraries. When you link your application with shared libraries, your application executable does not include the shared library; it includes only information it needs to load the shared library. If the shared libraries are unavailable when you run your application, it fails.

- You cannot use `-O3` or `-O4` optimization options when you link your C application with shared libraries or when you create shared libraries.

  If you want to optimize your C application by using one of these options, you must link with archive libraries. (You might be able to optimize applications written in other languages that you link with shared libraries. For more information about linking applications written in languages other than C with shared libraries, see the documentation for the language you are using.)

- All code must be position-independent code when you create a shared library.

  You must recompile and link your code with a Tru64 UNIX compiler in order to have position-independent code. Assembler code must also be written to be position-independent code, using the rules mentioned in the *Calling Standard for Alpha Systems* manual.

- Do not use profiling with shared libraries.

  The `pixie` and `pixstats` commands are supported only with archive libraries.

- Do not link shared libraries with archive libraries.

  Shared libraries should only depend on other shared libraries. Linking a shared library with an archive library could create conflicting references at run time, causing unpredictable program behavior.

- Applications might need to be modified when linking with shared libraries if they depend on specifics of the ULTRIX call frame or on run-time stack tracing of libraries.

  See the *Calling Standard for Alpha Systems* for specific information on changes.

- The stack version of the `alloca()` function is currently unusable in shared libraries.

6.7 Standard Application Programming Interfaces

In addition to making your source code portable with respect to applicable language standards, you must make your applications conform to specific application programming interfaces (APIs) in order to link correctly and produce correct results. The Tru64 UNIX system supports the following APIs:

- Application Environment Specification (AES)

AES is the specification to which OSF/1 Version 1.0 was built. Applications that use only the interfaces specified by the AES will compile and run successfully on all implementations of OSF/1 Version 1.0 and all compliant platforms.

- POSIX

  POSIX (IEEE Std 1003.1-1990; ISO/IEC 9945-1:1990(E)) describes run-time behavior and provides definitions for programming interfaces. It provides applications with the maximum portability across OSF/1 and other platforms. The Tru64 UNIX system also meets the National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS) 151-1.

- XPG3 Base

  X/Open's XPG3 Base describes the definitions and run-time behavior for a set of interfaces. This standard extends beyond the POSIX standard to cover additional features in the X/Open environment.

- ANSI C

  The ANSI C language standard (ANSI X3.159-1989; ISO/IEC 9899:1990(E)), in addition to specifying a definition for the C programming language, contains definitions for the standard library functions.

- System V and BSD

  System V Release 3 (based on the System V Interface Definition (SVID) 2), System V Release 2, and BSD represent implementation standards and are available for applications that depend upon specific behavior unique to the System V and BSD environments.

  There are areas in which these implementation standard APIs conflict with the more formal standard APIs described earlier. You can resolve these conflicts by using the compiler and linker options described in Section 7.6.2 and Section 7.6.3.

6.8 Network Programming Software

The networking programming facilities available in the Tru64 UNIX system provide a high degree of commonality and interoperability with the ULTRIX system. Both systems provide APIs, including X/Open Transport Interface (XTI), Data Link Interface (DLI), and sockets, as described in the following sections. In addition, Tru64 UNIX provides support for STREAMS, which is compatible with System V Release 3.2 STREAMS.

6.8.1 X/Open Transport Interface

The X/Open Transport Interface (XTI) defines a transport interface for networking applications that is independent of any specific transport provider. The XTI design and implementation on Tru64 UNIX are new. XTI applications are interoperable between ULTRIX and Tru64 UNIX systems. XTI is similar to, and backward compatible with, the System V Transport Layer Interface (TLI). Libraries for both XTI (`-lxti`) and TLI (`-ltli`) are provided. See Section 7.8 and the *Network Programmer's Guide* for more information.

6.8.2 Data Link Interface

The Data Link Interface (DLI) defines a transport interface for networking applications on Ethernet and Fiber Distributed Data Interface (FDDI) networks. DLI applications are interoperable between ULTRIX and Tru64 UNIX systems. On Tru64 UNIX systems, the location of the `dli_var.h` library is `/usr/include/dli/dli_var.h`. In addition, the `sockaddr_dl` structure has a new field, `dli_len`, in the first byte. See the *Network Programmer's Guide* for more information.

6.8.3 Sockets Interface

Sockets are the end points of communication channels and are used much the same way as file descriptors are used. The socket interface provided by Tru64 UNIX is compatible with the ULTRIX socket interface. See the *Network Programmer's Guide* for more information.

6.8.4 SNMP Compatibility

Tru64 UNIX and ULTRIX both support the Simple Network Management Protocol (SNMP) Agent. The Tru64 UNIX system does not support the ULTRIX Extended SNMP Agent for defining private Management Information Base (MIB) objects through a set of library routines. See Section 7.6.1 for more information.

6.9 Distributed Services Programming Software

This section discusses the following distributed services programming facilities:

- Remote procedure calling (RPC)
- Kerberos authentication service
- Berkeley Internet Name Domain (BIND) service
- Network Information Service (NIS, formerly YP)

- Hesiod naming service

6.9.1 Remote Procedure Calling

The ULTRIX system provides a general RPC mechanism, DEC RPC Version 1.0, which is based on and is compatible with the RPC component of the Hewlett-Packard/Apollo Network Computing System (NCS), Version 1.5. Tru64 UNIX systems do not provide a development or run-time environment for DEC RPC Version 1.0. Specifically, the DEC RPC Version 1.0 components, which include the Network Interface Definition Language (NIDL) compiler, the Location Brokers (Local Location Broker and Global Location Broker), and the RPC run-time library, are unavailable.

Tru64 UNIX and ULTRIX systems both provide the capabilities to interoperate with the Sun Microsystems ONC (Open Network Computing environment). Tru64 UNIX systems also enable ONC RPC application development by providing a high-level set of operations that can be used to execute procedures on remote systems across a network. See the *Programming with ONC RPC* manual for more information on using ONC RPC.

As part of the Distributed Computing Environment (DCE), OSF has defined an RPC mechanism that integrates with the other DCE components (for example, Cell Directory Service and Global Naming – X.500). Components to support this mechanism are not part of the base Tru64 UNIX product. See the Compaq DCE Starter Kit for these features.

6.9.2 Network Authentication

Kerberos is an authentication service that validates the identity of a user or service, preventing fraudulent requests. It provides a programming interface for authentication by applications communicating across a TCP/IP network with a socket interface. The ULTRIX system supports a version of Kerberos that is derived from MIT/Athena's Kerberos Version 4. No Kerberos programming interfaces are available on the Tru64 UNIX system. Many binaries that were built with Kerberos on an ULTRIX system will run on Tru64 UNIX systems when using an ULTRIX server, as long as the ULTRIX system is Version 4.2 or higher. See Section 7.6.1 and Section B.18 for more information.

6.9.3 Naming Services

The Tru64 UNIX system supports both the Berkeley Internet Name Domain (BIND) service and the Network Information Services (NIS, formerly YP) service. Both of these services are interoperable between ULTRIX and Tru64 UNIX systems. By itself, the BIND name service allows you to distribute a host naming database. The NIS service can distribute several different databases. BIND and NIS support equivalent functions in the run-time library.

The Hesiod service, available on ULTRIX systems, is not supported by the Tru64 UNIX system. No Hesiod programming interfaces exist on Tru64 UNIX systems. See Section B.18 for more information.

6.10 Internationalization Features

An internationalized application allows users to interact with that application in their native language. The application is also designed to reflect the culture of the user's region. For example, data such as dates and monetary values are displayed or read as input in the style of that region.

The Tru64 UNIX internationalization features are compatible with the ANSI C, POSIX, and XPG4 specifications for an international programming environment.

In addition to the ULTRIX internationalization features, the Tru64 UNIX internationalization features provide the following:

- Support for more locales
- Support for the LC_MESSAGES environment variable
- Internationalized library routines and system commands

Some differences exist between the ULTRIX and Tru64 UNIX internationalization features. The following sections give an overview of the differences. For more information about the Tru64 UNIX internationalization features, see *Writing Software for the International Market* and the `i18n_intro` command.

6.10.1 Message Catalog System

The message catalog system isolates program messages from the body of your program. This isolation makes it easier for you to translate messages into other languages. The message catalog system consists of message extraction tools, tools for translating messages, a tool for generating message catalogs, and routines for accessing message catalogs.

6.10.1.1 Message Extraction Tools (extract, strextract, and strmerge)

Like the ULTRIX system, the Tru64 UNIX system provides the `extract`, `strextract`, and `strmerge` commands that extract message text from your program and store it in a message text source file. In most ways, these commands are the same as their ULTRIX equivalent. This section describes the two ways the commands differ between the Tru64 UNIX and ULTRIX systems.

The output file name for the strextract command, and therefore the input file name for the strmerge command, is different. On Tru64 UNIX systems the intermediate file that strextract creates is named *filename*.str. On ULTRIX systems, this file is named *filename*.msg.

The other difference is the name of the internationalization directory. On Tru64 UNIX systems, related internationalization files are stored in the /usr/lib/nls directory. On ULTRIX systems, these files are stored in the /usr/lib/intln directory. This change affects the following:

- The location of the systemwide patterns file, which on Tru64 UNIX systems is /usr/lib/nls/patterns.
- The location of the help file for the extract command, which on Tru64 UNIX systems is /usr/lib/nls/help.
- The search path for user-specified patterns and ignore files. On Tru64 UNIX systems, the extract, strextract, and strmerge commands search for patterns and ignore files in the current directory, your home directory, and then /usr/lib/nls.

6.10.1.2 Tool for Translating Messages (trans)

You can use the Tru64 UNIX trans command to help you translate message text source files from one native language to another. This command is the same as the ULTRIX trans command.

6.10.1.3 Tools for Creating a Message Catalog (mkcatdefs and gencat)

The Tru64 UNIX system provides the mkcatdefs and gencat commands, which work together to generate a formatted message catalog. Some system limits that affect the gencat command have increased on Tru64 UNIX systems. See Table B–1 for complete information. Other than the difference in the system limits, the Tru64 UNIX gencat command is the same as the ULTRIX gencat command. The mkcatdefs command is the same on both systems.

6.10.1.4 Routines for Accessing a Message Catalog (catopen, catgets, and catclose)

You use the Tru64 UNIX catopen, catgets, and catclose library routines to open, read messages from, and close message catalogs. These routines are the same as the ULTRIX routines of the same names.

By default, these routines search the /usr/lib/nls/msg/%*L* / %*N* path for a message catalog. In the preceding pathname, %*L* represents the locale name specified by the LANG environment variable, and %*N* represents the name of the message catalog passed to the catopen function. Typically, the name of the message catalog is messages.cat.

The Tru64 UNIX catopen routine differs from the ULTRIX catopen routine in two ways. First, the Tru64 UNIX catopen routine does not search the current directory for message catalogs. The ULTRIX catopen routine searches the current directory for message catalogs it does not find in either the /usr/lib/nls/msg/%*L* /%*N* directory or the directories specified by the NLSPATH environment variable. Second, the Tru64 UNIX catopen routine ignores the NLSPATH environment variable when it attempts to find a message catalog for the root user. The routine searches only the /usr/lib/nls/msg/%*L* /%*N* directory. This difference affects applications that use the setuid system call to become the root user.

6.10.2 Program Localization

Writing an international application involves more than isolating and translating program messages. The application must also reflect the culture of the user by displaying dates and times, monetary values, numbers, alphabetic lists, and so on in the style that the user expects. On Tru64 UNIX systems (as on ULTRIX systems), the application behavior in these areas is controlled by the program's locale.

6.10.2.1 Announcement Mechanism

You control which locale an application runs in by defining environment variables. The environment variables announce to the system what local data the application should use.

The Tru64 UNIX system provides the same environment variables as the ULTRIX system, with the addition of LC_ALL and LC_MESSAGES. The following list describes these environment variables:

- LANG controls all categories of an application's locale. However, you can override the setting of LANG by defining one of the environment variables that control a specific category (LC_COLLATE, LC_CTYPE, and so on).
- LC_ALL controls all categories of an application's locale. Unlike LANG, you cannot override the setting of LC_ALL by defining one of the environment variables that control a specific category.
- LC_COLLATE controls the collation category of the application's locale. The collation category affects the operation of the strcoll and strxfrm library routines.
- LC_CTYPE controls the character classification category of the application's locale. This variable affects the operation of the isdigit and isalpha library routines, among others.
- LC_NUMERIC affects the radix and thousands separator character as it is used by the printf and scanf library routines.
- LC_TIME affects the behavior of the strftime library routine.

- LC_MONETARY affects what the strmon library routine returns as the format for monetary values.
- LC_MESSAGES affects the format of application messages and the string the user can specify to answer a yes or no question.

The only difference between these environment variables on a Tru64 UNIX system and on an ULTRIX system is the naming convention used for the locales. For information about defining these environment variables, see Section 3.1.3.

6.10.2.2 The setlocale Routine

To determine what locale has been set, you call the setlocale routine in your program. This routine has the following format:

setlocale (*category, locale*)

The *category* argument specifies the category for which you are requesting locale information, that is, LANG, LC_COLLATE, LC_CTYPE, and so on. The *locale* argument is usually an empty string ("") that causes the setlocale routine to determine the setting of a category by reading the corresponding environment variable. However, you can specify a locale name in this argument. If you do, be aware that the naming convention for the Tru64 UNIX locales is different from the ULTRIX naming convention. For information about the names of Tru64 UNIX locales, see the *Technical Overview*.

By default, setlocale expects the locale-specific data to be in the language support databases contained in the /usr/lib/nls/loc directory (the /usr/lib/intln directory on ULTRIX systems).

On ULTRIX systems, you can store the locale-specific data in a directory that is not in the default search path. You specify where the locale-specific data is by defining the INTLINFO variable. On Tru64 UNIX systems you specify where the locale-specific data is by defining the LOCPATH variable.

Except for these differences, the setlocale routine is the same on Tru64 UNIX and ULTRIX systems.

6.10.3 Creating Locale-Specific Information

On Tru64 UNIX systems, you can create your own locale-specific information. Use the localedef command to process locale and character map files and produce a locale database. This command replaces the ULTRIX ic command. For information about using the localedef command, see localedef(1).

6.10.4 The iconv Command

Like the ULTRIX iconv command, the Tru64 UNIX iconv command converts the encoding of characters in one codeset to another codeset. On Tru64 UNIX systems, you can use iconv to convert between a number of character sets. The system provides conversion tables in the /usr/lib/nls/loc/iconv directory. For information about using iconv to convert codesets, see iconv(1).

6.11 Event-Logging Software

On Tru64 UNIX systems, system events are recorded using two facilities:

- A systemwide event-logging facility, which logs events in ASCII format.
- A binary event-logging facility, which logs hardware and software events in the kernel in binary format records.

The binary event logging is like the binary error logging provided on ULTRIX systems. There are differences between the system logging facilities on ULTRIX and Tru64 UNIX systems. Both the ULTRIX and Tru64 UNIX systems provide a set of application interfaces for syslog. See Section B.16 for more information.

6.12 Security

The *Security* manual contains information about migrating a programming interface from ULTRIX to Tru64 UNIX, including basic migration issues and ways to move ULTRIX authentication files to a Tru64 UNIX system.

The Tru64 UNIX system does not support the functions for getting and setting authorization entries in the auth database. See the discussion of secauthmigrate in the *Security* manual for more information. Additionally, Section B.18 contains a list of security-related header files that exist in ULTRIX but do not exist on a Tru64 UNIX system.

6.13 Curses Libraries

The curses package is a set of cursor optimization routines for writing screen-management programs. ULTRIX and Tru64 UNIX systems both support X/Open and BSD curses library routines. The capabilities in both of these libraries have been extended beyond what is available with Version 4.2 and Version 4.3 of the ULTRIX system. For example, the Tru64 UNIX curses package provides multibyte character support.

ULTRIX applications will need to change references to header files and libraries. See Section 7.2, Section 7.7, and Section 4.7 for more information.

7 Migrating Your ULTRIX Application to a Tru64 UNIX System

The best way to move your application from an ULTRIX system to a Tru64 UNIX system is to migrate your source code to the Tru64 UNIX system. When you port source code, the result is a native Tru64 UNIX application that is easy to move to new versions of Tru64 UNIX and new platforms. In addition, you can take advantage of Tru64 UNIX features, such as 64-bit data types and addressing and shared libraries.

This chapter describes the tasks you perform to migrate source code from an ULTRIX to a Tru64 UNIX system after you have transported the source files to the Tru64 UNIX system by using `rcp`, `ftp`, or `uucp` commands, `tar` archives, Network File System (NFS) mounting, or any other appropriate method. This chapter also gives information about ULTRIX header files that are not supplied on a Tru64 UNIX system, differences in using the C compiler on an ULTRIX and a Tru64 UNIX system, and ULTRIX function libraries that are not supplied on a Tru64 UNIX system.

7.1 Modifying Your Makefile

To allow you to conveniently build your application on a Tru64 UNIX system, modify your makefile so that it works on the Tru64 UNIX system. The following list describes differences between Tru64 UNIX and ULTRIX systems that could affect your makefile:

- The `s5make` command is unsupported on the Tru64 UNIX system. Remove references to that command and replace them with the `make` command.
- The Tru64 UNIX directory structure is different from the ULTRIX directory structure. This difference might require you to modify pathnames in your makefile.
- Changes in command options could require changes to your makefile. For information about differences between ULTRIX and Tru64 UNIX command options, see Appendix A.
- Differences in how system libraries are organized could require changes to your makefile. For information about differences, see Section 7.6.

- Differences between ULTRIX and Tru64 UNIX header files and routine definitions could require changes to your makefile. For information about these differences, see Section 7.2 and Appendix B.
- By default, the Tru64 UNIX compiler links your application with shared libraries. If you want to link your application with static libraries, specify the `-non_shared` option on the `cc` or `ld` command lines in the makefile.
- The `make` command on Tru64 UNIX systems does not support retrieving source files automatically from a Source Code Control System (SCCS) archive.

For information about using `make` on a Tru64 UNIX system, see `make`(1).

The ULTRIX `make` command is also in the `/usr/opt/ultrix/usr/bin` directory. To use the ULTRIX `make` command, edit the `.login` file and add the following line to the end of the file:

```
source /etc/ultrix_login
```

This entry modifies your `PATH` variable to allow access to the ULTRIX `make` command. For information about using the ULTRIX `make` command on a Tru64 UNIX system, see `make`(1u).

7.2 Migrating References to Header Files

The set of header files on a Tru64 UNIX system is slightly different from the set of header files on an ULTRIX system.

The contents of some Tru64 UNIX header files differ from the contents of the equivalent ULTRIX header files. These differences can appear in a number of ways. For example, the interface to a service might be slightly different, structure definitions might be located in different header files, values might have changed to reflect the 64-bit Alpha architecture, or nearly identical structures or constants might have different names. For a list of differences in `/usr/include` header files, see Appendix B.

Some of the ULTRIX header files are unavailable. These header files are primarily:

- Header files corresponding to features that are unsupported in the Tru64 UNIX system; for example, `/usr/include/hesiod.h`, which is not present because the Tru64 UNIX system does not support the Hesiod service
- Header files used by specific ULTRIX system facilities but which are not needed by Tru64 UNIX utilities

For a list of the unavailable `/usr/include` header files, see Table B–2.

The Tru64 UNIX header files are kept in a directory hierarchy descending from the /usr/include directory. Table 7–1 lists most of the directories containing standard header files.

Table 7–1: Locations of Standard Tru64 UNIX Header Files

Directory	Description
/usr/include	General C header files
DPS	Display PostScript System C header files
DXm	Compaq extensions to Motif C header files
Mrm	Motif resource manager C header files
X11	X Toolkit header files
Xm	Motif C header files
dec	Compaq specific interface header files
lvm	C header files for Logical Volume Manager (LVM)
mach	Mach-specific C include files
net	Miscellaneous network C header files
netimp	C header files for IMP protocols
netinet	C header files for Internet standard protocols
netns	C header files for XNS standard protocols
nfs	C header files for Network File System (NFS)
protocols	C header files for Berkeley service protocols
rpc	C header files for remote procedure calls (RPCs)
servers	C header files for servers
sys	System C header files (kernel data structures)
tli	C header files for Transport Layer Interface (TLI)
ufs	C header files for UNIX File System (UFS)

The compiler can help you migrate your application by finding inconsistencies in the application's use of a symbol, function, or declarations in a header file. The Tru64 UNIX C compiler issues error messages for the following conditions:

- Header file not found:

```
cfe: Error: file.c: 1: Cannot open file cursesX.h for #include
```

- Undefined symbol (a symbol that is not defined before its use)

  This message helps you to find references to header-file symbols that have moved or are no longer available:

```
cfe: Error: file.c, line 8: 'ENOSYSTEM' undefined,
reoccurrences will not not be reported
```

- Multiply defined symbol (a local definition that conflicts with a header-file definition):

```
cfe: Warning: file.c:4: Tried to redefine the macro EDEADLK,
this macro keeps the old definition in std/std1 mode,
otherwise the macro is redefined.
```

- Redeclared function (a local function declaration that conflicts with a header-file declaration):

```
cfe: Error: t.c, line 7: redeclaration of 'openlog'; previous
declaration at line 120 in file '/usr/include/syslog.h'
int openlog(char*, int);
----^
```

- Mismatched function use and prototype (failure of a function usage to supply the number of arguments declared by the prototype declaration):

```
cfe: Error: file.c, line 12: Number of arguments doesn't
agree with number in declaration
```

- Incompatible function arguments (an attempt to supply incompatible arguments to a function) :

```
cfe: Warning: file.c, line 12: Incompatible pointer type
assignment
```

Because function declarations or prototypes are not required by the C language before a function call, the compiler cannot detect misuse of functions that did not have a preceding prototype declared. You might need to find differences in these cases by first determining which header files your application depends on, generating a list of the function declarations these header files contain, and then using this list of functions to generate a cross-reference for the needed header files on a Tru64 UNIX system. Then you can cross-check the actual declarations for changes in the function interfaces and modify your source code where necessary. Doing this may require that you build short shell scripts to help search for the appropriate definitions in the list of header files. The compiler has features that might be of some use in these tasks:

- To produce a complete list of pathnames for include files a program depends on, use the following command on the ULTRIX system:

```
% cc -M file_name.c
```

  Be sure to use the same define (–D), undefine (–U), and include (-I) command directives that you would typically use to compile this program.

- To generate a list of functions that your application needs, and to compile without allowing any library definitions on the command line when building your ULTRIX application, use the following command:

  ```
  % cc file_name.c -L
  ```

  Do not include any additional system `-ldirectory` options. The `–L` option inhibits `ld` from searching the standard directories for libraries. The `ld` command will issue messages identifying any unresolved symbols. The following short scripts can help you locate the files containing objects that the compiler fails to resolve:

  – Use the following command line to locate references to a particular string (`what to find` in the following example) in all files contained in the working directory or in subdirectories of the working directory:

  ```
  % find . -type f -print | xargs grep "what to find" > logfile
  ```

  Typically, this command is used from the `/usr/include` directory to locate information in header files.

  – The following script searches every library archive (`*.a`) on the system for the object named as its first command-line argument:

  ```
  #! /bin/sh
  for i in /lib/*.a /usr/lib/*.a; do
    ar t $i | grep $1 && echo "$1 found in $i"
  done
  ```

  Use this script (called `arfind` in this example) as follows:

  ```
  % arfind object-to-find
  ```

See Section 7.6.1 for more information on libraries.

7.3 Migrating to a 64-Bit Environment

The 64-bit Tru64 UNIX system is different from the 32-bit ULTRIX system in the size of addresses, the availability of 64-bit integer types, the data type alignment restrictions, byte and word accessibility, and interoperability between 32-bit and 64-bit systems. These differences affect the following areas in your programs:

- Pointers
- Constants
- Structures
- Variables
- Library calls

The following sections discuss each of these areas and the changes you must make to your program to take full advantage of the 64-bit environment, and to permit interoperability with 32-bit systems.

7.3.1 Pointers

This section describes migration problems that some applications will encounter because they make assignments based on the assumption that pointers are the same length as `int` variables. This section also contains information on how to overcome problems with pointer-to-`int` assignments with little or no recoding. (Information about other types of pointer assignments that may require recoding is provided in Section 7.3.4.2.)

The following table shows the lengths of the data types that are used to hold addresses and that can, in some usage situations, cause problems when migrating an application to a Tru64 UNIX system:

	ULTRIX	Tru64 UNIX
Pointer	32 bits	64 bits
int	32 bits	32 bits
long	32 bits	64 bits

Many C programs, especially older C programs that do not conform to currently accepted programming practices, assign pointers to `int` variables. Such assignments are not recommended, but they do produce correct results on systems in which pointers and `int` variables are the same size. However, on a Tru64 UNIX system, this practice can produce incorrect results because the high-order 32 bits of a Tru64 UNIX address are lost when a 64-bit pointer is assigned to a 32-bit `int` variable. The following code fragment shows this problem using Tru64 UNIX:

```
{
char *x;   /* 64-bit pointer */
int z;     /* 32-bit int variable */
   .
   .
   .
  x = malloc(1024); /* get memory and store address in 64 bits */
  z = x;    /* assign low-order 32 bits of 64-bit pointer to
               32-bit int variable */
}
```

Similar problems with the length of pointers occur in applications that consist of a mix of C and FORTRAN programs in which a pointer in a C program is declared as an `INTEGER*4` variable in a FORTRAN program, leaving the conversion implicit and causing the loss of the 32 high-order bits in the pointer.

7.3.1.1 Controlling Pointer Size and Allocation

The Tru64 UNIX system has a set of compiler options and pragmas you can use to control pointer size and allocation, thereby allowing ULTRIX applications that may make assumptions about pointers being 32 bits to more easily migrate to a Tru64 UNIX environment.

The set of options for the cc command is known as the xtaso option. Combined with the –taso linker option (which is required when the xtaso option is used), the xtaso option can prevent problems with invalid addressing and pointer truncation that could occur when migrating applications with 32-bit pointers to the Tru64 UNIX system. There are limits to the use of the xtaso option. First, the option should only be used in end-user application programs, and not in library programs. Second, the end-user application should be known to have 32-bit dependencies.

This option is most useful for applications that have already been migrated to the Tru64 UNIX system, but exhibit performance problems due to either memory limitations or the heavy use of dynamic memory allocation.

The elements of the xtaso option are:

- **#pragma pointer_size *specifier***

  A C language pragma for controlling pointer size allocation that is recognized by the compiler, but only acted on when the –xtaso option is specified. This pragma is defined in the *Programmer's Guide* manual.

- The –xtaso and –xtaso_short options to the cc command

  The –xtaso option causes the compiler to respond to the pragmas that control pointer size allocation. The –xtaso_short option forces the compiler to allocate 32-bit pointers by default.

  For more information about these compiler options, see cc(1).

- –taso linker option

  The linker option that enables correct –xtaso support. The –xtaso option allows you to create pointers that are only 32 bits. Because 32-bit pointers cannot represent the entire range of addresses that are possible in the Tru64 UNIX system environment, you must make sure that any programs you compile are linked using the –taso option.

  For more information, see the following sections.

7.3.1.2 Correcting the Pointer-to-int Assignment Problem

The most portable way to fix the problem presented by pointer-to-int assignments in existing source code is to modify the code to eliminate this type of assignment. However, in the case of large applications, this can be time consuming. (To find pointer-to-int assignments in existing source code, use the lint –Q command.)

Another way to overcome this problem is to use the Truncated Address Support Option (–taso option). The –taso option makes it unnecessary for the pointer-to-int assignments to be modified. It does this by causing a program's address space to be arranged so that all locations within the program when it starts execution can be expressed within the 31 low-order bits of a 64-bit address, including the addresses of routines and data coming from shared libraries.

The –taso option does not affect the sizes used for any of the data types supported by a Tru64 UNIX system. Its only effect on any of the data types is to limit addresses in pointers to 31 bits (that is, the size of pointers remains at 64 bits but addresses use only the low-order 31 bits).

The 31-bit address limit is used to avoid the possibility of setting the sign bit (bit 31) in 32-bit int variables when pointer-to-int assignments are made. Allowing the sign bit to be set in an int variable by a pointer-to-int assignment would create a potential problem with sign extension. For example:

```
{
char *x;   /* 64-bit pointer */
int z;     /* 32-bit int variable */
.
.
.
           /* address of named_obj = 0x0000 0000 8000 0000 */
    x = &named_obj;  /* 0x0000 0000 8000 0000 = original pointer
                        value */
    z = x;  /* 0x8000 0000 = value created by pointer-to-int
               assignment */
    x = z;  /* 0xffff ffff 8000 0000 = value created by pointer-
               to-int-to-pointer or pointer-to-int-to-long
               assignment (32 high-order bits set to ones by
               sign extension) */
}
```

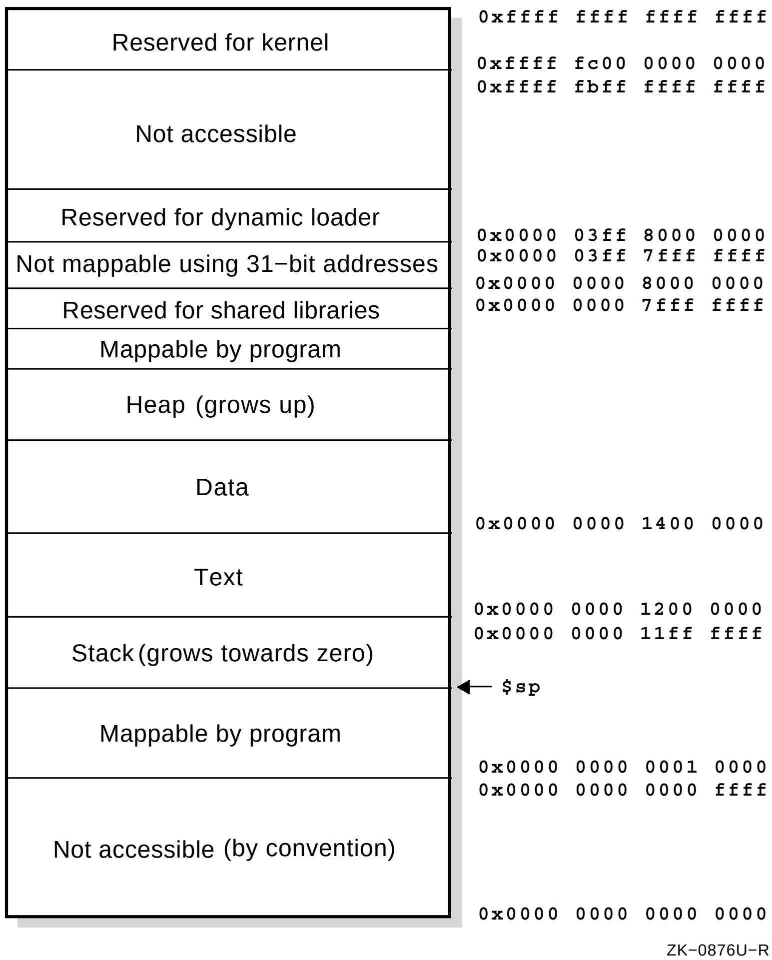
7.3.1.3 Use and Effects of the –taso Option

You can specify the –taso option on the cc or ld command lines used to create an application's object modules. (If you specifiy it on the cc command line, the option is passed to the ld linker.) The –taso option directs the linker to set a flag in object modules and this flag directs the loader to load the modules into 31-bit address space.

The –taso option ensures that *text* and *data* segments of an application are loaded into memory that can be reached by a 31-bit address. Thus, whenever a pointer is assigned to an int variable, the values of the 64-bit pointer and the 32-bit int variable will always be identical (except in the special situations described in Section 7.3.1.4).

Figure 7–1 is an example of a memory diagram of programs that use the –taso and –call_shared options (and do not use threads). (If you invoke the linker (ld) through the cc command, the default is –call_shared. If you invoke ld directly, the default is –non_shared.)

Figure 7–1: Layout of Memory Under the -taso Option



Note that stack and heap addresses will also fit into 31 bits. The stack grows downward from the bottom of the text segment, and the heap grows upward from the top of the data segment.

The −T and −D options (linker options that are used to set text and data segment addresses, respectively) can also be used to ensure that the text and data segments of an application are loaded into low memory. The −taso option, however, in addition to setting default addresses for text and data segments, also causes shared libraries linked outside the 31-bit address space to be appropriately relocated by the loader.

The default addresses used for the text and data segments are determined by the options that you specify on the cc command line:

- Specifying the –non_shared or –call_shared option with the –taso option results in the following defaults:

  0x0000 0000 1200 0000 (text segment's starting address)
  0x0000 0000 1400 0000 (data segment's starting address)

- Specifying the –shared option with the –taso option results in the following defaults:

  0x0000 0000 7000 0000 (text segment's starting address)
  0x0000 0000 8000 0000 (data segment's ending address)

Using these default values produces sufficient amounts of space for text and data segments for most applications (see the *Assembly Language Programmer's Guide* for details on the contents of text and data segments). The default values also allow an application to allocate a large amount of mmap space.

If you specify the –taso option and also specify text and data segment address values with –T and –D, the values specified override the –taso default addresses.

You can use the odump utility to check whether a program was built successfully within a 31-bit address space. To display the start addresses of the text, data, and bss segments, enter the following command:

```
% odump -ov obj_file_x.o
```

None of the addresses should have any bits set in bits 31 to 63; only bits 0 to 30 should ever be set.

Shared objects built with the –taso option cannot be linked with shared objects that were not built with the –taso option. If you attempt to link shared objects that way, the following error message is displayed:

```
Cannot mix 32 and 64 bit shared objects without the -taso
option.
```

7.3.1.4 Limits on the Effects of the –taso Option

The –taso option does not prevent a program from mapping addresses outside the 31-bit limit, and it does not issue warning messages if this is done. Such addresses could be established using any one of the following mechanisms:

- –T and –D options

  As previously noted, if the –T and –D options are used with the –taso option, the values that you specify for them will override the –taso option's default values. Therefore, to avoid defeating the purpose of the

–taso option, you must select addresses for the –T and –D options that are within the address range observed by the –taso option.

- malloc() function

  To avoid problems with addressing when you use malloc in a taso application that does not use threads, you must ensure that the combination of the default data-size resource limit and the starting address of the data segment do not exceed the maximum 31-bit address (0x7fff ffff). Applications that use threads are unlikely to encounter this problem because memory allocations for thread applications start in a much lower address space than that used for nonthread applications.

  The data-size resource limit is the maximum amount of data space that can be used by a process. This limit can be adjusted using the limit (C shell) or ulimit (Korn shell) commands. As previously noted, you can adjust the starting address of the data segment by using the –D option on the cc command.

- mmap system call

  Applications that use the mmap system call must use a jacket routine to mmap to ensure that mapping addresses do not exceed a 31-bit range. This entails taking the following steps:

  1. To prevent mmap from allocating space outside 31-bit address space, specify the following compilation option on the cc command line for all modules (or at least all modules that refer to mmap):

     ```
     -Dmmap=_mmap_32_
     ```

     This option equates the name of the mmap function with the name of a jacket routine (_mmap_32_). As a result, the jacket routine is invoked whenever references are made to the mmap function in the source program.

  2. If the mmap function is invoked in only one of the source modules, either include the jacket routine in that module or create an mmap_32c.o object module and specify it on the cc command line. (The file specification for the jacket routine is /usr/opt/alt/usr/lib/support/mmap_32.c.)

     If the mmap function is invoked from more than one source file, you must use the method of creating an mmap_32c.o object module and specifying it on a cc command line because including the jacket routine in more than one module would generate linker errors.

7.3.2 Constants

Check the use of constants in your program, particularly if you are going to exchange data between 32-bit and 64-bit systems. Some constants might have different values between 32-bit and 64-bit systems, which might change the behavior of some operators. For example, hexadecimal constants are more likely to become long on Tru64 UNIX systems. The following table lists some constants and their values:

C Constant	Value	Value (32-Bit)	Value (64-Bit)
0xFFFFFFFF	$2^{32}-1$	–1	4,294,967,295
4294967296	2^{32}	0	4,294,967,296
0x100000000	2^{32}	0	4,294,967,296
0xFFFFFFFFFFFFFFFF	$2^{64}-1$	–1	–1

7.3.2.1 Integer and Long Constants—Assignment and Argument Passing

In C, an integer constant is specified as 543210. To specify a `long int` constant, use the letter suffix L or l. To specify an unsigned `long`, you use the UL or ul suffix. (L is preferred since lowercase l is easily confused with the number one.) Note the example where three different constants are passed to the function, `labs()`:

```
labs(543210)
labs(543210L)
labs(543210UL)
```

On an Alpha system, 543210 is treated as a 4-byte constant, and 543210L (or 543210UL) is treated as an 8-byte constant. If the `labs()` function expects a `long` argument, each of these invocations would work as expected since the `int` constants would be converted to `long`. If the `labs()` function expects type `int`, the `long` constant would be truncated to an integer constant. This truncation would result in the loss of significant digits if the constant was greater than the maximum integer constant (INT_MAX) of +2147483647, or less than the minimum integer constant (INT_MIN) of –2147483648, or for unsigned constants greater than the maximum unsigned integer constant (UINT_MAX) of 4294967295. This problem would also be present in an assignment expression where a `long` integer constant was assigned to a variable of type `int`. In these cases, explicitly use the L or UL suffix and make sure the function arguments or variables being assigned to are of the appropriate `long` type.

When you need to pass zero to a pointer argument and no function prototype is visible, always use NULL (defined in the `stdio.h` file). Using zero will result in using a 4-byte zero instead of a 8-byte zero (0L). In a comparison, an assignment, or a function call where the correct function prototype is in scope, standard C promotion rules will be in effect and the correct value will be assigned.

To minimize assignment and argument errors in your code, use function prototypes because the number and type arguments are checked.

7.3.2.2 Integer and Long Constants—Shift Operations

A bit shift operation on an integer constant will yield a 32-bit constant. If you need a result of type `long`, then you need to use the L or UL suffix for `long` integer constants. The following example results in `value` being assigned a 32-bit constant:

```
long value;

value = 10 << 2;
```

The top 32 bits of `value` will depend on the type of the value shifted. Signed values are sign extended; unsigned values are zero extended. If you want a 64-bit constant, be sure to use the L or the UL suffix. (Note that only the left operand of a shift operator determines the result type. The type of shift count operand is irrelevant.)

7.3.3 Structures

The 64-bit data size of the `long` and `pointer` types affects the size, member alignment, alignment, and bit fields of structures.

7.3.3.1 Size

The size of structures and unions on Tru64 UNIX systems can be different from those on 32-bit systems. For example, the following structure, `TextNode`, doubles in size on a 64-bit system because the `pointer` types are double in size (from 4 bytes to 8 bytes):

```
struct TextNode{
 char                    *text;
 struct TextNode         *left;
 struct TextNode         *right;
 };
```

If you are sharing data defined in structures between 32-bit and 64-bit systems, be careful about using the `long` and `pointer` data types as members in shared structures. These data types introduce sizes that are not available on 32-bit systems.

To increase your application's portability, do the following in your application:

- Use `typedef` types in structures and set up the types as appropriate for the system. You can automatically do this by using information in the `limits.h` header file.
- Be careful when building unions between the `int` and `pointer` data types, because they are not the same size.

7.3.3.2 Member Alignment

Members of structures and unions are said to be aligned on their natural boundaries. That is, `char` is aligned on a byte boundary, `short` on a word boundary, `int` on a longword boundary, and `long` and `pointer` on quadword boundaries.

This means that additional space can be used for padding member alignment in structures and unions. For example, on 32-bit systems the size of the following structure is 16 bytes. On 64-bit systems, the size of the structure is 32 bytes: 8 bytes for each pointer and 4 bytes of padding after the member, `size`, for the alignment of the pointer, `left`.

```
struct  TextCountNode {
         char                          *text;
         int size,
         struct TextCountNode          *left;
         struct TextCountNode          *right;
};
```

7.3.3.3 Alignment

In the 64-bit environment, structures are aligned according to the strictest aligned member. This aids in aligning other structure members on their required boundaries. Structures are also padded to ensure proper alignment. Padding can be added within the structure or at the end of the structure, to terminate the structure on the same alignment boundary on which it started. Therefore, observe the following alignment guidelines when working with structures in a 64-bit environment:

- Always use the `sizeof` operator to determine the size of a structure. Do not assume the size of a structure is the accumulated size of all of the objects defined in it. Additional space might be needed for padding the member alignment.
- Minimize the amount of padding needed in a structure by reordering the members.

In the following example, the size of `CountedString` is 16 bytes (`*text` = 8 bytes, `count` = 4 bytes, tail padding = 4 bytes). This structure is aligned on a quadword boundary because the pointer requires quadword alignment. No additional padding (beyond 4 bytes of tail padding) is necessary because `CountedString` will naturally align on a quadword boundary.

```
struct  {
          char *text;
          int count;
          }CountedString;
```

In the following example, the inclusion of CountedString as a member in the definition forces the alignment of the beginning of the structure to be on a quadword boundary. Additional padding might be introduced (depending upon the value of MAX_LINE) to ensure proper quadword alignment for the structure member, string. Additional padding might also be introduced at the end of the structure, to ensure proper structure alignment for arrays of these structures.

```
CountedString CsArray[10]
struct {
          char line[MAX_LINE];
          struct CountedString string;
}TextAndString;
```

In the following example, the structure has a size of 24 bytes:

```
struct s{
           int count;
           struct s        *next;
           int total;
}
```

If this structure is reordered, the structure now has a size of 16 bytes.:

```
struct s{
          struct s         *next;
          int count;
          int total;
}
```

7.3.3.4 Bit Fields

Bit fields are allowed on any integral type on Alpha systems. (ANSI C only requires bit fields with int, signed int, and unsigned int types.) In a C declaration, if one bit field immediately follows another in a structure declaration, the second bit field will be packed into adjacent bits of the former unit. Since long is 64 bits in length on Alpha systems, adjacent declarations of bit fields of type long might contain multiple bit field definitions in cases that previously did not on RISC or VAX systems. This change might cause different results in operations on these bit fields.

To ensure the same behavior in operations on bit fields, change bit field definitions of type long to int.

7.3.4 Variables

The 64-bit Tru64 UNIX environment also changes assumptions about how you declare your variables, and how you use them in assignments and in function arguments.

7.3.4.1 Declarations

To enable your application to work on both 32-bit and 64-bit systems, check your `int` and `long` declarations. If you have specific variables that need to be 32 bits in size on both ULTRIX MIPS and Alpha systems, define the type to be `int`. If the variable should be 32 bits on ULTRIX MIPS systems and 64 bits on Alpha systems, define the variable to be `long`. Remember, if the type specifier is missing from a declaration, it defaults to `int` type. For example, here are six declarations that declare the variables to be of size `int` and the function to be returning type `int`:

```
extern   e;
register n;
static   x;
unsigned i;
const    c;
function ();
```

7.3.4.2 Assignments and Function Arguments

Since pointer, `int`, and `long` are no longer the same size in the 64-bit Tru64 UNIX environment, problems may arise depending on how the variables are assigned and used in your application. Use the following guidelines:

- Do not use `int` and `long` interchangeably because of the possible truncation of significant digits. For example, avoid assignments similar to the following:

  ```
  int i;
  long l;
  i = l;
  ```

  Use the `lint -Q` command to help you find these potential problems. See Section 7.5 and `lint`(1) for more information on the `lint` command.

- Do not pass arguments of type `long` to functions expecting type `int`. For example, avoid assignments similar to the following:

  ```
  int  toascii(int);
  int i;
  long l;
  i= toascii(l)
  ```

- Do not freely exchange pointers and integers. Assigning a pointer to an `int`, assigning back to a pointer, and dereferencing the pointer may result in a segmentation fault. For example, avoid assignments similar to the following:

  ```
  int i ;
  char *buffer;

  buffer = (char *)malloc(MAX_LINE)
  ```

```
i = (int)buffer;
buffer = (char*)i;
```

Use the `lint –Q` command to find these pointer-to-`int` assignments.

If special steps are taken, pointer-to-`int` assignments can be retained without causing addressing problems. See Section 7.3.1 for information on how this is done.

- Do not pass a pointer to a function expecting an `int` as this will result in lost information. For example, avoid assignments similar to the following:

  ```
  void f();
  char *cp;
  f(cp);
  ```

  This nonportable function declaration will produce a compiler warning if you use ANSI C prototypes such as the following:

  ```
  void f(int);
  char *cp;

  f(cp);
  ```

  Use the `lint -Q` command to find these pointer-to-`int` assignments.

- Use `void *type` if you need to use a generic pointer type. This is preferable to converting a pointer to a type `long`.

- Beware of the use of aliasing (different multiple definitions of the same object). For example, the following two structures refer to the same object in different ways:

  ```
  struct node {
     int src_addr,dst_addr;
     char *name;
   } ;

  struct node {
     struct node *src, *dst;
     char *name;
   }
  ```

  Replace this coding with a union declaration. Be thorough when migrating this type of code to a 64-bit system, because the interdependencies and incompatibilities between these two structures might be difficult to find.

- Examine all assignments of a `long` to a `double` as this can result in a loss in accuracy.

  On a 32-bit system, code can assume that a `double` contains an exact representation of any value stored in a `long` (or a pointer). By default, a `long` is 32 bits and a `double` is 64 bits with 48 bits of mantissa.

On a 64-bit Tru64 UNIX system, this is no longer a valid assumption. For example, the following code executes differently on a 32-bit and 64-bit machine:

```
#include <stdio.h>

        long l = 7777777777777777777L;
        long l2;
        double d;

        main()
        {
                d = l;
                l2 = d;
                if(l == l2)
                        printf("On a 32-bit machine\n");
                else
                        printf("On a 64-bit machine\n");

                return(0);
        }
```

7.3.4.3 The sizeof Operator

The result of the `sizeof` operator is of type `size_t`, which is an `unsigned long` on Alpha systems.

7.3.4.4 Pointer Subtraction

The length of the integer required to hold the difference between two pointers to members of the same array, `ptrdiff_t` (`stddef.h`), is a `signed long` on Alpha systems.

7.3.4.5 Functions with a Variable Number of Arguments

When writing a routine that receives a variable (context-dependent) number of arguments, you must use the `stdargs` (`stdarg.h`) or `varargs` (`varargs.h`) mechanism. See `varargs`(3) for more information on the use of these macros.

Programs written using `varargs` that expect the `va_list` type to be a pointer, or that make assumptions about the stack layout of a routine's arguments, are nonportable. Such programs must be modified to correctly use the `varargs`(3) mechanism. Failure to do so will give compile-time errors, or incorrect run-time results.

See `varargs`(3) for more information.

7.3.5 Library Calls

The 64-bit data types also affect the following library calls:

- printf and scanf functions
- malloc and calloc functions
- lseek system call
- fsetpos and fgetpos functions

The following sections describe how these functions are affected.

7.3.5.1 The printf and scanf Functions

When using the printf function for long types, you use the length modifier l (lowercase letter L) with the d, u, o, and x conversion characters to specify assignment of type long or unsigned long. For example, when printing a long as a signed decimal, use %ld instead of %d. When printing a long as a unsigned decimal, use %lu instead of %u. If the length modifier is not used, the type is assumed to be int, or unsigned int, depending upon the conversion character. In this case, the long types will be converted to the smaller int types and information might be lost.

When printing a pointer, use %p. If you want to print the pointer as a specific representation, the pointer should be cast to an appropriate integer type long before using the desired format specifier. For example, to print a pointer as a long unsigned decimal number, use %lu:

```
char *p;

printf ( "%p %lu\n",  (void *)p,  (long)p );
```

As a rule, to print an integer of arbitrary size, cast the integer to long or unsigned long, and use the %ld (unsigned long) conversion character. For example:

```
printf ("%ld\n", (long) num ));
```

7.3.5.2 The malloc and calloc Functions

Memory allocation library functions such as malloc() guarantee to return data aligned to the maximum alignment of any object. In the 64-bit Tru64 UNIX environment, malloc() returns a pointer to memory that is at least quadword aligned.

7.3.5.3 The lseek System Call

When calling the lseek() system call to set the current position in a file, use the off_t type defined in types.h for the file offset. Passing an int or long constant might work, but it is not portable and is not guaranteed to continue to work. The following example shows correct uses of lseek():

```
lseek function:

#include <unistd.h>

off_t offset, pos;
pos = lseek( fd, offset, SEEK_SET );
pos = lseek( fd, (off_t)0, SEEK_CUR);
```

7.3.5.4 The fsetpos and fgetpos Functions

When setting or getting the file postions for a file with the ANSI C functions of fsetpos() or fgetpos(), respectively, the file position is specified by a value of type fpos_t. This type is defined as a long in the 64-bit Tru64 UNIX environment.

7.4 Correcting C Syntax Errors

The C compiler on the Tru64 UNIX system is different from the C compilers on the ULTRIX system. Because of differences in the compilers, you might encounter C syntax errors on Tru64 UNIX systems that you did not encounter on ULTRIX systems. This section contains information to help you find and correct these errors. In particular, it includes a list of the Tru64 UNIX predefined symbol names and their meanings. This section also provides information about using Tru64 UNIX compiler options to get maximum compatibility with ULTRIX compilers, and information about differences between Tru64 UNIX and ULTRIX C syntax for each of the ULTRIX compilers.

7.4.1 Differences Between Tru64 UNIX and ULTRIX Predefined Symbols

Both the Tru64 UNIX and ULTRIX systems supply predefined symbols for the cc command. You use these symbols to write conditional code for different hardware platforms, different operating systems, and different programming environments. On Tru64 UNIX systems, the __STDC__ symbol is defined as follows:

- When you use the −std0 option, the __STDC__ symbol is undefined. (The −std0 option compiles code as defined by Kernighan and Ritchie (K&R) C.)
- When you use the −std option, the __STDC__ symbol is 0. (The std option compiles code as specified by the ANSI C standard. This option also allows some extensions to the ANSI C standard.)

- When you use the –std1 option, the __STDC__ symbol is 1. (The –std1 option compiles code strictly according to the ANSI C standard.)

The predefined symbols on Tru64 UNIX systems have different names from their equivalents on ULTRIX systems. Table 7–2 compares the Tru64 UNIX and ULTRIX predefined symbols. If you use these symbols in your application, you must modify the symbol name in your source file before you recompile your application.

Table 7–2: Comparison of Tru64 UNIX and ULTRIX Predefined Symbols for the cc Command

Name for –std and –std1 Modes	Name for –std0 Mode	Name for ULTRIX on RISC Systems	Name for ULTRIX on VAX Systems
String containing the host hardware name:			
__alpha	__alpha	__host_mips__	vax
String containing the target hardware name:			
__alpha	__alpha	mips	vax
String containing the operating system name:			
__osf__	__osf__	unix	unix
__unix__	__unix__	ultrix	ultrix
	unix	bsd4_2	bsd4_2
String indicating that the host is a BSD system:			
_SYSTYPE_BSD	_SYSTYPE_BSD	SYSTYPE_BSD	Not applicable
	SYSTYPE_BSD		
String indicating that the application is written in C:			
__LANGUAGE_C__	__LANGUAGE_C__	LANGUAGE_C	Not applicable
	LANGUAGE_C		
String indicating that double floating-point format is used:			
Not applicable	Not applicable	Not applicable	GFLOAT

7.4.2 Differences Between Tru64 UNIX C and ULTRIX C on RISC Systems

Note

This section describes the behavior of ULTRIX C on Versions 4.3 and earlier RISC ULTRIX systems, and not the behavior of ULTRIX C on Versions 4.3A or later systems. The reason is that Versions 4.3A and later systems employ the MIPS Version 3.0 compiler environment, which is more completely similar to the Tru64 UNIX C compiler environment than the MIPS Version 2.10 compiler environment on earlier ULTRIX RISC systems, which is described here.

When you compile your ULTRIX application on a Tru64 UNIX system, you may notice some differences in how the compilers operate. For example, the Tru64 UNIX compiler might issue errors or warnings in cases for which the

ULTRIX compiler does not. To minimize the effect of these differences, use the Tru64 UNIX compiler option that provides the most compatibility, as shown in Table 7–3.

Table 7–3: Compilation Options that Are Compatible with ULTRIX C on RISC Systems

If You Use This ULTRIX Option	Use This Tru64 UNIX Option
Default — K&R C with ANSI extensions.	Default (`–std0`)—K&R C with ANSI extensions. Some ANSI extensions are implemented differently.
`–std`	`–std` (Issues a warning message for certain non-ANSI syntax. This mode is stricter on a Tru64 UNIX system, so you receive more warnings than you do on an ULTRIX system.)

Although the Tru64 UNIX compiler options offer compatibility with the ULTRIX C for RISC compiler, some differences between the two compilers exist. The ULTRIX and Tru64 UNIX compilers operate differently in some respects regardless of which Tru64 UNIX compiler mode you use. Other differences occur only when you use the `–std0` or the `–std1` option. The rest of this section describes these differences.

7.4.2.1 Differences that Apply to All Modes

The following list describes compilation differences between ULTRIX C on RISC systems and Tru64 UNIX C. You might notice the following differences regardless of the compilation mode you use:

- The Tru64 UNIX C compiler issues a warning if constants are longer than the maximum allowed by `ULONG_MAX`. A similar warning occurs if octal and hexadecimal character escape sequences exceed the value of `UCHAR_MAX`. The ULTRIX C compiler issues no warnings in these situations.
- If a signed multicharacter constant is converted to an integer, the value of the integer might differ between Tru64 UNIX systems and ULTRIX systems. This situation is true if the constant contains a negative value.
- As required by the ANSI C standard, the Tru64 UNIX C compiler strips a backslash (\) followed by a carriage return (`^M`) during the preprocessing stage. On ULTRIX systems, these characters are stripped during the later translation phase. Programs containing such constructs might not work properly when input to the Tru64 UNIX C compiler.
- The Tru64 UNIX C compiler does not allow you to modify a type you create with the `typedef` statement. For example, the following statement is invalid on Tru64 UNIX systems:

```
typedef int account;
   .
   .
   .
account monthly;
unsigned account display_account;
```

To achieve this effect on Tru64 UNIX systems, you must create both a signed and unsigned type, as shown:

```
typedef int account;
typedef unsigned int display_variable;
   .
   .
   .
account monthly;
display_variable display_account;
```

- On Tru64 UNIX systems, you cannot declare or define a type within a function prototype. The ULTRIX compiler allows this, although doing so causes the parameter to be incompatible with any other type.

  For example, suppose the structure S shown in the following declaration has not been declared previously. Any further type matching of the parameter list results in an error. At the end of the prototype, the scope ends, which means that S is no longer available:

  ```
  int convert_array (struct S *p);
  ```

  On Tru64 UNIX systems, you must declare the structure S outside of the function prototype, as shown:

  ```
  struct S *p;

  int convert_array(struct S);
  ```

- If you include a directory specification as an option to the #line directive, the Tru64 UNIX C preprocessor uses the directory as the local directory for all subsequent #include directives. The ULTRIX C preprocessor did not process the #line directives in this manner. To force the compiler to search locally instead of using the #line directive information, use the –I option to the cc command and specify the local directory (the period character), as follows:

  ```
  cc –g –O0 –I. –c sample_module.c
  ```

  Since various C code generators (for example, lex and yacc) insert a #line directive into the generated C code, you might encounter this error inadvertently.

- The Tru64 UNIX C compiler does not allow you to specify #if directives within a macro call. Move #if directives outside of macro calls.

- The Tru64 UNIX C compiler requires you to use braces ({ }) in initializers more precisely than the ULTRIX C compiler.

For example, the following initialization is valid on ULTRIX systems:

```
struct S {char i[10]; int j} y = {{"aeiou", 1}};
```

The Tru64 UNIX C compiler issues an error message in response to the preceding initialization. To achieve the same effect on Tru64 UNIX systems, use the following initialization statement:

```
struct S {char i[10]; int j} y = {"aeiou", 1};
```

In this example, the initialization of `y` contains only one set of braces.

- On Tru64 UNIX systems, you cannot declare a single type name (using `typedef`) more than once except within an inner block.
- The Tru64 UNIX C compiler allows you to specify hexadecimal escape sequences in character strings and constants. On ULTRIX systems, the escape sequence is translated; for example, \x is interpreted as x on ULTRIX systems.

7.4.2.2 Differences that Apply to the Default Mode

The default Tru64 UNIX C compilation mode (specified by the `−std0` option) differs from ULTRIX C in the following ways that can affect migrating C source code from ULTRIX C:

- To comply with the ANSI C standard, the Tru64 UNIX C compiler replaces comments with one space character during preprocessing. Therefore, you cannot use a comment as a concatenation character on Tru64 UNIX systems.

  On ULTRIX systems, comments within C statements are deleted with no spaces. This action allows you to use a comment as a concatenation character.

  On Tru64 UNIX systems, replace a comment that you use as a concatenation character with the ANSI-defined concatenation characters (##).

- The Tru64 UNIX compiler uses the ANSI definition of a string for C programs. ANSI defines a string in the C language as a contiguous sequence of characters terminated by, and including, the first null character. As a result, a partial string is not a valid processing token, so you cannot use a partial string in the replacement list of a macro definition.

  The ULTRIX compiler allows you to use a partial string in a macro definition, as shown:

  ```
  #define abc "123
  ```

  You can use this definition in a `printf` statement, as follows:

  ```
  printf(abc 456");
  ```

The output from this `printf` statement is the following:

```
123 456
```

To get the same effect on a Tru64 UNIX system, use the following definition and `printf` statement:

```
#define abc "123"

printf(abc " 456");
```

- On Tru64 UNIX systems, you can use recursive macro definitions when you specify the `-std0` option. On ULTRIX systems, you cannot define macros recursively.

7.4.2.3 Differences that Apply to Strict ANSI Mode

The strict ANSI C Tru64 UNIX compilation mode (specified by the `-std1` option) differs from ULTRIX C in the following ways that can affect migrating C source code from ULTRIX C:

- On Tru64 UNIX systems, declaring a local and external variable of the same name causes an error. You must use unique identifier names for each scope.

  On ULTRIX systems, you can declare a local variable of the same name as an external variable. The local variable has precedence.

- On Tru64 UNIX systems, you must not use a trailing comma in an enumerator list. ULTRIX systems allow the trailing comma as shown:

  ```
  enum protocols { TCP, SNMP, OSI, };
  ```

  The trailing comma causes an error on Tru64 UNIX systems, so you must remove it.

- On Tru64 UNIX systems, you cannot specify an empty declaration such as the following one:

  ```
  main
  {
  ;
  ⋮
  }
  ```

  Remove all empty declarations from your program.

- You cannot cast the left-hand side of an assignment statement on Tru64 UNIX systems. You must remove any such casts.

  On ULTRIX systems, you can cast the left-hand side of an assignment statement, so long as the result of the left-hand side is the same size as the result of the right-hand side.

- On Tru64 UNIX systems, each identifier declaration must contain either a type or a storage class. On ULTRIX systems, you can declare an identifier without specifying a storage class or a type, as shown:

  ```
  account;
  float profit;
  ```

  In the preceding example, the ULTRIX C compiler assumes that the account identifier is of type `extern int`.
- The Tru64 UNIX C compiler issues a warning message if you omit the ending semicolon in a structure declaration list, as shown:

  ```
  struct {int a,b} a;
  ```

  The following shows the correct syntax to use for a structure declaration list on Tru64 UNIX:

  ```
  struct {int a,b;} a;
  ```
- The Tru64 UNIX C compiler allows you to use a special `struct` declaration to declare two structures that reference each other.

  On ULTRIX systems, to declare two structures that reference each other within a block, you use a declaration similar to the following:

  ```
  struct x { struct y *p; /* ...  */ };
  struct y { struct x *q; /* ...  */ };
  ```

  If `struct y` is declared in an outer block, the first field of `struct x` refers to the declaration of `struct y` in the outer block.

  In some cases, you might want the first field of `struct x` to refer to the declaration of `struct y` that follows `struct x`. To allow this type of declaration, the Tru64 UNIX C compiler defines the following special declaration:

  ```
  struct y;

  struct x { struct y *p; /* ...  */ };
  struct y { struct x *q; /* ...  */ };
  ```

  The partial declaration, `struct y;` supersedes the declaration of `struct y` in the outer block. The compiler uses the next declaration of `struct y` it encounters to define the first field of `struct x`.

7.4.3 Differences Between Tru64 UNIX C and DEC C

When you compile an application on a Tru64 UNIX system that is compiled with DEC C on an ULTRIX system, you should notice few differences in how the program is compiled. Both compilers comply with the ANSI C language definition. However, you might notice some differences that result from implementation-specific features, standards-compatible extensions, or differences in interpretations of the ANSI standard.

To minimize the effect of any differences, use the Tru64 UNIX C compiler option that offers the most compatibility, as shown in Table 7–4.

Table 7–4: Compilation Options that Are Compatible with DEC C

If You Use This ULTRIX Option	Use This Tru64 UNIX Option
Default (ANSI C with a few compatible extensions)	`–std` (ANSI C with a few compatible extensions. Some differences exist between this mode and the `c89` default mode.)
`–std` (Strict ANSI)	`–std1` (Strict ANSI.)
`–common` (K&R C)	Default (`–std0`, which is K&R C with a few ANSI extensions.)
`–vaxc`	No equivalent.[a]

[a] For information about migrating applications written in the VAX C programming language on ULTRIX, see Section 7.4.5.

The following list describes some differences you might notice between the Tru64 UNIX C compiler and the DEC C compiler:

- The Tru64 UNIX C compiler supports function inlining when you use the `–O3` option. Function inlining eliminates the overhead associated with calling a procedure and allows the compiler to apply general optimization methods across functions.

  The DEC C compiler also supports function inlining; however, that compiler uses a heuristic approach to performing the inline expansion of function calls.

  Because of this implementation difference, the `–noinline` option has a different effect on Tru64 UNIX and ULTRIX systems. The option has no meaning on a Tru64 UNIX system, unless you also specify the `–O3` option. With DEC C, the option applies any time you use it.

- The Tru64 UNIX C compiler does not support using `#pragma` directives to control function inlining; that is, the compiler does not support the following DEC C syntax:

  #pragma inline (*function_name* [[, *function_name*...]])

  #pragma noinline (*function_name* [[, *function_name*...]])

- The Tru64 UNIX C compiler supports only predefined macros that begin with two underscore characters (_ _) when you use the `–std` option. Macro names that do not begin with two underscore characters are valid when you use the default compilation mode of the DEC C compiler.

- The Tru64 UNIX C compiler does not support the VAX C (`vcc`) compatibility mode keywords, language interpretations, or extensions. See Section 7.4.5 for information about differences between the (`vcc`) compiler and the Tru64 UNIX `cc` compiler.

- The Tru64 UNIX C compiler does not support the –check option for checking code similar to the way the lint command checks it. To check your Tru64 UNIX C code, use the lint command as described in lint(1).
- The Tru64 UNIX C compiler does not support the –source_listing or –show options for displaying source code listing and intermediate and final macro expansions.

7.4.4 Differences Between Tru64 UNIX C and C on VAX Systems

If you compile an application you wrote for the cc compiler on VAX ULTRIX systems with the Tru64 UNIX C compiler, you might notice some differences in the language definitions the compilers accept. Some of these differences are hardware specific, others are differences in how the compilers are implemented.

To minimize the effect of these differences, use the Tru64 UNIX C compiler option that offers the most compatibility, as shown in Table 7–5.

Table 7–5: Compilation Option that Is Compatible with C on VAX Systems

If You Use This ULTRIX Option	Use This Tru64 UNIX Option
Default (K&R C)	Default (–std0)—K&R C with ANSI extensions. Some differences exist due to differences between VAX and RISC systems and differences between the compilers.

The following list details the differences between the Tru64 UNIX C compiler when you use the –std0 option and the cc command on a VAX machine:

- The pointers on RISC systems are unsigned; on VAX systems they are signed.
- On RISC systems, you cannot dereference NULL pointers, including arguments to the strlen function.
- The varargs function on RISC systems is different from that function on VAX systems. Your application will fail if it walks an argument list by incrementing the address of an argument, particularly if the arguments are double-precision values. Use the macros in varargs.h when you use functions that have a variable number of arguments. Compiling with the –varargs option on RISC systems causes the compiler to detect nonportable code.
- The setjmp/longjmp buffer is larger on RISC systems than on VAX systems. Applications with a hard-coded, 10-word buffer fail; applications that include setjmp.h and declare a variable of type jmp_buf work correctly.

- RISC systems define a macro (for example, LANGUAGE_C) for the preprocessor that makes it possible to write multilingual include files.
- The `–Md` or `–Mg` option is not needed when on RISC systems. The software supports only one double-precision format.
- The Tru64 UNIX C compiler does not allow the following obsolete form of initialization:

  ```
  int i 0;
  ```

  The preceding example works on a VAX system, but the VAX `cc` compiler issues a warning. The Tru64 UNIX C compiler issues an error message.
- The Tru64 UNIX C compiler has boundary alignment rules. The only effect this difference should have on your application is that its performance might be slower than on a VAX system. This performance change could occur because the kernel corrects alignment errors. Where possible, align double words, words, and half words on natural boundaries.
- The Tru64 UNIX C compiler does not allow you to use a global data item as if it is a code location. For example, the compiler does not allow you to use a data structure that has the same name as a system call. If you use a global data item as a code location, the Tru64 UNIX C compiler displays an error message similar to the following one at load time:

  ```
  /lib/libc.a(gethostent.o): jump relocation out-of-range,
  bad object file produced, can't jump from 0x4197a0
  to 0x10008198 (stat)
  ```

  If you see this message, change the name of the data structure. (In this example, it was named `stat`.)
- The Tru64 UNIX C compiler does not allow the same `.c` or `.o` file to be listed twice in a command line. The compiler generates errors that indicate that symbols are defined more than once. The `cc` compiler on VAX systems allows you to specify the same source or object file twice.
- By default, the Tru64 UNIX C compiler links your application with shared libraries, instead of archive libraries. If you want your application to be linked with archive libraries, use the `–non_shared` option. For more information, see Section 8.1.
- The Tru64 UNIX `cc` command uses a different syntax for the `asm` pseudofunction call.
- On Tru64 UNIX systems, the `–L` option to the `cc` command operates only on the `–l` options that follow it. On VAX systems, the `cc –L` option affects all `–l` options. If you want the `–L` option to affect all `–l` options on the command line when you use the Tru64 UNIX C compiler, specify the `–L` option first.

- The Tru64 UNIX C compiler does not support the –R option (read-only text).
- The Tru64 UNIX Version 2.0 and earlier systems support two levels of profiling that you use by running the postprocessor `pixie` utility. Profiling on VAX systems has two levels that you select with the –p and –pg options. The Tru64 UNIX Version 3.0 system supports these levels of profiling, as well as the `pixie` utility.
- The Tru64 UNIX C compiler supports five levels of optimization, which are controlled using the –O option. The C compiler on VAX systems supports only one level of optimization, which is disabled by default and enabled with the –O option.

  By default, the Tru64 UNIX C compiler optimizes as if you specified the –O1 option. The optimization that the compiler performs is similar to the optimizations performed by the `cc` command on a VAX system. You disable optimizations by specifying the –O0 option when you use the Tru64 UNIX C compiler.
- The Tru64 UNIX C compiler offers four levels for debugging information (controlled by the –g option). The C compiler on VAX systems has only two (on and off).
- Both the Tru64 UNIX C compiler and VAX `cc` command support the –t and –B options for specifying passes and paths. However, the Tru64 UNIX C compiler has more pass names. In addition, the Tru64 UNIX C compiler option –h is equivalent to the VAX `cc` compiler option –B. The –B option to the Tru64 UNIX C compiler specifies a suffix for the pass name.

7.4.5 Differences Between Tru64 UNIX C and VAX C (vcc) Software

If you compile an application you wrote for the `cc` compiler on VAX ULTRIX systems with the Tru64 UNIX C compiler, you might notice some differences in how the compilers operate. Some of these differences are hardware specific, others are differences in how the compilers are implemented.

To minimize the effect of these differences, use the Tru64 UNIX C compiler option that offers the most compatibility, as shown in Table 7–6.

Table 7–6: Compilation Option for Compatibility with VAX C Software

If You Use This ULTRIX Option	Use This Tru64 UNIX Option
Default (VAX C on ULTRIX)	Default (–std0)—K&R C with ANSI extensions. Some differences exist due to differences between VAX and RISC systems and differences between the compilers.

The following list details the differences between the Tru64 UNIX C compiler when you use the –std0 option and the vcc command:

- The pointers on RISC systems are unsigned; on VAX systems they are signed.
- On RISC systems, you cannot dereference NULL pointers, including arguments to the strlen function.
- The varargs function on RISC systems is different from that function on VAX systems. Your application will fail if it walks an argument list by incrementing the address of an argument, particularly if the arguments are double-precision values. Use the macros in varargs.h when you use functions that have a variable number of arguments. Compiling with the –varargs option on RISC systems causes the compiler to detect nonportable code.
- The setjmp/longjmp buffer is larger on RISC systems than on VAX systems. Programs with a hard-coded, 10-word buffer fail; applications that include setjmp.h and declare a variable of type jmp_buf work correctly.
- RISC systems define a macro (for example, LANGUAGE_C) for the preprocessor that makes it possible to write multilingual include files.
- The –Md or –Mg option is not needed when on RISC systems. The software supports only one double-precision format.
- The Tru64 UNIX C compiler does not support the following VAX C keywords:
  - _align
  - globaldef
  - globalvalue
  - noshare
  - readonly
  - variant_struct
  - variant_union

- The Tru64 UNIX C compiler does not support the main_program option. On VAX ULTRIX systems, this option allows you to give the main function a different name.
- To be compatible, Tru64 UNIX C structure and union types must be identical. The vcc compiler treats structure and union types as compatible if they are the same size in bytes. The types need not be identical to be compatible.
- The Tru64 UNIX C compiler does not support applying the unary & (address-of) operator to a constant in the argument list of a function call. The vcc compiler supports this use of the & operator.
- The Tru64 UNIX C compiler replaces comments that separate tokens in a macro definition with one space character during preprocessing. Therefore, you cannot use a comment as a concatenation character on Tru64 UNIX systems.

  On VAX ULTRIX systems, comments that separate tokens within a macro definition are deleted with no spaces. This action allows you to use a comment as a concatenation character.

  On Tru64 UNIX systems, replace a comment that you use as a concatenation character with the ANSI-defined concatenation characters (##).
- On Tru64 UNIX systems, you can redefine a macro only if the token you use in the new macro definition is identical to the token you used in the existing macro definition. The vcc compiler allows you to redefine macros.
- The Tru64 UNIX C and vcc compilers use a different algorithm for substituting macro definitions. These different algorithms might cause you to notice differences in how your macros are processed on a Tru64 UNIX system.
- By default, the Tru64 UNIX C compiler links your application with shared libraries, instead of archive libraries. If you want your application to be linked with archive libraries, use the –non_shared option. For more information, see Section 8.1.
- The Tru64 UNIX Version 2.0 and earlier systems support two levels of profiling that you use by running the postprocessor pixie utility.

  Profiling on VAX systems has two levels that you select with the –p and –pg options. The Tru64 UNIX Version 3.0 system supports these two levels of profiling as well as the pixie utility.
- The Tru64 UNIX C compiler supports five levels of optimization, which are controlled by using the –O option. The vcc compiler supports only one level of optimization, which is disabled by default and enabled with the –O option.

By default, the Tru64 UNIX C compiler optimizes as if you specified the –O1 option. The optimization the compiler performs is similar to the optimizations performed by the vcc command. You disable optimizations by specifying the –O0 option when you use the Tru64 UNIX C compiler.

- The Tru64 UNIX C compiler offers four levels for debugging information (controlled by the –g option). The vcc compiler has only two (on and off).
- Both the Tru64 UNIX C compiler and the ULTRIX vcc command support the –t and –B options for specifying passes and paths. However, the Tru64 UNIX C compiler has more pass names. In addition, the Tru64 UNIX C compiler option –h is equivalent to the vcc compiler option –B. The –B option to the Tru64 UNIX C compiler specifies a suffix for the pass name.
- The Tru64 UNIX C compiler does not produce a listing that contains the source code, symbol table, machine code, and cross-reference information.

7.5 Running lint to Find Other Errors

After you create object files for your application, use the lint command to find other problems. The lint command gives you information about whether you use data types correctly in your application, whether you use routines and variables correctly, whether there are any 64-bit migration problems, and so on.

The –Q option is included as support for migrating ULTRIX applications to the Tru64 UNIX system by identifying those programming techniques that might cause problems on a 64-bit Tru64 UNIX system. The techniques identified include: pointer alignment; pointer and integer data type combinations; assignments that cause a truncation of long data types; assignments of long data types to another type; structure and pointer combinations; type castings; and format control strings in scanf and printf calls.

Be aware that if you never used lint on your ULTRIX application, it might give you quite a bit of information about your Tru64 UNIX application, some of which will not be pertinent to the problems with porting your application.

For more information about using lint, see lint(1).

7.6 Linking Your Program

Use the cc compiler to link your application. The linker reports errors caused by routines that do not exist on a Tru64 UNIX system or by global symbols that are undefined. In some cases, these errors occur because the Tru64 UNIX system does not provide a routine or a global symbol definition. In other cases, the name of the routine or global symbol has changed.

To determine whether a routine exists, see the Tru64 UNIX documentation. Check the documentation carefully because the Tru64 UNIX system has some routines or symbols that use names different from those on the ULTRIX system. If a Tru64 UNIX routine or symbol exists that performs the task that the ULTRIX routine or symbol performs, modify your program. Replace each reference to the ULTRIX routine or symbol name with the appropriate Tru64 UNIX routine or symbol name. As you make this change, check each call to ensure that it passes the correct number of parameters in the correct order and that the parameters have the appropriate data type.

If no routine exists on the Tru64 UNIX system, remove the routine from your application and make appropriate modifications to your applications.

Some ULTRIX libraries are unavailable on Tru64 UNIX systems. In some cases, the routines that are in the ULTRIX libraries are available in a different Tru64 UNIX library. In other cases, the ULTRIX library routines are unavailable on the Tru64 UNIX system. Section 7.6.1 describes ULTRIX libraries that are unavailable on Tru64 UNIX systems.

The Tru64 UNIX system provides two libraries for compatibility with ULTRIX systems:

- The `libbsd.a` library contains routines that are compatible with the ULTRIX BSD programming environment. (Section 7.6.2 describes this library.)
- The `libsys5.a` library contains routines that are compatible with the ULTRIX System V programming environment and other System V programming environments. (Section 7.6.3 describes this library.)

You might need to link your application with one of these libraries if it depends on the behavior of a BSD or System V library routine.

7.6.1 Changes in Libraries

The following list summarizes differences between ULTRIX and Tru64 UNIX system libraries:

- Merger of libraries into the `libc` library

  Unlike ULTRIX systems, the internationalization library, `libi.a`, the POSIX library, `libcP.a`, and the System V library, `libcV.a`, are part of the standard C library, `libc`, except where conflicts between System V and other standards exist.

  Remove references to these libraries from your `cc` or `make` command line.

- Separation of libraries from the `libc` library

  Unlike ULTRIX systems, the `libmld` library is not part of the standard C library, `libc`.

Add a reference to this library in your cc or make command line if you want to include these functions.

- Movement of functions between libraries

  On ULTRIX Version 4.3A and earlier systems, the DXmHelpSystemOpen, DXmHelpSystemDisplay, and DXmHelpSystemClose functions were contained in the libDXm. On Tru64 UNIX systems, these functions are contained in libbkr. (This difference does not apply to ULTRIX Version 4.4 systems.)

- Libraries supporting unavailable features

  A number of libraries are not included in the Tru64 UNIX system due to differences in features between ULTRIX and Tru64 UNIX systems. These include:

  – Extended SNMP Agent: libsnmp.a

  – Kerberos library routines: libkrb.a, libknet.a, libdes.a, and libacl.a

  – Authorization library: libauth.a

  – ULTRIX/SQL library: libsql.a

  – Graphics and plotting libraries (located in /usr/lib on ULTRIX systems): plot, plotaed, plotbg, plotdumb, plotgigi, plotgrn, plot2648, plot7221, plotimagen, 300, 300s, 450, 4013, 4014, and lvp16

  You must remove calls to routines in these libraries from your application if you want to compile it on a Tru64 UNIX system. Also, be sure to omit references to these libraries from the command line you use to build the application.

7.6.2 ULTRIX BSD Compatibility Library

The Tru64 UNIX system provides the libbsd.a library to allow you to use library routines that are compatible with ULTRIX BSD library routines. Table 7–7 lists the routines in the library and describes the BSD compatibility they offer. The most significant behavior of the routines in this library are siginterrupt() and signal(), which restart system calls that are interrupted by signals. (The default, in compliance with the POSIX standard, is not to restart system calls that are interrupted by signals.)

To use the BSD functions, use the –D_BSD and –lbsd options on the compilation line.

Table 7–7: Routines in the ULTRIX BSD Compatibility Library

Routine Name	Compatibility
`int ftime(struct timeb *)`	Allows your application to continue to use the `ftime` function, which is not otherwise provided on Tru64 UNIX systems. This feature has been made obsolete by the `gettimeofday()` function.
`char *mktemp(char *)`	Constructs a unique file name; expects a string of at least six characters with trailing 'X' characters, and overwrites the 'X' characters with a unique encoding of the process's process identification (PID) and a pseudorandom number. Unlike the standard Tru64 UNIX `mktemp()`, this routine is not thread safe.
`int nice(int)`	Returns a value in the range from –20 to 20. By default, the Tru64 UNIX system defines process priorities in the range from 0 to 39. This is the same range defined on System V systems. Additionally, if the `libc` version of `nice()` fails, `errno` may be set to the same values as by the `setpriority()` function.
`int rand()` `void srand(u_int seed)`	The `rand()` routine returns a number in the range of 0 to 2^{31} –1. The `srand()` routine provides a seed for the random number generator.
`char *re_comp(char *)`	Converts a string into an internal form suitable for pattern matching. Returns 0 if the string was compiled successfully; otherwise, returns a pointer to an error message.
`int re_exec(char *)`	Compares the string parameter with the last string passed to the `re_comp()` function. Returns 1 if the string matches the last compiled regular expression. (The default returns 1 when the string *fails* to match the regular expression.) Returns 0 if the string fails to match the last compiled regular expression. (The default returns 0 if the string *does* match the regular expression.) Returns –1 if the compiled regular expression is invalid (indicating an internal error).

Table 7–7: Routines in the ULTRIX BSD Compatibility Library (cont.)

Routine Name	Compatibility
`int siginterrupt(int, int)`	Allows you to set the signal state so that system calls are restarted if they are interrupted by the specified signal and no data has been transferred.
`sig_t signal(int, sig_t)`	Causes the system to preserve the value of the SA_RESTART flag if your process explicitly enables or disables system call restart by using the `siginterrupt()` call.
`char *timezone(int, int)`	The arguments are the number of minutes of time you are westward from Greenwich and whether daylight saving time (DST) is in effect. Returns a string giving the name of the local time zone. Provided for compatibility only.
`char * valloc(size_t)`	Allocates bytes aligned on a page boundary. Provided for compatibility only.
`int vtimes(struct vtimes*, struct vtimes*)`	Returns accounting information for the current process and for the terminated child processes of the current process. Provided for compatibility only; superseded by the `getrusage()` function.

`MINT * xtom(char *key)`
`char * mtox(MINT *key)`
`void mfree(MINT *a)`
	Provided for BSD compatibility for performing arithmetic on integers of arbitrary length.

`int wait(union wait *)`	Provides a `wait` call whose *`status_location`* parameter is of type `union wait *`.

7.6.3 System V Compatibility Library

The Tru64 UNIX system provides the `libsys5.a` library to allow you to use library routines that are compatible with System V library routines. Table 7–8 lists the routines in the library and describes the System V compatibility they offer. This library contains routines for those `libc` routines whose behavior is incompatible with POSIX or X/Open standards. The ULTRIX system also provides a System V compatibility library, `libcV.a`, which supplies a number of features similar to those that `libsys5.a` provides. The most significant behavior of the routines in this library is the compatibility with System V nonblocked signals.

For more information about the System V (SVID-2) features in Tru64 UNIX systems, see the *System V Compatibility User's Guide*.

Table 7–8: Routines in the System V Compatibility Library

Routine Name	Compatibility
`int mknod(char *, int , int )`	Supports passing of `mode` and `dev` as an `int`, instead of `mode_t` and `dev_t`, respectively.
`char * mktemp(char *)`	Uses `getpid()` to generate a unique file name. Is not thread safe.
`int mount(char *, char *, int, char *, char *, int)`	Does not support specifying the type of file system, mount flags, such as M_RDONLY and M_NOEXEC, or mount data. Allows you to specify whether the file system is a read-only or read/write system. Also provides SVID-2 compatibility via the MS_DATA flag.
`int ptrace(int, int, int, int)`	Supports passing of `pid` as an `int` type rather than `pid_t`.
`int rmdir(const char *)`	Sets the value of the global variable `errno` to EEXIST if the directory to be removed contains entries other than dot (`.`) and double dot (`..`).
`int setjmp(jmp_buf) void longjmp(jmp_buf, int)`	Do not save and restore the signal state.
`pid_t setpgrp(void)`	If this call is successful, it returns the new process identification (PID).
`void (*signal(int, int(*func()) ) )`	The specified signal is not blocked from delivery when the handler is entered, and the disposition of the signal reverts to SIG_DFL when the signal is delivered.
`int unlink(const char *)`	An attempt to unlink nonempty directories will cause the `unlink` call to fail and set `errno` to ENOTEMPTY, even if the process has superuser privileges.
`int umount(char *)`	Does not support the MNT_NOFORCE, MNT_WAIT, MNT_FORCE, or MNT_NOWAIT flags.

7.7 Porting Terminal-Dependent Applications

The Tru64 UNIX system supports two versions of the `termcap` library and two versions of the `curses` library. To use the default `termcap` library (similar to the BSD 4.3 `termcap` library), use the `–ltermcap` option in the compilation line. To use the BSD 4.3-5 `termcap` `curses` functions (similar to ULTRIX Version 4.2), use the `–D_BSD` and `–lcurses` options in the compilation line. The ULTRIX system supports one version of the `termcap` library and two versions of the `curses` library:

- The X/Open `curses` functions, which are part of the `cursesX` library
- The BSD 4.2 `curses` functions, which are part of the `curses` library

Table 7–9 helps to clarify how to port ULTRIX specific applications to the Tru64 UNIX system.

Table 7–9: Terminal Capability Differences

If You Use this ULTRIX Option	Use this Tru64 UNIX Option	Library Used by C Compiler
–ltermcap or –ltermlib	-D_BSD –ltermcap or -D_BSD–ltermlib	BSD 4.2 termcap library (IBM AIX library on a Tru64 UNIX system; similar to BSD 4.3 library)
-D_BSD -lcurses –ltermcap or -lcurses –ltermlib	–D_BSD –lcurses -ltermlib	BSD 4.2 termcap and curses libraries (BSD 4.3-5 curses and termcap functions on a Tru64 UNIX system)
–lcursesX	–lcurses	X/Open curses library (System V Release 3 curses and terminfo functions on a Tru64 UNIX system)
–lcurses	–D_BSD –lcurses	BSD 4.2 curses library (BSD 4.3-5 curses functions on a Tru64 UNIX system)

In addition, the /usr/include/cursesX.h header file is replaced by /usr/include/curses.h, so that you must change all pertinent cursesX references in your source files and makefile.

7.8 Differences in Standard Interfaces

As described earlier, there are different versions of some library calls included for compatibility with the ULTRIX system. There are a few areas where ULTRIX specific library behavior is not in the Tru64 UNIX system. The following list describes the known differences in library behavior that are not reflected by changes in the call interface or header file. These differences require that you change your source code.

- The ULTRIX sprintf routine returns its first argument for success and end-of-file (EOF) for failure. The Tru64 UNIX sprintf routine returns the number of displayable characters in the output (not necessarily the number of bytes) for success and a negative number for failure. The number returned for success does not include the terminating \0 character.
- The printf, sprintf, and fprintf routines do not support the use of the %D parameter. If applications use the %D parameter to display a long number in decimal format, the routines print the character D instead of the number. Instead, use the %d or %ld parameter in your print routines.
- On ULTRIX systems, if you call malloc for a zero length buffer, a pointer to the buffer is returned. The Tru64 UNIX malloc call returns a NULL pointer and sets errno to EINVAL.
- On ULTRIX systems, the default definition of the getpgrp system call is:

```
int getpgrp(pid_t, pid_t)
```

The POSIX-conformant definition of getpgrp on Tru64 UNIX systems states that getpgrp is called without arguments and returns the process group of the current process:

```
pid_t = getpgrp();
```

The ULTRIX system's mechanism for setting a process's group ID is:

```
void = setpgrp(int, int);
```

This system call is supported on Tru64 UNIX systems for compatibility only. In new applications, use the POSIX-standard setgpid call:

```
pid_t = setpgid(pid_t, pid_t);
```

- On ULTRIX systems, read operations on directories are supported by the following statements:

  ```
  #include <sys/dir.h>
  struct direct *readdir(dirp);
  DIR *dirp;
  ```

  On Tru64 UNIX systems, read operations on directories are supported by the following statements:

  ```
  #include <sys/dirent.h>
  struct dirent *readdir(DIR *dirp);
  ```

  See opendir(3) for more information.

- On Tru64 UNIX systems, the setsysinfo and getsysinfo system calls have been expanded to provide unaligned access control similar to that found on ULTRIX systems. In addition, SSI_UACPROC and SSI_UACPARNT options accept three other options as arguments:

  – UAC_NOPRINT

    Suppresses the printing of the unaligned error message to the user.

  – UAC_NOFIX

    Instructs the operating system not to fix the unaligned access fault.

  – UAC_SIGBUS

    Forces a SIGBUS signal to be delivered to the thread.

  These options are defined in sys/proc.h, and can be specified in any combination on a per task basis.

  UAC settings are inherited by children, so forked processes will have the same UAC characteristics as their parent. The SSI_UACSYS option only accepts the UAC_NOPRINT option and suppresses unaligned fixup messages regardless of the users' setting. Only the superuser is allowed to use this option.

  The following example shows the setsysinfo call usage in an application:

```
#include <sys/sysinfo.h>
#include <sys/proc.h>
   .
   .
   .
int buf[2], val, arg;
   .
   .
   .
/* Do not print the warning to the user */
buf[0] = SSIN_UACPROC;
buf[1] = UAC_NOPRINT;
error = setsysinfo(SSI_NVPAIRS, buf, 1, 0, 0);
   .
   .
   .
/* Do not print the warning and deliver a SIGBUS signal */
buf[0] = SSIN_UACPROC;
buf[1] = UAC_NOPRINT | UAC_SIGBUS;
error = setsysinfo(SSI_NVPAIRS, buf, 1, 0, 0);
   .
   .
   .
```

- On ULTRIX systems, the `catopen` routine opens a message catalog and returns a catalog descriptor if successful. On Tru64 UNIX systems, the `catopen` routine does not open the message catalog. Instead, it is the `catgets` routine that opens a message catalog. Therefore, if your application checks whether a message catalog was successfully opened, you must change your program to reflect this change. For example, the following ULTRIX code will not work on a Tru64 UNIX system:

  ```
  catd = catopen("example.cat", 0);
  if (catd == (nl_catd) -1)
      /* message catalog was not opened */
  else
       /* message catalog was opened */
  ```

  The following code shows how the previous code is modified to use the `catgets` routine:

  ```
  catd = catopen("example.cat", 0);
  if (catgets(catd, 1, 1, NULL) == NULL)
      /* message catalog was not opened */
  else
      /* message catalog was opened */
  ```

- The manner for establishing controlling terminals is an implementation-defined process that is different for Tru64 UNIX and ULTRIX systems. On the Tru64 UNIX system (and according to the POSIX standard), a process must be a session leader to establish a controlling terminal. The Tru64 UNIX system defines allocation of a control terminal with an explicit call to `ioctl()`. When porting source code, you need to obtain a controlling terminal in the following way:

```
(void) setsid();
fd = open("/dev/tty01", O_RDWR );
(void) ioctl(fd, TIOCSCTTY, 0);
```

- The manner for establishing and controlling modem connections is different for Tru64 UNIX and ULTRIX systems. The Tru64 UNIX system uses POSIX conventions for modem control. Information about a serial line can be inspected and altered in the POSIX `termios` structure, using the `tcgetattr()` and `tcsetattr()` library routines. On ULTRIX systems, modem control was accomplished using the TIOCCAR, TIOCNAR, and TIOCWONLINE requests to the `ioctl()` system call. These requests are not supported on a Tru64 UNIX system. When porting source code, open a serial line in the following manner:

  ```
  fd = open(ttyname,O_RDWR | O_NONBLOCK);
  ```

  The O_NONBLOCK flag enables you to complete a read request, in case the CLOCAL flag is not set and you are monitoring the modem status lines.

  Get the current line attributes; set the CLOCAL flag, in case it is not already set; and turn off the O_NONBLOCK flag in the following manner:

  ```
  tcgetattr(fd,&tty_termios); /* get current line attributes */
  if ((tty_termios.c_cflag & CLOCAL) == 0) {
      tty_termios.c_cflag |= CLOCAL;
      tcsetattr(fd,TCSANOW,&tty_termios);
  }
  flags = fcntl(fd, F_GETFL)
         fcntl(fd, F_SETFL, flags & ~O_NONBLOCK)
  ```

  You can now use your implementation-defined process for dialing the phone number and negotiating with the modem. After this, monitor the modem signals by doing the following:

  ```
  tty_termios.c_cflag &= ~CLOCAL;
  tcsetattr(fd,&tty_termios); /* watch for modem signals now */
  alarm(40);   /* set a timer; do not wait forever */
  read(fd,buffer,count);  /* this read() blocks, pending the
         appearance of modem signals */
  alarm(0);   /* turn off timer */
  ```

  See Appendix D for a comparison of an ULTRIX application using modem control and a Tru64 UNIX application using modem control. The comparison is for an outgoing call. In addition, Appendix D also contains a sample application for an incoming call.

- The Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) transport providers are supported by both ULTRIX and Tru64

UNIX X/Open Transport Interface (XTI). However, when creating a transport end point with the t_open call, ULTRIX does not need device information to specify a transport provider. In the Tru64 UNIX system, this information must be present because XTI is implemented using the xtiso pseudostreams driver.

You must change all your t_open calls to reflect this change for both TCP and UDP transport providers or change your application to determine the end point at run time. For example:

```
#ifdef __osf__
  t_open ("/dev/streams/xtiso/tcp", ...)
#else
  t_open ("tcp", ...)
#endif
```

- On ULTRIX systems, XTI is layered on sockets. If you call select for an incoming asynchronous XTI connection, the socket becomes writable. On Tru64 UNIX systems, XTI is layered on streams. If you call select, the socket becomes readable. You can either modify your application to work with the Tru64 UNIX select call or substitute the poll call for the select call and modify your application to use this call. See the *Network Programmer's Guide* for more information on XTI.
- The ULTRIX ccmn_ccbwait() function is replaced by the Tru64 UNIX ccmn_send_ccb_wait() function. The Tru64 UNIX function sends a CAM Control Block (CCB) to the transport (XPT) layer and sleeps on the address of the CCB at the passed priority level, waiting for the CCB to complete. For more information, see the *Writing Device Drivers for the SCSI/CAM Architecture Interfaces* manual.
- On ULTRIX systems, if you call open with a null pathname, it defaults to the current directory. On Tru64 UNIX systems, if you call open with a null pathname, it returns an error.

7.9 Running Your Program

After your application links successfully, you are ready to run and test it. Correct run-time errors by using the dbx debugger as an aid.

After you correct the semantic errors, your application is ported to the Tru64 UNIX system. In some cases, it might still not work properly. One possible problem area is differences in the way certain routines on Tru64 UNIX systems are called or the return values. See Section 7.8 and Appendix B for more information.

8

Postmigration Programming Features

After you migrate your source code from an ULTRIX to a Tru64 UNIX system, you might want to enhance it by using features of the Tru64 UNIX system. This chapter gives an overview of using three Tru64 UNIX features: shared libraries, semaphores, and the number of open file descriptors.

For complete information on using Tru64 UNIX features, see the *Programmer's Guide*.

8.1 Using Shared Libraries

Shared libraries allow several applications to use a single copy of a library routine at run time. Shared libraries help save disk space and memory, and they can improve the performance of your application and system.

Using Tru64 UNIX shared libraries is similar to using archive libraries.

To link your application with a shared library, you must have compiled it on a Tru64 UNIX system. Therefore, you must recompile and relink ULTRIX applications if you want them to use shared libraries.

This section describes how to use the `cc` command to link with a shared library. It also describes how to create shared libraries.

For complete information about using shared libraries, see the *Programmer's Guide*.

8.1.1 Linking with Shared Libraries

On Tru64 UNIX systems, the `cc` command links your application with shared libraries by default. The following example shows the command you enter to link with the shared version of `libc`:

```
% cc -o hello hello.c
```

This command creates an executable file named `a.out`, which you run.

You can also link your application with a shared library that you create. For example, suppose you create a shared library named `libspecial_math.so` and store that library in the directory `/usr/person`. To link with that library, use the `–l` and `–L` options, as shown:

```
% cc -o hello hello.c -L/usr/person -lspecial_math
```

To link the application that does not use shared libraries, you must specify the –non_shared option in the cc command line, as shown:

```
% cc -non_shared -o hello hello.c -L/usr/person -lspecial_math
```

Although shared libraries are the default for most applications, the following applications cannot use them:

- Applications that need to run in single-user mode cannot be linked to shared libraries because the /usr/shlib directory must be mounted to allow access to the shared libraries.
- Applications whose primary purpose is single-user benchmarks should not be linked with shared libraries because position-independent code is less efficient than position-dependent code. Also, there is no benefit to sharing memory when only one application is running.
- Real-time applications using memory-locking features should not be linked to shared libraries. Memory-locking functions will lock the entire shared library into memory.

8.1.2 Symbol Resolution and Shared Libraries

If you link your application with shared libraries instead of archive libraries, you might notice some differences in the way symbols are resolved. This section describes these differences.

8.1.2.1 How Libraries Are Searched

The shared libraries supplied with Tru64 UNIX systems are stored in the /usr/shlib directory. Place all system shared libraries in this directory to avoid confusion. When the linker searches for files that have been specified using the –l option, it searches the following directories, in order:

- /usr/shlib
- /usr/ccs/lib
- /usr/lib/cmplrs/cc
- /usr/lib
- /usr/local/lib

The linker searches all of the directories for a shared library (an .so file). If it does not find a shared library with the specified name, the linker searches all of the directories a second time for a static (an archive) library (an .a file).

When you develop applications, you might work with private shared libraries that are contained in directories other than the /usr/shlib directory. In this case, use the –L option to specify these directories. Before you execute the program, set the LD_LIBRARY_PATH environment variable to point to the directory containing the private shared libraries. When the program is executed, the run-time loader, /sbin/loader, examines this environment variable and searches the path, if defined, before searching the default list of directories.

Set the LD_LIBRARY_PATH variable in the following ways:

- Enter the setenv command at the system prompt, followed by a colon-separated path. The following example sets the path as current directory, $HOME/testdir directory (if defined), and the default shared library directory. For example:

  ```
  % setenv LD_LIBRARY_PATH .:$HOME/testdir:/usr/shlib
  ```

- Add the variable definition to your login or shell startup files. For example, you might add the following line to your .login or .cshrc file, if you work in the C shell:

  ```
  setenv LD_LIBRARY_PATH .:$HOME/testdir:/usr/shlib
  ```

In the following examples, the /usr/person directory contains two versions of the special math library: libspecial_math.so is a shared library and libspecial_math.a is an archive library.

When you link with a shared library, symbols must be referenced before the linker searches the shared library. Otherwise, the linker does not find the symbol in the shared library and lists the symbol as undefined.

For example, suppose your library object file, libspecial_math.o, defines two functions, getvalue and setvalue. Suppose that you create a shared library, libspecial_math.so, and an archive library, libspecial_math.a, from the object file. You call the getvalue routine in the program1 module of your application, and you call the setvalue routine in the program2 module of your application.

Suppose you link your application using the archive library, as follows:

```
% cc -non_shared program1.o -lspecial_math program2.o
```

The application module program1 references the getvalue routine, which the libspecial_math archive library defines. That library also defines the setvalue routine, and the linker is able to define setvalue when it encounters that symbol in the program2 module.

Now, suppose you enter the same command, but use the shared library instead of the archive library:

```
% cc program1.o -lspecial_math program2.o
```

This command succeeds, but prints a warning message indicating that the symbol is undefined.

To correctly link this application, enter the following command:

```
% cc program1.o program2.o -lspecial_math
```

In general, always specify the `-l` option last in the command line.

8.1.2.2 When Symbols Are Defined More than Once

Symbol name resolution when using shared libraries is similar to name resolution when using static libraries. Symbol names are resolved according to the order in which the object file or shared object containing the symbols appears on the command line. The linker typically takes the leftmost definition for any symbol that must be resolved.

The sequence in which names are resolved proceeds as if the `link` command line were stored in the executable. When the program runs, all referenced symbols must be resolved. The program aborts if any undefined symbols are referenced.

When you link your application with shared libraries, do not define the same symbol twice. For example, you cannot use the following `cc` command to link an application that contains a shared library:

```
% cc program1.o libspecial_math.so program2.o libspecial_math.a
```

This command succeeds, but prints warning messages indicating that a symbol is defined multiple times.

8.1.3 Using Your Makefile with Shared Libraries

If you use the `make` command to build your ULTRIX application, you can use it to build a Tru64 UNIX application that uses shared libraries. You need not modify your `makefile` file to use it with shared libraries. Unlike ULTRIX systems, linking with shared libraries is the default on Tru64 UNIX systems.

The following example shows a `Makefile` file that links with shared libraries on a Tru64 UNIX system:

```
# Makefile for the Math Program
all: math.h program1.o program2.o
  cc program1.o program2.o -L/usr/person -lspecial_math

program1.o: project.h
  cc -c program1.c
```

8.1.4 Creating Shared Libraries from Object Files

To create a shared library:

1. Create one or more source files that define the routines you want to include in the library.
2. Compile the source file into an object file, as shown:

```
% cc -c special_math.c
```

3. Create the library by using the ld command. (You cannot use the cc command to create a shared library. You must invoke the ld command directly.) The following shows a sample ld command:

   ```
   % ld -shared -no_archive -o libspecial_math.so
   special_math.o -lc
   ```

   In this example, the −shared option specifies creating a shared (rather than an archive) library. The −no_archive option tells the linker to resolve all symbols from shared libraries only. The −o option specifies the name of the shared library.

   For this command to succeed without printing warning messages, all symbols in the special_math.o object must be resolved. In this case, the special_math.o object references symbols that are defined in libc. The −lc option specifies that ld search libc to resolve those symbols. The ld linker searches the /usr/shlib directory for libc, by default.

   If the shared library you are creating references symbols defined in another shared library, you must name the other shared library in the ld command line. Name the shared library last in the command line to ensure that the linker encounters the reference to the symbol before it encounters the definition of the symbol.

For more information on using ld to create shared libraries, see ld(1).

8.1.5 Creating Shared Libraries from Archive Libraries

You can also create a shared library from an existing static (archive) library by using the ld command. The following example converts the static library, old.a, into the shared library, libold.so:

```
% ld -shared -no_archive -o libold.so -all old.a -none -lc
```

In this example, the −all option tells the linker to link all objects from the old.a archive library. The −none option tells the linker to turn off the −all option. The −no_archive option applies to the resolution of the −lc option, but not to old.a, since it is explicitly mentioned.

8.1.6 Optimizing Application Startup when Using Shared Libraries

Your application starts more efficiently if your shared libraries can be loaded at a preassigned starting address in virtual memory. To allow this efficiency, the ld linker preassigns a starting address to each shared library you create.

At application startup time, a shared library’s preassigned starting address may already be in use. In this case, the system assigns a new starting address to the library and the advantage of the preassigned starting address is lost.

To make it more likely that a shared library can use its preassigned starting address, you can cause the ld linker to assign a unique starting address to each shared library you create.

If you create a shared library that only you will use, use the –check_registry option in the ld command line. This option causes ld to search the file you specify to determine what starting addresses are assigned to shared libraries. The linker then assigns an unused starting address to your shared library. The following example shows how to use the –check_registry option:

```
% ld -shared -no_archive -check_registry \
/usr/shlib/so_locations \
libspecial_math.so special_math.o -lc
```

If the shared library you create will be used by other programmers on your system, use the –update_registry option. This option causes the ld linker to search the file you specify to determine what starting addresses are assigned to shared libraries. The linker then assigns an unused starting address to your shared library. The linker then adds to the file the information that your shared library has been assigned that starting address. Because that information is stored in the file, the linker can determine that the address is already assigned when it assigns a starting address to other shared libraries.

If no -check_registry or -update_registry option is specified when building a shared library, the linker defaults to the -update_registry option and the ./so_locations file.

The following list describes the procedure you follow to use the –update_registry option with the system's /usr/shlib/so_locations file:

1. Copy the system's so_locations file to your local area:

   ```
   % cp /usr/shlib/so_locations .
   ```

2. Give yourself write access to the file:

   ```
   % chmod +w so_locations
   ```

3. Create the shared library and use the –update_registry option:

   ```
   % ld -shared -no_archive -update_registry \
   ./so_locations -o libspecial_math.so \
   special_math.o -lc
   ```

4. Move the so_locations file back to the /usr/shlib directory:

   ```
   % mv /home/smith/so_locations /usr/shlib/so_locations
   ```

You must have write privileges to the /usr/shlib directory to move the so_locations file into it. If you cannot write to the directory, ask your system administrator to move the file.

8.2 Using Semaphores

On an ULTRIX system, you use semaphores through the atomic_op call. This call allows you to test and set a user-space address that you specify. The Tru64 UNIX system contains the atomic_op call; however, the system also includes library routines that perform semaphore operations.

Modify your source code to use the Tru64 UNIX library routines rather than the atomic_op system call. The library routines are more portable than the atomic_op system call, which might not be included in all Tru64 UNIX systems. The library routines are also more powerful than the atomic_op system call.

The Tru64 UNIX library routines are as follows:

- msem_init, which initializes a semaphore
- msem_lock, which locks a semaphore
- msem_unlock, which unlocks a semaphore
- msem_remove, which removes a semaphore

For more information about these routines, see msem_init(3), msem_lock(3), msem_unlock(3), and msem_remove(3).

8.3 Using File Descriptors

On both the ULTRIX and UWS and the Tru64 UNIX systems, the number of open file descriptors a process can use is configurable. By default, the number for Tru64 UNIX systems is 4096; on ULTRIX systems the default is 64. Your system administrator configures the number of open file descriptors. For information about configuring this number, see the *System Administration* manual.

Because the system administrator can configure the maximum number of open file descriptors your processes can use, you might want to modify your program before you recompile it on a Tru64 UNIX system. The following list describes the changes needed:

- Use the getdtablesize call to determine the maximum number of open file descriptors configured on the system.

  The following example shows a call to getdtablesize to get the maximum number of open file descriptors:

```
int maxfds;

maxfds = getdtablesize():
```

Use the `maxfds` variable in other calls, such as the `select` call, that require you to pass the number of open file descriptors of interest. For more information, see `getdtablesize`(2).

- Use a short integer or longer data type to store file descriptors.

  On ULTRIX systems, you might have used a character data type to store file descriptors. Because file descriptor values on Tru64 UNIX systems can be greater than 128, you must use at least a short integer to store file descriptors.

- Use the `fd_set` data type and its associated macros as defined in the `/usr/include/sys/types.h` file to declare parameters to the `select` call.

  Using the `fd_set` data type ensures that the parameters are large enough to accommodate up to 4096 file descriptors. The `fd_set` data type is large enough for 64 file descriptors in ULTRIX. For more information, see `select`(2).

Part 5

Appendixes

This part contains appendixes that describe:

- The differences between specific Tru64 UNIX and ULTRIX commands (Appendix A)
- The differences between specific Tru64 UNIX and ULTRIX header files and routines (Appendix B)
- The differences between specific Tru64 UNIX and ULTRIX system calls (Appendix C)
- The differences between ULTRIX and Tru64 UNIX system terminal modem control (Appendix D)
- The differences between the Motif and XUI graphical user interfaces (GUIs) (Appendix E)
- The DECwindows Motif component names (Appendix F)
- Migration issues between ULTRIX Version 4.5 and Tru64 UNIX Version 4.0 (Appendix G)

A

Differences Between Tru64 UNIX and ULTRIX Commands

This appendix describes the differences between Tru64 UNIX commands and ULTRIX commands. In some cases, the difference is that an ULTRIX command does not exist on a Tru64 UNIX system. Other differences are caused by the options provided for a command being different or by some difference in the arguments to a command. For example, the Tru64 UNIX command might expect a different name for a particular argument than the ULTRIX command. Some commands operate differently on Tru64 UNIX systems than they do on ULTRIX systems.

To use the table in this appendix, look for the name of an ULTRIX command in the left-hand column of the table. Read the second column of the table to determine what difference exists between the Tru64 UNIX and ULTRIX commands. Read the right-hand column to determine how to get the effect of the ULTRIX command on a Tru64 UNIX system.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
2780d	Not supported.	No Tru64 UNIX equivalent.
2780e	Not supported.	No Tru64 UNIX equivalent.
3780d	Not supported.	No Tru64 UNIX equivalent.
adb	Not supported.	Use the dbx debugger.
addnode	Not supported.	Supported in the DECnet/OSI for Tru64 UNIX product.
ansi_ps	Not supported.	No Tru64 UNIX equivalent.
arff	Not supported.	No Tru64 UNIX equivalent.
audgen	Not supported.	Auditing not supported.
auditd	Not supported.	Auditing not supported.
auditmask	Not supported.	Auditing not supported.
audit_tool	Not supported.	Auditing not supported.
backup	Not supported.	No Tru64 UNIX equivalent.
bad144	Not supported.	No Tru64 UNIX equivalent.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
biod	Not supported.	Use the nfsiod command.
bootparamd	Not supported.	No Tru64 UNIX equivalent.
catman	The catman command automatically processes source reference pages by using tbl, neqn, and nroff −Tlp −h. It does not process through col. The catman command formats reference pages for the generic man/catman device −Tlp, which defaults to formatting for VT100 terminals rather than for the Teletype Model 37 terminal, which is not supported.	Do not preprocess sources through tbl or neqn before placing them in /usr/share/man/... directories. Postprocessing with col can be necessary for non-Compaq devices.
	Reference pages are formatted for online viewing rather than for printing, and are not paginated. These online formatted reference pages do not print correctly on hardcopy printers. No support is provided for non-Compaq devices except for generic dumb printers.	To create paginated reference pages, process the source reference pages using the −man.page macro package. See man(1) for instructions on how to format for printing.
catpw	Not supported.	Use the printpw command.
ccat	Not supported.	No Tru64 UNIX equivalent.
ccr	Not supported.	No Tru64 UNIX equivalent.
chpt	Not supported.	Use the disklabel command.
col	The −h option outputs tabs instead of spaces.	Use the −x option to output spaces.
compact	Not supported.	Use the compress command.
cpio	No −k option.	No Tru64 UNIX equivalent.
	The -C option specifies a record size for input and output instead of providing a compatibility mode.	No Tru64 UNIX equivalent.
crash	Not supported.	Use the kdbx command.
crontab	When both day of week and day of month arguments are specified, the command is executed when both match.	The command is executed when either of these specified arguments match.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
csh	No hashstat built-in command.	No Tru64 UNIX equivalent.
	No CSHEDIT environment variable.	Use the editmode variable.
ctrace	Not supported.	Use the dbx debugger.
dms	Not supported.	No Tru64 UNIX equivalent.
drtest	Not supported.	No Tru64 UNIX equivalent.
dupterm	Not supported.	No Tru64 UNIX equivalent.
edauth	Not supported.	Auditing not supported.
elcsd	Not supported.	Use the syslogd daemon.
eli	Not supported.	No Tru64 UNIX equivalent.
enroll	Not supported.	Secret mail not supported.
ex	The ULTRIX ex editor uses the termcap database.	The Tru64 UNIX ex editor uses the terminfo database.
ext_srvtab	Not supported.	Kerberos not supported.
eyacc	Not supported.	Use the yacc command.
flcopy	Not supported.	No Tru64 UNIX equivalent.
format	Not supported.	No Tru64 UNIX equivalent.
from	No -f option.	Use the mailx –f –H *mailbox* command.
fsirand	Not supported.	No Tru64 UNIX equivalent.
gencat	Message catalog limits increased.	No usage difference.
genra	Not supported.	Use the gendisk command.
gcore	Not supported.	No Tru64 UNIX equivalent.
getauth	Not supported.	Auditing not supported.
hesupd	Not supported.	No Tru64 UNIX equivalent.
iconv	Does not accept user-defined conversion tables as input.	Use iconv to convert only between pc850 (IBM personal computer code) and ISO 8859-1 (Latin/1) character sets.
ifconfig	No copyall and -copyall parameters.	No Tru64 UNIX equivalents.
	No dstaddr parameter.	Use the dest_address parameter.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
install	Derived from System V Version 3 install program.	For an installation program that has the BSD install program behavior, use the installbsd command.
ipcs	No -C option.	No Tru64 UNIX equivalent.
	No -N option.	No Tru64 UNIX equivalent.
kdb_destroy	Not supported.	Kerberos not supported.
kdb_edit	Not supported.	Kerberos not supported.
kdb_init	Not supported.	Kerberos not supported.
kdb_util	Not supported.	Kerberos not supported.
kdestroy	Not supported.	Kerberos not supported.
kgconv	Not supported.	Kerberos not supported.
kinit	Not supported.	Kerberos not supported.
klist	Not supported.	Kerberos not supported.
kprop	Not supported.	Kerberos not supported.
kpropd	Not supported.	Kerberos not supported.
kstash	Not supported.	Kerberos not supported.
lb_admin	Not supported.	No Tru64 UNIX equivalent.
lcp	Not supported.	Use the latcp command.
ld	The -l option links with shared libraries by default.	Use the -non_shared option to link with static libraries.
lk	Not supported.	Use the ld command.
llbd	Not supported.	No Tru64 UNIX equivalent.
load	Not supported.	No Tru64 UNIX equivalent.
login	No -C, -e, or -P options.	No Tru64 UNIX equivalents.
	No -r option.	The Tru64 UNIX system automatically initializes the rlogin protocol in the rlogind daemon prior to executing the login utility.
lpr	No -z option.	No Tru64 UNIX equivalent.
	No Ddatatype option.	No Tru64 UNIX equivalent.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
	Translating ASCII, ReGIS, or TEKTRONIX data into PostScript data is not supported. Displaying messages from a PostScript printer is not supported.	Embed PostScript commands in the PostScript file to allow data translation or to display messages from a printer.
lpx	Not supported.	No Tru64 UNIX equivalent.
ls	The −l option displays the group by default.	Use the −o option in place of the ULTRIX −l option. Use the −l option in place of the ULTRIX −lg option combination.
mail	If the /usr/ucb directory was searched before the /bin directory, you used the /usr/ucb/mail program.	Use the mailx command to use a similar program.
	If the /bin directory was searched before the /usr/ucb directory, you used the /usr/bin/mail program.	Use the binmail command to use a similar program.
MAKEHOSTS	Not supported.	No Tru64 UNIX equivalent.
man	Does not reformat a reference page every time standard out is redirected to a pipe or file.	Reformat the reference page manually. See man(1).
	Reference pages are displayed by more −svf instead of page −s.	Use more −svf or page −svf when viewing formatted reference pages directly.
	Does not recognize the sections local, new, old, or public.	Specify sections l, n, o, and p.
	Multicharacter subsection names are no longer hard coded.	
	See the catman command.	
mdtar	Not supported.	Use the tar command.
miscd	Not supported.	Use the inetd daemon in place of miscd.
mkconsole	Not supported.	No Tru64 UNIX equivalent.
mkfs	Not supported.	Use the newfs command.
mktemp	Not supported.	No Tru64 UNIX equivalent.
mop_mom	Not supported.	Supported in the DECnet/OSI for Tru64 UNIX product.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
more	Does not pass Escape sequences by default.	Specify the –v option.
	The default number of lines displayed is *k*–1 instead of *k*–2.	Use the –*n* option to override the default.
	Does not allow hyphens in the MORE environment variable.	Remove all hyphens and spaces from the MORE environment variable.
	See the ex command.	
mountd	By default, the Tru64 UNIX command requires you to be superuser.	Specify the -n option when you are not the superuser.
nawk	Not supported.	Use the gawk command.
netx	Not supported.	No Tru64 UNIX equivalent.
nfssetlock	Not supported.	No Tru64 UNIX equivalent.
nrglbd	Not supported.	No Tru64 UNIX equivalent.
nroff	Instead of the Teletype Model 37 terminal, the default output device for nroff is a generic dumb printer with no reverse line capabilities.	No Tru64 UNIX support for the Teletype Model 37 terminal.
	Converts bold font data to *char*<BS> *same_char* sequences if the device does not have a bold font. This overstriking is invisible except on line printers.	Pipe the output through the ul(1) command if bold text is not visible, and use more –svf to view the result.
	RISC ULTRIX nroff drivers are not compatible.	Convert RISC ULTRIX nroff drivers to C code and recompile them. See term(4).
ntalkd	Not supported.	Use the talkd daemon for remote use of the talk command.
ntpd	Not supported.	Use the xntpd daemon.
opser	Not supported.	No Tru64 UNIX equivalent.
otalk	Not supported.	No Tru64 UNIX equivalent.
page	The default number of lines displayed is *k* instead of *k*–1.	Use the –*n* option to override the default.
passwd	No -a option.	No Tru64 UNIX equivalent.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
pc	Not supported.	No Tru64 UNIX equivalent. However, you can purchase a Pascal compiler separately from the Tru64 UNIX system.
pdx	Not supported.	Use the dbx debugger.
pg	See ex.	
ping	The -l option causes ping to send a specified number of packets, rather than causing ping to display long output.	Use the ping command without the -q option to receive long output.
pixie	The –dwops, –idtrace, –itrace, –dtrace, and –idtrace_file, options are not supported.	No Tru64 UNIX equivalents.
plot	Not supported.	No Tru64 UNIX equivalent.
pmerge	Not supported.	No Tru64 UNIX equivalent.
print	Not supported.	Use the lpr command.
prmail	Not supported.	Use the binmail –p command.
pstat	Not supported.	No Tru64 UNIX equivalent.
pxp	Not supported.	No Tru64 UNIX equivalent.
pxref	Not supported.	No Tru64 UNIX equivalent.
pxtar	Not supported.	Use the tar command.
rc	Not supported.	The rc2 and rc3 commands are run to bring the system to multiuser mode. These commands are invoked by the inittab procedure.
rc.local	Not supported.	The rc2 and rc3 commands are run to bring the system to multiuser mode. These commands are invoked by the inittab procedure.
regis_ps	Not supported.	No Tru64 UNIX equivalent.
remnode	Not supported.	Supported in the DECnet/OSI for Tru64 UNIX product.
rmauth	Not supported.	Auditing not supported.
routed	No Simple Network Management Protocol (SNMP) support.	Use the gated routing daemon.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
rsh5	Not supported.	Use the Rsh shell.
	No -n option.	No Tru64 UNIX equivalent.
	No -d option.	No Tru64 UNIX equivalent.
	No -r option reverses the sort order of the display, rather than displaying only hosts that are running.	
rwho	No -h option.	No Tru64 UNIX equivalent.
rxformat	Not supported.	No Tru64 UNIX equivalent.
rzdisk	Not supported.	Use the scu program.
s5make	Not supported.	Use the make command.
scamp	Not supported.	No Tru64 UNIX equivalent.
secsetup	Not supported.	No Tru64 UNIX equivalent.
setauth	Not supported.	Auditing not supported.
sh	The sh command is like the ULTRIX sh5 command, not the ULTRIX sh command.	No Tru64 UNIX equivalent for ULTRIX sh.
	No -n option for the echo command. The echo command interprets escape sequences, such as \c, \n, or \t.	To suppress the newline character, specify \c at the end of a string argument to the echo command instead of the -n option. To make echo display the characters in the escape sequence, enclose arguments to echo in quotation marks and specify extra backslashes. For example, to cause echo to display \c, enter \\\c as an argument.
	No set - command.	No Tru64 UNIX equivalent.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
sh5	Not supported.	Use the sh shell. Additionally, the UNIX sh command determines whether the argument to the built-in cd command is a subdirectory of any of the directories specified in the CDPATH environment variable. The shell changes your current directory to the first subdirectory that matches the argument.
shexp	Not supported.	No Tru64 UNIX equivalent.
snapcopy	Not supported.	No Tru64 UNIX equivalent.
sort5	Not supported.	Use the sort command.
spline	Not supported.	No Tru64 UNIX equivalent.
startcpu	Not supported.	No Tru64 UNIX equivalent.
stcode	Not supported.	No Tru64 UNIX equivalent.
sticky	Not supported.	No Tru64 UNIX equivalent.
stopcpu	Not supported.	No Tru64 UNIX equivalent.
su	To become superuser, a user must be a member of the system group (GID 0) if GID 0 exists.	Delete the group with GID 0 from the group access list or add user names for all users that should have root access to the group. The group access list is stored in the /etc/group database.
symorder	Not supported.	No Tru64 UNIX equivalent.
syscript	Not supported.	No Tru64 UNIX equivalent.
tapex	No -N option.	No N-buffered I/O support.
tar	The ULTRIX header format for multivolume tapes is unsupported.	Use the −U option.
	No −d, −D, −H, −M, −N, −O, −R, or −V option.	No Tru64 UNIX equivalents.
	The −s option tells tar to strip off leading slashes from pathnames instead of specifying the number of 512-byte blocks.	Use the −S option to specify the number of 512-byte blocks.
tek4014_ps	Not supported.	No Tru64 UNIX equivalent.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
test	The -f option determines whether a file exists and is a regular file.	Use other options to emulate the ULTRIX test command as described in Section 3.2.1.
timed	No -E option.	No Tru64 UNIX equivalent.
trace	Not supported.	Use the dbx debugger.
trigger	Not supported.	No Tru64 UNIX equivalent.
uac	Not supported.	No Tru64 UNIX equivalent.
uerf	No -A option.	No Tru64 UNIX equivalent.
	Not all type codes for the -O and -r options are available.	See uerf(8) for a list of supported type codes.
ul	The ULTRIX ex editor uses the termcap database.	The Tru64 UNIX ex editor uses the terminfo database.
uncompact	Not supported.	Use the compress command to compress files and the uncompress command to expand the files.
uuclean	Not supported.	Use the Tru64 UNIX uucleanup command.
uucompact	Not supported.	No Tru64 UNIX equivalent.
uucp	Significant differences.	For information about using the Tru64 UNIX uucp command, see uucp(1).
uuid_gen	Not supported.	No Tru64 UNIX equivalent.
uulog	No -u*user* option.	Use the Tru64 UNIX uustat -u*user* command.
uumkspool	Not supported.	No Tru64 UNIX equivalent.
uupoll	Not supported.	Use the Tru64 UNIX uutry command.
uurespool	Not supported.	No Tru64 UNIX equivalent.
uusub	Not supported.	No Tru64 UNIX equivalent.
vc	Not supported.	No Tru64 UNIX equivalent.
vcc	Not supported.	Use the cc command.
vi	The ULTRIX ex editor uses the termcap database.	The UNIX ex editor uses the terminfo database.
wait	Not supported.	Use the /bin/sh built-in wait command.

ULTRIX Command	Differences on a Tru64 UNIX System	Use this Instead
`xget`	Not supported.	Secret mail not supported.
`xlator_call`	Not supported.	No Tru64 UNIX equivalent.
`xsend`	Not supported.	Secret mail not supported.
`zic`	Not supported.	No Tru64 UNIX equivalent.

B

Differences in ULTRIX and Tru64 UNIX Header Files and Library Routines

A number of system header files are different between the Tru64 UNIX and ULTRIX systems. In some cases, an ULTRIX header file is unavailable on Tru64 UNIX systems. Some differences between Tru64 UNIX and ULTRIX header files are in the definitions of constants. Some constants that are defined on ULTRIX systems are undefined on Tru64 UNIX systems; other constants have different values on Tru64 UNIX and ULTRIX systems. Other differences are in the definitions of functions. Some ULTRIX functions are not defined on Tru64 UNIX systems; others have different parameters or return values. These differences might affect the binary or source portability of your application.

The header files for the system are so numerous that it is difficult to compile a complete list. The following sections describe known differences in `/usr/include` that may cause problems when porting binary or source code and describes the effects the differences have on program portability.

B.1 Changes in the acct.h File

The `/usr/include/sys/acct.h` header file defines data types and structures for use by programs that perform accounting. The following definitions are different between ULTRIX and Tru64 UNIX systems:

Definition	Type on ULTRIX	Type on Tru64 UNIX
`ac_uid`	`short`	`uid_t` (4 bytes)
`ac_gid`	`short`	`gid_t` (4 bytes)
`ac_tty`	`short`	`dev_t` (4 bytes)

B.2 Changes in the disktab.h File

The `disktab.h` header file defines structures, symbols, and routines that work with disk geometries and disk partition characteristics. On Tru64 UNIX systems, the file omits the following definition:

```
struct disktab  *creatediskbyname();
```

The Tru64 UNIX system does not provide the `creatediskbyname` routine. You must remove references to that routine from your application.

B.3 Changes in the dli_var.h File

The `dli_var.h` header file defines constants and structures used by Data Link Interface (DLI) applications. The file is named `/usr/include/dli/dli_var.h` on Tru64 UNIX systems. In addition, the `sockaddr_dl` structure contains the following new field, beginning in the first byte:

```
u_char dli_len;
```

B.4 Changes in the errno.h File

The `errno.h` header file defines constants that system calls store in the global `errno` variable when an error occurs.

The following definitions are not available and will have an impact on your ability to port source code:

- EACTIVE
- EALIGN
- ENOACTIVE
- ENORESOURCES
- ENOSYSTEM
- ENODUST
- EDUPNOCONN
- EDUMPNODISCONN
- EDUPNOTCNTD
- EDUPNOTIDLE
- EDUPNOTWAIT
- EDUPNOTRUN
- EDUPBADOPCODE
- EDUPINTRANSIT
- EDUPTOOMANYCPUS

Most of these definitions are used with DIGITAL Storage Architecture (DSA) mass storage controllers, such as the CI bus and HSC controller, which are not supported by the Tru64 UNIX system.

See `intro`(2) for a list of Tru64 UNIX `errno` definitions.

B.5 Changes in the fcntl.h File

The /usr/include/fcntl.h file on ULTRIX systems includes the /usr/include/sys/file.h file. On Tru64 UNIX systems, the included file is named /usr/include/sys/fcntl.h, and it contains a different set of definitions. If your application needs the definitions in /usr/include/sys/file.h, you must include that file explicitly.

B.6 Changes in the fstab.h File

The fstab.h header file defines information about the known file system. On Tru64 UNIX systems, the file omits the following definition:

```
struct fstab  *getfstype();
```

In addition, the last two members of the fstab structure have been renamed from fsname to fs_vfstype and from fs_opts to fs_mntops.

B.7 Changes in the in.h File

The /usr/include/netinet/in.h file defines constants and structures defined by the internet system. On Tru64 UNIX systems, the file has changed the definition of the in_addr structure.

On ULTRIX systems:

```
    struct in_addr {
            union {
                    struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
                    struct { u_short s_w1,s_w2; } S_un_w;
                    u_long S_addr;
            } S_un;
    };
```

On Tru64 UNIX systems:

```
    struct in_addr {
            u_int s_addr;
    };
```

B.8 Changes in the ioctl.h and ioctl_compat.h Files

The ioctl.h and the ioctl_compat.h header files define requests and structures that you use with the ioctl system call.

The following definitions are not available and will have an impact on your ability to port source code:

- TIOCCAR
- TIOCNAR

- TIOCWONLINE

Tru64 UNIX systems use POSIX library routines to provide greater application portability. See Appendix D for examples of ULTRIX and Tru64 UNIX modem control applications.

B.9 Changes in the langinfo.h File

The `langinfo.h` header file defines constants that you use to get internationalization information with the `nl_langinfo` routine. Two of the constants that you could use as arguments to the ULTRIX `nl_langinfo` routine are not defined in the Tru64 UNIX `langinfo.h` file. The EXPL_STR constant, which on the ULTRIX system returns a lowercase letter that you can use for an exponent character, is not defined on Tru64 UNIX systems. The EXPU_STR constant, which on ULTRIX returns an uppercase character that you can use for an exponent character, is also not defined on Tru64 UNIX systems.

The `ic` compiler ignores these constants if you use them.

B.10 Changes in the limits.h File

On Tru64 UNIX systems, the `limits.h` file defines certain system limits, such as the maximum number of bytes that you can use to specify a pathname or the maximum message set number that you can use in an internationalization message catalog. Some limits have changed between ULTRIX and Tru64 UNIX systems. Table B–1 describes the differences.

Table B–1: Differences in System Limits

Macro Name	Description	ULTRIX Limit	Tru64 UNIX Limit
LONG_BIT	Maximum number of bits in a type	32	64
LONG_MAX	Maximum value of a `long` type	2,147,483,647	9,223,372,036,854,775,807
LONG_MIN	Minimum value of a `long` type	–2,147,483,648	–9,223,372,036,854,775,808
MB_LEN_MAX	Maximum number of bytes in a multibyte character	1	2
NL_LANGMAX	Maximum length, in bytes, of a string that can be stored in the LANG environment variable	32	14
NL_LBLMAX	Maximum number of labels that can be specified in an internationalization message catalog	32,767	Undefined

Table B–1: Differences in System Limits (cont.)

Macro Name	Description	ULTRIX Limit	Tru64 UNIX Limit
NL_MSGMAX	Maximum number that can be assigned to a message in an internationalization message catalog	32,767	65,535
NL_NMAX	Maximum n-to-1 bytes in mapping character	2	10
NL_SETMAX	Maximum message set number that can be used in an internationalization message catalog	255	65,535
NL_TEXTMAX	Maximum number of bytes that can be in a single program message specified in an internationalization message catalog	2048	4096
ULONG_MAX	Maximum value of an unsigned long type	4,294,967,295	18,446,744,073,709,551,615

B.11 Changes in the math.h File

The `math.h` header file declares the functions in the math library, as well as various functions in the C library that return floating-point values. The Tru64 UNIX `math.h` file omits the declaration of the `atof` routine. This routine is declared in the `stdlib.h` file on the Tru64 UNIX system.

If you use the `atof` routine on a Tru64 UNIX system, be sure your source file includes the `stdlib.h` file.

B.12 Changes in the resource.h File

The `resource.h` file defines a structure named `rusage`. This structure has fewer fields on a Tru64 UNIX system than it does on an ULTRIX system. The definition on the ULTRIX system contains a field for the integral shared text size. The Tru64 UNIX definition omits this field.

You must modify your application if it depends upon the ULTRIX definition of the `rusage` structure.

On Tru64 UNIX systems, the two members of the `rlimit` structure, `rlim_cur` and `rlim_max` are defined as unsigned `long` instead of as `int` on ULTRIX systems. You must modify your application if it depends on this structure. Otherwise, the `getrlimit` and `setrlimit` calls will fail because of a register sign extension.

B.13 Changes in the stddef.h File

On ULTRIX systems, the `wchar_t` variable that is defined in `stddef.h` is declared to be an unsigned integer (32 bits). On Tru64 UNIX systems, the variable is declared to be an unsigned short integer (16 bits).

On ULTRIX systems, the `size_t` variable that is defined in `stddef.h` is declared to be an unsigned integer (32 bits). On Tru64 UNIX systems, the variable is declared to be an unsigned long integer (64 bits).

B.14 Changes in the stdio.h File

The `stdlib.h` header file defines constants and functions I/O services. The following constant values have been changed in the Tru64 UNIX `stdio.h` file:

Constant	ULTRIX Value	Tru64 UNIX Value
`BUFSIZ`	1024	8192
`FILENAME_MAX`	1024	255
`TMP_MAX`	17,576	16,384

B.15 Changes in the stdlib.h File

The `stdlib.h` header file defines constants and functions for ANSI compatibility. Two constants are defined to a different value on Tru64 UNIX and ULTRIX systems. On Tru64 UNIX systems, the RAND_MAX value is defined as 2,147483,647. On ULTRIX systems, this constant is defined to be 32,767. On Tru64 UNIX systems, MB_CUR_MAX is defined as the function `__getmbcurmax()`. This function returns the maximum number of bytes allowed in a multibyte character in the current locale. That number is 1 for all the Tru64 UNIX locales. On ULTRIX systems, MB_CUR_MAX is defined as 4.

B.16 Changes in the syslog.h File

The `syslog.h` header file defines constants that are used in the system log. This header file also defines the routines that control the system log. The definition for the `openlog` routine is different on Tru64 UNIX systems. On a Tru64 UNIX system, the definition is as follows:

```
int openlog (const char *, int, int);
```

This definition adds an extra parameter to the `openlog` call. For information about using the Tru64 UNIX `openlog` call, see `openlog`(3).

B.17 Changes in the termio.h and termios.h Files

The `termio.h` and `termios.h` header files define structures and flags used to control terminals. The definition of the structures `termio` and `termios` differs between Tru64 UNIX and ULTRIX systems. For `termios`, the Tru64 UNIX system defines additional members to control input and output speeds and does not contain the ULTRIX `c_line` member for defining the line discipline. ULTRIX systems define line speed in the low-order 4 bits of the `c_cflag` member.

A number of flags that have been defined for both `termios.h` and `termio.h` are common between ULTRIX and Tru64 UNIX systems; however, their actual definitions can be different across systems. Additionally, several of the definitions in `/usr/include/termio.h` on the ULTRIX system are located in the Tru64 UNIX `/usr/include/termios.h` file. This change should be transparent because the ULTRIX `termios.h` file includes `termio.h` for these definitions.

ULTRIX and Tru64 UNIX systems have different implementations of some of the processing options that are extensions to the POSIX and X/Open standards, as shown in the following table:

termios Member	ULTRIX Name	Tru64 UNIX Name
Special control characters defined by the array `c_cc`	None	VSTATUS
Bit fields defined by `c_iflag` for basic terminal input control	None	IMAXBEL
	PPENDIN	None
	TCBREAK	None
Bit fields defined by `c_oflag` for system treatment of output	None	OXTABS
	None	ONOEOT
	PTILDE	None
	PFLUSHO	None
	PLITOUT	None
	PNL2	None
Bit fields defined by `c_lflag` for control of various terminal functions	None	ECHOKE
	None	ECHOPRT
	None	ALTWERASE
	None	MDMBUF
	None	FLUSHO
	None	NOHANG
	None	PENDIN
	None	NOKERNINFO
	PRAW	None
	PPRTERA	None
	PCRTBS	None
	PCRTERA	None
	PCRTKIL	None

B.18 Nonexistent Header Files

Several header files that are part of the ULTRIX system are not included in the Tru64 UNIX system. Table B–2 lists these files and describes the effects of removing references to them from your source code.

Table B–2: ULTRIX Header Files Not Present on Tru64 UNIX Systems

Header File	Description
ansi_compat.h	Defines ANSI-style predefined macros. On Tru64 UNIX systems, these definitions are provided either by the C preprocessor or the standards.h file. Removing references to this file has no effect.
auth.h	Defines symbols for the authorization library routines, which are unavailable on Tru64 UNIX systems. You must also remove references to the getauthuid, endauthent, storeauthent, and setauthfile routines.
cat.h	Contains no definitions on an ULTRIX system. Removing the #include directive for this file has no effect.
cursesX.h	Defines symbols used by the curses terminal-handling routines. Replaced by the curses.h file; change references accordingly.
des.h krb.h	These files define symbols used by the Kerberos library routines, which are unavailable on Tru64 UNIX systems. You must remove references to these files and to any Kerberos routines.
dial.h	Contains definitions used by the dial() and undial() routines.
elcsd.h elwindow.h	These files define symbols used by the ULTRIX error logger routines. They are not used by user applications.
execargs.h	Contains no definitions on an ULTRIX system. Removing the #include directive for this file has no effect.
fpi.h	Contains definitions used by the floating-point mathematical routines.
hesiod.h	Defines symbols for the Hesiod name service, which is unavailable on Tru64 UNIX systems. You must remove references to this file and to the hes_init, hes_to_bind, hes_error, and hes_resolve routines.
ieeefp.h	Contains definitions for handling Not_a_Numbers (NaN). For standards conformance, these definitions are now in the /usr/include/math.h and /usr/include/nan.hm files.

Table B–2: ULTRIX Header Files Not Present on Tru64 UNIX Systems (cont.)

Header File	Description
i_errno.h	Defines internationalization error numbers. Not typically used by user applications. If your application uses these definitions, create your own file of definitions and include that file.
nlm_prot.h	Contains definitions used by the ONC lock manager daemon. This header file is not needed by user applications.
prof.h	Contains no definitions on an ULTRIX system. Removing the #include directive for this file has no effect.
resscan.h	Defines symbols used by the ULTRIX Hesiod routines. This file is not used by user applications.
stand.h	Contains definitions for the ULTRIX standalone system. This header file is not needed on Tru64 UNIX systems.
sysmips.h	Defines MIPS specific system calls. All system calls are defined in the /usr/include/syscall.h file.
ttyio.h	Contains terminal (tty) common structures and definitions. Use the /usr/include/ioctl.h file.

C

Differences Between Tru64 UNIX and ULTRIX System Calls

This appendix describes the differences between Tru64 UNIX system calls and ULTRIX system calls.

To use the table in this appendix, look for the name of an ULTRIX system call in the left-hand column of the table. Read the second column of the table to determine what difference exists between the Tru64 UNIX and ULTRIX system call. Read the right-hand column to determine how to get the effect of the ULTRIX system call on a Tru64 UNIX system.

ULTRIX System Call	Difference from ULTRIX Systems	Using on Tru64 UNIX Systems
`getrusage`	The structure returned in the `rusage` parameter is different on Tru64 UNIX systems from the structure returned on ULTRIX systems. The ULTRIX structure contains a field (`ru_ismrss`) that the Tru64 UNIX structure does not contain.	Modify your application to use the Tru64 UNIX structure, rather than the ULTRIX structure. (The structure is defined in the `resource.h` file.)
`open`	The O_FSYNC flag is not supported.	Use the O_SYNC flag.
`shmctl`	The SHM_LOCK and SHM_UNLOCK commands are not supported.	Use the F_GETLCK and F_SETLCK requests to the `fcntl` call. See `fcntl`(2) for more information about these requests.
`setsockopt`	The *optval* and *optlen* parameters are not optional; that is, they cannot be specified as zero (0).	Pass the appropriate parameters for *optval* and *optlen*.
`startcpu`	Not on Tru64 UNIX systems.	No equivalent use.
`stopcpu`	Not on Tru64 UNIX systems.	No equivalent use.
`vfork`	Makes a copy of the parent's address space when the child process attempts to write to it.	Investigate using DECthreads software. (See the *Guide to DECthreads*.)

D

Differences Between Tru64 UNIX and ULTRIX Terminal Modem Control

This appendix contains three sample programs showing terminal (tty) modem control: an ULTRIX program showing an outgoing phone call, a Tru64 UNIX program showing an outgoing phone call, and a Tru64 UNIX program showing an incoming phone call. The ULTRIX system uses TIOCCAR, TIOCNAR, and TIOCWONLINE requests to the `ioctl()` system call. These requests are not supported on a Tru64 UNIX system. See Section 7.8 for more information.

Example D–1 demonstrates how an ULTRIX application interacts with a modem for outgoing calls. Error checking of the return values of the system calls is purposely omitted to simplify the example.

Example D–1: Modem Control for Outgoing Calls (ULTRIX)

```
fd = open(dcname, O_RDWR|O_NDELAY);        [1]
ioctl(fd, TIOCMODEM, &temp);               [2]
ioctl(fd, TIOCNCAR);                       [3]
⋮

/*
 * Dial the phone number and negotiate with auto calling unit.
 */
⋮

ioctl(fd, TIOCCAR);                        [4]
alarm(40);
ioctl(fd, TIOCWONLINE);                    [5]
alarm(0);
```

[1] Opens the line and does not wait for carrier.

[2] Monitors the modem signals.

[3] Allows read and write calls to succeed regardless of whether carrier is present.

[4] Allows read and write calls to succeed only if carrier is present.

[5] Waits for carrier.

Example D–2 demonstrates how a Tru64 UNIX application interacts with a modem for outgoing calls. Error checking of the return values of the system calls is purposely omitted to simplify the example.

Example D–2: Modem Control for Outgoing Calls (Tru64 UNIX)

```
int fd, flags;
struct termios tty_termios;           [1]
fd = open(ttyname,O_RDWR | O_NONBLOCK);  [2]
tcgetattr(fd,&tty_termios);              [3]
if ((tty_termios.c_cflag & CLOCAL) == 0) {
 tty_termios.c_cflag |= CLOCAL;        [4]
 tcsetattr(fd,TCSANOW,&tty_termios);
}
flags = fcntl(fd, F_GETFL)
       fcntl(fd, F_SETFL, flags & ~O_NONBLOCK)   [5]
   .
   .
   .
/*
 * dial phone number and negotiate with modem.
 */
   .
   .
   .
tty_termios.c_cflag &= ~CLOCAL;      [6]
tcsetattr(fd,&tty_termios);   [7]
alarm(40);   [8]
read(fd,buffer,count);  [9]
alarm(0);   [10]
```

[1] Contains information about the serial line that can be inspected and altered using the POSIX `tcgetattr()` and `tcsetattr()` library routines.

[2] Opens the terminal line. The CLOCAL flag is usually set by default, allowing you to ignore modem status lines. Use O_NONBLOCK in case CLOCAL is not set.

[3] Gets current line attributes.

[4] Sets CLOCAL, if it is not set. The line must be in local mode in order to talk to the modem.

[5] Turns off O_NONBLOCK; in local mode the application does not need it.

[6] Puts the line into modem mode by turning off CLOCAL. The next I/O operation to the line will block until carrier is present.

[7] Watches for modem signals.

8 Sets a timer so the application does not wait forever.

9 This read() call blocks, pending the appearance of modem signals.

10 Turns off timer; the connection with the remote system has been established.

Example D–3 demonstrates how a Tru64 UNIX application interacts with a modem for incoming calls. Error checking of the return values of the system calls is purposely omitted to simplify the example.

Example D–3: Modem Control for Incoming Calls (Tru64 UNIX)

```
int fd;
int lined;
struct termios tty_termios;         1
fd = open(ttyname,O_RDWR | O_NONBLOCK);  2
flags = fcntl(fd, F_GETFL)
       fcntl(fd, F_SETFL, flags & ~O_NONBLOCK)  3
setsid();   4
ioctl(0, TIOCSCTTY, 0);    5
lined = 0;
ioctl(0, TIOCSETD, &lined); 6
tcgetattr(fd,&tty_termios); 7
tty_termios.c_cflag &= ~CLOCAL;  8
tty_termios.c_cflag &= ~CBAUD;
tty_termios.c_cflag |= B2400; 9
tcsetattr(fd,TCSANOW,&tty_termios);  10
for (;;) {
 write(fd,"\r\nlogin: ",9); 11
 read(fd,buffer,count);  12
 if (valid_login_name(buffer))
  execl("/usr/bin/login",buffer);  13
 }
```

1 Contains information about the serial line that can be inspected and altered using the POSIX tcgetattr() and tcsetattr() library routines.

2 Opens the terminal line. The CLOCAL flag is usually set by default, allowing you to ignore modem status lines. Use O_NONBLOCK in case CLOCAL is not set.

3 Turns off O_NONBLOCK; you do not need it.

4 Creates a new session, becomes session leader for new session, and becomes process leader for new process group.

5 Sets the controlling terminal.

6 Sets the line discipline to 0 (POSIX).

7 Gets current line attributes.

8 Turns off CLOCAL for a modem.

9 Clears baud rate bits and sets them to 2400.

10 Sets the line attributes.

11 This write call, containing the login message, blocks until someone dials in on the modem and all the signals are present.

12 Gets the user’s login name.

13 Executes the login program, if the login name is valid.

E

Summary of XUI and Motif Differences

This appendix summarizes the differences between the XUI and Motif interfaces in the following areas:

- Terminology
- Windows and window managers
- Menus and menu items
- Mouse button bindings
- Standard message boxes
- Keyboard behavior

Appendix F contains a list of widget naming differences.

E.1 Terminology

Table E–1 lists terminology differences between the XUI and Motif interfaces.

Table E–1: Terminology Differences Between XUI and Motif Interfaces

XUI Interface	Motif Interface
Dialog box, modal	Dialog box, primary application modal, application modal, or system modal
Direct manipulation interface	Graphical user interface
End box (in a dialog box)	Command line (in a dialog box)
Exit (menu item)	Exit (menu item). Unlike XUI Exit, you are prompted for whether you want to save the file
Ghost	No equivalent
Hierarchical dialog boxes	Secondary windows XUI does not talk about secondary windows, and OSF does not talk about hierarchical dialog boxes
Icons	Icons or minimized windows
Maximum sliders, minimum sliders	Sliders (no distinction)

Table E–1: Terminology Differences Between XUI and Motif Interfaces (cont.)

XUI Interface	Motif Interface
No equivalent	Maximize
No equivalent	Stepper buttons
No equivalent	Sash (window sash)
Option box	Option menu
Pointer speed	Gain
Push to back	Lower
Quit (menu item)	Exit (menu item). You are prompted for whether you want to save the file
Radio icons	No equivalent
Scales, scroll bars, sliders	Valuators (includes scales, scroll bars, and sliders)
Shrink to icon	Minimize
Stepping arrows	Stepper arrows
Submenu	Cascading menu
Terminal Screen	Workspace
Text Entry Field	Entry box
Text insertion character	Insertion cursor
Toggle button	Check button
Work area	Client area

E.2 Windows and Window Managers

Table E–2 lists differences between XUI and Motif windows and window managers.

Table E–2: Differences Between XUI and Motif Windows and Window Managers

XUI Interface	Motif Interface
Does not have a root menu.	Has a root menu (a menu that pops up in the root window when you press the Select button in a blank area of the root window).
Shrink-to-icon button is in upper left.	Shrink-to-icon (minimize) button is the left-hand button of the two buttons in the upper right.
Does not have a window menu.	Has a window menu (a menu that pops up in the window when you press the Menu button).
Has a resize button in the far right.	Has a resize border and resize handles.
The default window manager has an icon box.	The window manager has an icon box, but by default does not display an icon box.
The text label in the title bar is left-justified.	The text label in the title bar is centered.
Only has explicit focus policy.	Has both explicit (pointer) focus and implicit focus.
Has a push-to-back button.	Has a lower menu item.

E.3 Menus and Menu Items

Table E–3 lists the differences between the XUI and Motif window menu items.

Table E–3: Motif Window Menu Items and Functions

Function	XUI Action or Object	Motif Menu Item
Return a window to original size after it has been shrunk to an icon or enlarged.	Resize button	Restore
Change the location of a window.	Press and drag on the title bar	Move
Change the size of a window.	Resize button	Size
Shrink a window to an icon.	Shrink-to-icon button	Miminize
Enlarge a window to cover the whole screen.	Resize button	Maximize

Table E–3: Motif Window Menu Items and Functions (cont.)

Function	XUI Action or Object	Motif Menu Item
Send a window to the back or bottom of the window stack.	Push-to-back button	Lower
Close a window and remove it from the workspace.	Not in XUI	Close

E.3.1 Menu Bar and Standard Menus

Table E–4 lists the standard menus in each menu bar, and describes the differences between the XUI and Motif interfaces. The XUI interface uses dotted lines as separators; the Motif interface uses solid lines.

Table E–4: Differences Between the XUI and Motif Menus in the Menu Bar

XUI Menu	Motif Menu	Explanation
File	File	Mainly the same menu items.
Edit	Edit	Mainly the same menu items.
	View	Some XUI applications have a View menu.
Customize	Options	Motif provides no specific menu items; the menu items are application specific.
Font		No equivalent in Motif.
Help	Help	Menu items have different names, some similar functions.

E.3.2 File Menu Items

Table E–5 lists the File menu items and describes the differences between the XUI and Motif interfaces.

Table E–5: Differences Between the XUI and Motif File Menu Items

Menu Item	XUI Interface	Motif Interface
New	Creates an empty copy of window; does not affect the previous window.	Clears the existing window; does not provide a new window. To continue to have the XUI "new" capability, include a check button in your File Selection box labeled "Open in New Window."
Open . . .	Generates a dialog box that allows users to open an existing file.	Same in Motif.
Include	Generates a dialog box that allows users to add the contents of a specified file.	Not standard in Motif, but use it if appropriate.
Revert	Generates a dialog box that allows users to erase current work and revert to last saved version of file.	Not standard in Motif, but use it if appropriate.
Print	Prints the current file using the current settings of a Print dialog box without displaying the box.	Exists in Motif; in Motif, Print covers Print... as well.
Print . . .	Generates a Print dialog box that allows users to set printing parameters and print the current file.	Not standard in Motif, but Motif Print menu item pops up a dialog box if printing information is required.
Quit	Shuts down application; prompts for saving if current version has not been saved.	Does not exist in Motif.

Table E–5: Differences Between the XUI and Motif File Menu Items (cont.)

Menu Item	XUI Interface	Motif Interface
Close	Closes the window, leaving the other windows in the application.	Removes the primary and associated secondary windows from the workspace, in applications that have more than one primary window. Closing the last primary window of an application causes the application to exit. If data will be lost, the application must prompt users to save changes. The Close menu item from the File menu should have the same effect as the Close menu item from the Window menu.
Exit	Saves file and shuts down application.	Shuts down application, and prompts for saving if the current version has not been saved.

E.3.3 Edit Menu Items

Table E–6 lists the Edit menu items and describes the differences between the XUI and Motif interfaces.

Table E–6: Differences Between XUI and Motif Edit Menu Items

Menu Item	XUI Interface	Motif Interface
Undo	Reverses the effects of a previous operation.	Same in Motif.
Redo	Redoes an operation after it has been undone.	Not in Motif, but you can add it if appropriate.
Cut	Transfers currently selected information to the clipboard and deletes the information from the application.	Same in Motif.
Copy	Transfers the current selection to the clipboard without altering the information in the application.	Same in Motif.
Paste	Copies information from the clipboard into the application and retains that information in the clipboard.	Same in Motif.
Clear	Deletes the current selection.	Same in Motif.

Table E–6: Differences Between XUI and Motif Edit Menu Items (cont.)

Menu Item	XUI Interface	Motif Interface
Delete	Not in XUI.	Removes selected portion of data from application and compresses the rest of the data to fill the space that the deleted data occupied.
Select All	Selects all the data in the file.	Not in Motif, but you can add it if appropriate.

E.3.4 Help Menu Items

The XUI and Motif Help Menu Items are shown in Table E–7.

Table E–7: Differences Between the XUI and Motif Help Menu Items

Menu Item	XUI Interface	Motif Interface
Overview	Provides general information about the window from which help was requested.	On Window.
About	Provides the name and version of the application.	On Version.
Glossary	Provides definitions of terms.	On Terms.
On Context	Does not exist as a menu item in XUI. In XUI, users press the Help key and any mouse button.	Initiates context-sensitive help.
On Help	Does not exist in XUI.	Provides information on how to use your application's Help facility.
On Keys	Does not exist in XUI.	Provides information about your application's use of function keys, mnemonics, and accelerators.
Index	Does not exist in XUI.	Provides an index for all Help information in your application.
Tutorial	Does not exist in XUI.	Provides access to your application's tutorial.

E.4 Mouse Button Behavior

Table E–8 provides a list of differences between the XUI and Motif mouse button behavior.

Table E–8: Differences in the XUI and Motif Mouse Buttons

Mouse Button	XUI Interface	Motif Interface
MB1	Used for selection.	Used for selection. Called the Select button.
MB2	Used to display pop-up menus.	Used for direct manipulation of objects and other application-specific needs. Called the Menu button.
MB3	Used for application-specific needs, and for Copy To and Copy From operations, if your application supports them.	Used to display pop-up menus. Called the Custom button.

E.5 Standard Message Boxes

Motif message boxes often have a Help push button in the lower right corner.

E.6 Keyboard Behavior

Table E–9 provides a list of differences between the XUI and Motif keyboard behavior. These changes apply to both `mwm` and `twm`.

Table E–9: Differences in the XUI and Motif Keyboard Mappings

Key	XUI Interface	Motif Interface
Compose	Used as `Meta` key.	Used for initiating compose sequences.
Alt Function	Does not exist on an ULTRIX system.	Used as `Meta` key.

Use either the `xmodmap` or `dxkeycaps` program to customize keyboard mappings.

F

DECwindows Motif Component Names

This appendix summarizes name changes for the following DECwindows Motif components:

- Widget classes
- Function names
- Resource names
- Enumeration literal names
- Callback reason names
- Compound string names
- Fontlist names
- Clipboard names
- Resource manager names

For complete descriptions of the widget classes, see the *OSF/Motif Programmer's Reference.*

F.1 Widget Classes

Table F–1 summarizes the differences between the XUI widget hierarchy and the OSF/Motif widget hierarchy.

Table F–1: Widget Class Name Changes

XUI Interface	Motif Interface
DwtAttachedDB	XmForm
DwtCommandWindow	XmCommand
DwtCommon	No equivalent in Motif. The resources are in XmPrimitive, XmManager, and XmGadget.
DwtDialogBox	XmBulletinBoard
DwtFileSelection	XmFileSelectionBox
DwtHelp	DXmhelp
Dwtlabel	XmLabel

Table F–1: Widget Class Name Changes (cont.)

XUI Interface	Motif Interface
DwtListBox	XmList
DwtMainWindow	XmMainWindow
DwtMenu	XmRowColumn
DwtMessageBox	XmMessageBox
DwtPullDownMenuEntry	XmCascadeButton
DwtPushButton	XmPushButton
DwtScale	XmScale
DwtScrollBar	XmScrollBar
DwtScrollWindow	XmScrolledWindow
DwtSelection	XmSelectionBox
DwtSeparator	XmSeparator
DwtSText	XmText
DwtToggleButton	XmToggleButton
DwtWindow	XmDrawingArea

F.2 Function Names

Table F–2 summarizes the differences between the XUI function names and the OSF/Motif function names.

Table F–2: Function Name Changes

XUI Interface	Motif Interface
Dwt*Create	XmCreate*[a]
DwtAttachedDBCreate	XmCreateForm
DwtAttachedDBPopupCreate	XmCreateFormDialog
DwtCautionBoxCreate	XmCreateWarningDialog,
	XmCreateMessageDialog,
	XmCreateErrorDialog,
	or XmCreateQuestionDialog
DwtCommandAppend	XmCommandAppendValue
DwtCommandErrorMessage	XmCommandError
DwtCommandSet	XmCommandSetValue

Table F–2: Function Name Changes (cont.)

XUI Interface	Motif Interface
DwtCommandWindowCreate	XmCreateCommand
DwtDialogBoxCreate	XmCreateBulletinBoard
DwtDialogBoxPopupCreate	XmCreateBulletinBoardDialog
DwtFileSelectionCreate	XmCreateFileSelectionDialog
DwtLabelCreate	XmCreateLabel
DwtLabelGadgetCreate	XmCreateLabelGadget
DwtListBoxCreate	XmCreateList
DwtMainWindowCreate	XmCreateMainWindow
DwtMenuBarCreate	XmCreateMenuBar
DwtMenuCreate	XmCreateRowColumn
DwtMenuPopupCreate	XmCreatePopupMenu
DwtMenuPulldownCreate	XmCreatePulldownMenu
DwtMessageBoxCreate	XmCreateInformationDialog[b]
DwtOptionMenuCreate	XmCreateOptionMenu
DwtPullDownMenuEntryCreate	XmCreateCascadeButton
DwtPullDownMenuEntryHilite	XmCascadeButtonHighlight
DwtPullEntryGadgetCreate	XmCreateCascadeButtonGadget
DwtPushButtonCreate	XmCreatePushButton
DwtPushButtonGadgetCreate	XmCreatePushButtonGadget
DwtRadioBoxCreate	XmCreateRadioBox
DwtScaleCreate	XmCreateScale
DwtScaleGetSlider	XmScaleGetValue
DwtScaleSetSlider	XmScaleSetValue
DwtScrollBarCreate	XmCreateScrollBar
DwtScrollBarGetSlider	XmScrollBarGetValues
DwtScrollBarSetSlider	XmScrollBarSetValues
DwtScrollWindowCreate	XmCreateScrolledWindow
DwtSelectionCreate	XmCreateSelectionBox
DwtSeparatorCreate	XmCreateSeparator
DwtSeparatorGadgetCreate	XmCreateSeparatorGadget

Table F–2: Function Name Changes (cont.)

XUI Interface	Motif Interface
DwtSTextCreate	XmCreateText
DwtToggleButtonCreate	XmCreateToggleButton
DwtToggleButtonGadgetCreate	XmCreateToggleButtonGadget
DwtWindowCreate	XmCreateDrawingArea
DwtWorkBoxCreate	XmCreateWorkingDialog[b]

[a] Most of the name changes follow this form. The table lists those function name changes that do not follow this form.
[b] Instantiates an XmMessageBox widget inside an XmDialogShell widget. To instantiate only the XmMessageBox widget, use XmCreateMessageBox.

F.3 Resource Names

Table F–3 summarizes the differences between the XUI resource names and the OSF/Motif resource names. Some XUI resource names have multiple Motif resource names. To help you determine which Motif resource name applies to your widget, the widget class is listed in parentheses after the Motif name.

Table F–3: Resource Name Changes

XUI Interface	Motif Interface
DwtN*	XmN*[a]
DwtNactivateCallback	XmNokCallback (XmSelectionBox)
DwtNadb*	XmN*[a]
DwtNapplyLabel	XmNapplyLabelString
DwtNautoShowInsertPoint	XmNautoShowCursorPosition
DwtNbuttonAccelerator	XmNaccelerator
DwtNcancelLabel	XmNcancelLabelString
DwtNchildOverlap	XmNallowOverlap
DwtNcols	XmNcolumns
DwtNconformToText	XmNrecomputeSize
DwtNdefaultHorizontalOffset	XmNhorizontalSpacing
DwtNdefaultPushbutton	XmNdefaultButtonType
DwtNdefaultVerticalOffset	XmNverticalSpacing
DwtNdirectionRtoL	XmNprocessingDirection (XmScale, XmScrollBar)

Table F–3: Resource Name Changes (cont.)

XUI Interface	Motif Interface
DwtNdirectionRtoL	XmNstringDirection (XmLabel, XmBulletinBoard, XmList)
DwtNextendCallback	XmNextendedSelectionCallback
DwtNfilterLabel	XmNfilterLabelString
DwtNfont	XmNfontList (XmLabel, XmList, XmScale, XmText)
DwtNfont	XmN*fontList (XmBulletinBoard)
DwtNhistory	XmNhistoryItems
DwtNhorizontal	XmNscrollBarDisplayPolicy
DwtNhotSpotPixmap	XmNcascadePixmap
DwtNiconPixmap	XmNsymbolPixmap
DwtNinc	XmNincrement
DwtNindicator	XmNindicatorOn
DwtNinsensitivePixmap	XmNlabelInsensitivePixmap
DwtNinsensitivePixmapOff	XmNlabelInsensitivePixmap
DwtNinsensitivePixmapOn	XmNselectInsensitivePixmap
DwtNinsertionPointVisible	XmNcursorPositionVisible
DwtNinsertionPosition	XmNcursorPosition
DwtNitems	XmNlistItems
DwtNitemsCount	XmNitemCount (XmList)
DwtNitemsCount	XmNlistItemCount (XmSelectionBox)
DwtNlabel	XmNlabelString (XmLabel, XmRowColumn)
DwtNlabel	XmNlistLabelString (XmSelectionBox)
DwtNlabel	XmNmessageString (XmMessageBox)
DwtNlabelAlignment	XmNmessageAlignment
DwtNlines	XmNhistoryItemCount
DwtNlostFocusCallback	XmNlosingFocusCallback
DwtNmaxValue	XmNmaximum
DwtNmenuAlignment	XmNisAligned
DwtNmenuEntryClass	XmNentryClass

Table F–3: Resource Name Changes (cont.)

XUI Interface	Motif Interface
DwtNmenuExtendLastRow	XmNadjustLast
DwtNmenuIsHomogeneous	XmNisHomogeneous
DwtNmenuNumColumns	XmNnumColumns
DwtNmenuPacking	XmNpacking
DwtNmenuRadio	XmNradioBehavior
DwtNmenuType	XmNrowColumnType
DwtNmergeTextTranslations	XmNtextTranslations
DwtNminValue	XmNminimum
DwtNokLabel	XmNokLabelString
DwtNpageDecCallback	XmNpageDecrementCallback
DwtNpageInc	XmNpageIncrement
DwtNpageIncCallback	XmNpageIncrementCallback
DwtNpixmap	XmNlabelPixmap
DwtNpixmapOff	XmNlabelPixmap
DwtNpixmapOn	XmNselectPixmap
DwtNprompt	XmNpromptString
DwtNpullingCallback	XmNcascadingCallback
DwtNresize	XmNlistSizePolicy (XmList)
DwtNresize	XmNresizePolicy (XmBulletinBoard)
DwtNselectedItemsCount	XmNselectedItemCount
DwtNselectionLabel	XmNselectionLabelString
DwtNshadow	XmNshadowThickness
DwtNshape	XmNindicatorType
DwtNshown	XmNsliderSize
DwtNsingleCallback	XmNsingleSelectionCallback
DwtNsingleConfirmCallback	XmNdefaultActionCallback
DwtNsingleSelection	XmNsingleSelectionPolicy
DwtNspacing	XmNlistSpacing
DwtNstyle	XmNdialogStyle
DwtNtextCols	XmNtextColumns

Table F–3: Resource Name Changes (cont.)

XUI Interface	Motif Interface
DwtNtitle	XmNdialogTitle (XmBulletinBoard)
DwtNtitle	XmNtitleString (XmScale)
DwtNunitDecCallback	XmNdecrementCallback
DwtNunitIncCallback	XmNincrementCallback
DwtNvalue	XmNcommand (XmCommand)
DwtNvalue	XmNset (XmToggleButton)
DwtNvalue	XmNtextString (XmSelectionBox)
DwtNvalueChangedCallback	XmNcommandChangedCallback (XmCommand)
DwtNvisibleItemsCount	XmNvisibleItemCount (XmList)
DwtNvisibleItemsCount	XmNlistVisibleItemCount (Xm-SelectionBox)
DwtNyesCallback	XmNokCallback
DwtNyesLabel	XmNokLabelString

[a] Most of the name changes follow this form. The table lists those resource name changes that do not follow this form.

F.4 Enumeration Literal Names

Table F–4 summarizes the differences between the XUI enumeration literal names and the OSF/Motif enumeration literal names. Some XUI enumeration literal names have multiple Motif enumeration literal names. To help you determine which Motif enumeration literal name applies to your widget, the widget class is listed in parentheses after the Motif name.

Table F–4: Enumeration Literal Name Changes

XUI Interface	Motif Interface
Dwt *AaaaAaaa*	Xm *AAAA_AAAA*[a]
DwtAttachAdb	XmATTACH_FORM
DwtAttachOppAdb	XmATTACH_OPPOSITE_FORM
DwtAttachOppWidget	XmATTACH_OPPOSITE_WIDGET
DwtCancelButton	XmDIALOG_CANCEL_BUTTON
DwtCString	XmSTRING
DwtMenuPackingColumn	XmPACK_COLUMN
DwtMenuPackingNone	XmPACK_NONE

Table F–4: Enumeration Literal Name Changes (cont.)

XUI Interface	Motif Interface
DwtMenuPackingTight	XmPACK_TIGHT
DwtMenuWorkArea	XmWORK_AREA
DwtModal	XmDIALOG_APPLICATION_MODAL
DwtModal	XmDIALOG_FULL_APPLICA-TION_MODAL
DwtModal	XmDIALOG_SYSTEM_MODAL
DwtModeless	XmDIALOG_MODELESS
DwtOrientationHorizontal	XmHORIZONTAL
DwtOrientationVertical	XmVERTICAL
DwtOval	XmONE_OF_MANY
DwtRectangular	XmN_OR_MANY
DwtResizeFixed	XmRESIZE_NONE (XmBulletinBoard)
DwtResizeFixed	XmCONSTANT (XmList)
DwtResizeGrowOnly	XmRESIZE_GROW (XmBulletinBoard)
DwtResizeGrowOnly	XmVARIABLE (XmList)
DwtResizeShrinkWrap	XmRESIZE_ANY (XmBulletinBoard)
DwtResizeShrinkWrap	XmVARIABLE (XmList)
DwtWorkArea	XmDIALOG_WORK_AREA
DwtYesButton	XmDIALOG_OK_BUTTON

[a] Most of the name changes follow this form. The table lists those enumeration literal name changes that do not follow this form.

F.5 Callback Reason Names

Table F–5 summarizes the differences between the XUI callback reason names and the OSF/Motif callback reason names. Some XUI callback reason names have multiple Motif callback reason names. To help you determine which Motif callback reason name applies to your widget, the widget class is listed in parentheses after the Motif name.

Table F–5: Callback Reason Names

XUI Interface	Motif Interface
DwtCR *AaaaAaaa*	XmCR_ *AAAA_AAAA*[a]
DwtCRActivate	XmCR_OK (XmSelectionBox)
DwtCRActivate	XmCR_CASCADING (XmCascadeButton)
DwtCRExtend	XmCR_EXTENDED_SELECTION
DwtCRHelpRequested	XmCR_HELP
DwtCRLostFocus	XmCR_LOSING_FOCUS
DwtCRPageDec	XmCR_PAGE_DECREMENT
DwtCRPageInc	XmCR_PAGE_INCREMENT
DwtCRSingle	XmCR_SINGLE_SELECT
DwtCRSingleConfirm	XmCR_DEFAULT_ACTION
DwtCRUnitDec	XmCR_DECREMENT
DwtCRUnitInc	XmCR_INCREMENT
DwtCRValueChanged	XmCR_COMMAND_CHANGED (XmCommand)
DwtCRYes	XmCR_OK

[a] Most of the name changes follow this form. The table lists those callback reason name changes that do not follow this form.

F.6 Compound Strings

Table F–6 summarizes the differences between the XUI compound string names and the OSF/Motif compound string names.

Although the compound string names are changed, some functions change the order and number of arguments. See the *OSF/Motif Programmer's Reference* (available from Prentice Hall); to verify the arguments.

Table F–6: Compound String Names

XUI Interface	Motif Interface
DwtCompString	XmString
DwtCSbytecmp	XmStringByteCompare
DwtCSempty	XmStringEmpty
DwtCSString	XmStringSegmentCreate[a]
DwtCStrcat	XmStringConcat
DwtCStrcpy	XmStringCopy

Table F–6: Compound String Names (cont.)

XUI Interface	Motif Interface
DwtCStrlen	XmStringLength
DwtCStrncat	XmStringNConcat
DwtCStrncpy	XmStringNCopy
DwtDisplayCSMessage	No equivalent in Motif.
DwtDisplayVMSMessage	No equivalent in Motif.
DwtGetNextSegment	XmStringGetNextSegment
DwtInitGetSegment	XmStringInitContext
DwtLatin1String	XmStringCreateSimple[a]
DwtString	XmStringSegmentCreate[a]

[a] Suggested replacement only.

F.7 Fontlist Names

Table F–7 summarizes the differences between the XUI fontlist names and the OSF/Motif fontlist names.

Table F–7: Fontlist Names

XUI Interface	Motif Interface
DwtAddFontList	XmFontListAdd
DwtCreateFontList	XmFontListCreate

F.8 Clipboard Names

Table F–8 summarizes the differences between the XUI clipboard names and the OSF/Motif clipboard names.

Table F–8: Clipboard Names

XUI Interface	Motif Interface
DwtBeginCopyToClipboard	XmClipboardStartCopy
DwtCancelCopyFormat	XmClipboardWithdrawFormat
DwtCancelCopyToClipboard	XmClipboardCancelCopy
DwtCopyFromClipboard	XmClipboardRetrieve
DwtCopyToClipboard	XmClipboardCopy
DwtEndCopyFromClipboard	XmClipboardEndRetrieve
DwtEndCopyToClipboard	XmClipboardEndCopy

Table F–8: Clipboard Names (cont.)

XUI Interface	Motif Interface
DwtInquireNextPasteCount	XmClipboardInquireCount
DwtInquireNextPasteFormat	XmClipboardInquireFormat
DwtInquireNextPasteLength	XmClipboardInquireLength
DwtListPendingItems	XmClipboardInquirePendingItems
DwtReCopyToClipboard	XmClipboardCopyByName
DwtStartCopyFromClipboard	XmClipboardStartRetrieve
DwtStartCopyToClipboard	XmClipboardStartCopy
DwtUndoCopyToClipboard	XmClipboardUndoCopy

F.9 Resource Manager Names

Table F–9 summarizes the differences between the XUI resource manager names and the OSF/Motif resource manager names.

Table F–9: Resource Manager Names

XUI Interface	Motif Interface
DwtCloseHierarchy	MrmCloseHierarchy
DwtDrmFreeResourceContext	No equivalent in Motif. Use MrmFetchLiteral, MrmFetchIconLiteral, or MrmFetchColorLiteral.
DwtDrmGetResourceContext	No equivalent in Motif. Use MrmFetchLiteral, MrmFetchIconLiteral, or MrmFetchColorLiteral.
DwtDrmHGetIndexedLiteral	No equivalent in Motif. Use MrmFetchLiteral, MrmFetchIconLiteral, or MrmFetchColorLiteral.
DwtDrmRCBuffer	No equivalent in Motif. Use MrmFetchLiteral, MrmFetchIconLiteral, or MrmFetchColorLiteral.
DwtDrmRCSetType	No equivalent in Motif. Use MrmFetchLiteral, MrmFetchIconLiteral, or MrmFetchColorLiteral.
DwtDrmRCSize	No equivalent in Motif. Use MrmFetchLiteral, MrmFetchIconLiteral, or MrmFetchColorLiteral.
DwtDrmRCType	No equivalent in Motif. Use MrmFetchLiteral, MrmFetchIconLiteral, or MrmFetchColorLiteral.

Table F–9: Resource Manager Names (cont.)

XUI Interface	Motif Interface
DwtFetchColorLiteral	MrmFetchColorLiteral
DwtFetchIconLiteral	MrmFetchIconLiteral
DwtFetchInterfaceModule	MrmFetchInterfaceModule
DwtFetchLiteral	MrmFetchLiteral
DwtFetchSetValues	MrmFetchSetValues
DwtFetchWidget	MrmFetchWidget
DwtFetchWidgetOverride	MrmFetchWidgetOverride
DwtInitializeDRM	MrmInitialize
DwtOpenHierarchy	MrmOpenHierarchy
DwtRegisterClass	MrmRegisterClass
DwtRegisterDRMNames	MrmRegisterNames

G

Migration from ULTRIX Version 4.5 to Tru64 UNIX Version 4.0B

This appendix contains brief descriptions of features that are new to the ULTRIX and UWS Version 4.5 operating system, and features that are new to the Tru64 UNIX Version 4.0B operating system. Each description notes any migration issues between ULTRIX Version 4.5 and Tru64 UNIX Version 4.0B. Then, this appendix discusses the interfaces that have been retired in Tru64 UNIX Version 4.0B and whether their retirement affects migration from ULTRIX Version 4.5.

G.1 New Features and Changes in ULTRIX and UWS Version 4.5

The following new features and changes are in ULTRIX and UWS Version 4.5: none has an effect on the migration of the operating system to Tru64 UNIX Version 4.0B.

- The `candc`(8) command is a shell script that examines the core image of the ULTRIX operating system to extract diagnostic data.

  This command has no effect on migration from ULTRIX to Tru64 UNIX.

- An `usrsms` option to the `param.c` file has been added for shared memory management.

  This option has no effect on migration from ULTRIX to Tru64 UNIX.

- A new option, `-l`, has been added to the `ypserv`(8yp) command to turn on log messages.

  This new option has no effect on migration from ULTRIX to Tru64 UNIX.

- The X11R5 `Xws` server now supports both MX and PX graphic options.

  Support for these graphic options has no effect on migration from ULTRIX to Tru64 UNIX.

- A new file, `/etc/securenets`, has been added: it is required for portmapper operations.

  This new file has no effect on migration from ULTRIX to Tru64 UNIX.

- The LinkWorks components have been retired and renamed DEClinks.

The retirement of the LinkWorks components have no affect on migration from ULTRIX to Tru64 UNIX.

G.2 New Features and Changes in Tru64 UNIX Version 4.0B

The remainder of this appendix contains the new and changed features in Tru64 UNIX Version 4.0B. The discussion of each new feature and change concludes with a summary of its affect on the migration of ULTRIX Version 4.5 capabilities to Tru64 UNIX Version 4.0.

G.3 Common Desktop Environment

The Common Desktop Environment (CDE) is the new default graphical user interface for Tru64 UNIX. The CDE environment is designed to provide common services across all UNIX platforms, including a consistent user interface for end users and a consistent development environment for application developers across multiple platforms.

CDE on Tru64 UNIX is based on the X Window System Release 6 (X11R6) and CDE/Motif 1.0 (OSF/Motif 1.2.4), and supplies the following desktop services and applications:

- Desktop Services:

Window Management	Workspace Management	Session Management
File Manager	Application Manager	Windowing `dtksh`
Help	Keyboard Customization	

- Desktop Applications:

Calendar	Calculator	MIME-capable Mail
Text Editor	Icon Editor	Terminal Emulator
Application Builder	Print Queue Manager	

CDE is provided in seven software subsets that require a total of 57.81 MB of free disk space for installation. See the *Installation Guide* for information on the subset names, contents, and sizes.

The CDE kit contains the following migration tools:

- `mailcv` mail conversion. This utility converts your `dxmail` folders to the conventional mail format used by CDE `dtmail`. If you plan to use the `mailcv` utility to convert your existing mail folders, back up the folders before converting them. Do not use the `-d` option with this version of the `mailcv` utility.

- `dxcaltodtcm` calendar conversion. This utility converts a DECwindows Calendar, `dxcalendar`, database for use with CDE Calendar, `dtcm`.

G.3.1 CDE Video Tour

A brief multimedia tutorial of CDE is located on the Tru64 UNIX Version 4.0B Associated Products Volume 1 CD–ROM. Once the video tour is installed, you can access it through the application manager in the Information folder by double clicking on the CDE Video Tour icon.

G.3.2 CDE Screen Savers

The CDE session manager supports X11R6 screen saver extensions and you can now select animated screen savers instead of a blank screen. This release also enables the automatic locking of screens after a specified idle time. You can modifiy or disable both features from the CDE Style Manager menu. Click on the Screen icon, and select the options you want.

G.3.3 ULTRIX Migration Issues

Because ULTRIX V4.5 uses X11R5 and OSF/Motif 1.1.3, there can be migration issues when using the migration tools in the CDE kit. These tools were intended only for migration from earlier versions of Tru64 UNIX to Tru64 UNIX Version 4.0B.

Although the mail and calendar conversion tools were designed for migrating from DECwindows on earlier versions of Tru64 UNIX to Tru64 UNIX Version 4.0, these same tools also can be used for converting ULTRIX DECwindows versions of the applications to Tru64 UNIX Version 4.0B.

DECwindows migration issues are described in the manual *CDE Companion* guide.

G.4 X/Open-Compliant Curses

The new Curses implementation in Tru64 UNIX Version 4.0B incorporates the following sets of programming interfaces:

- X/Open Curses, Issue 4
- System V Multinational Language Supplement (MNLS)
- Minicurses
- BSD Curses

G.4.1 ULTRIX Migration Issues

Because X/Open Compliant Curses, Issue 4, is backward compatible with earlier versions of X/Open Curses, there are no ULTRIX migration issues.

G.5 X11R6

This release of Tru64 UNIX supports Release 6 of the X Window System, Version 11 (X11R6) patchlevel 12. Prior versions of the operating system supported Release 5 (X11R5) patchlevel 26.

The Tru64 UNIX port of X11R6 supports all the features and functionality of previous releases of Tru64 UNIX. It also supports all X Consortium standard features of X11R6.

The following protocol extensions are new features in Tru64 UNIX Version 4.0B:

- BIG-REQUESTS. Gives clients the ability to use requests that are arbitrarily large, rather than being limited to the size restriction of the core protocol. This can result in a significant performance improvement for applications that use large requests.
- DOUBLE-BUFFER. Enables double buffering, using the new X Consortium standard.
- XIE (updated). Complete implementation of full XIE 5.0 protocol with a few exceptions.
- XKEYBOARD (XKB).

G.5.1 X Keyboard Extension for X11R6 (XKB)

The XKB (X Keyboard) server extension is new for X11R6 and for Tru64 UNIX. XKB enhances control and customization of the keyboard under the X Window System by providing the following:

- Support for the ISO9996 standard for keyboard layouts
- Compatibility with the core X keyboard handling (no client modifications are required)
- Standard methods for handling keyboard LEDs and locking modifiers such as CapsLock and NumLock
- Support for keyboard geometry

In addition, the X11R5 AccessX server extension for users with physical impairments has been incorporated into the XKB server extension. X11R5 applied to versions of Tru64 UNIX that preceded this release. These accessibility features include StickyKeys, SlowKeys, BounceKeys, MouseKeys, and ToggleKeys, and control over the autorepeat delay and rate.

Several applications that make use of XKB features are also new with Tru64 UNIX Version 4.0B. These applications include Xdec, xkbcomp, xkbprint, xkbdfltmap, dxkbledpanel, dxkeyboard, and accessx. See the reference pages for more information.

Note that the final revision of the X Keyboard Extension, XKB Version 1.0, will be different from XKB Version 0.65, which is shipping with Tru64 UNIX Version 4.0B. Avoid creating code that directly references the XKB API and data structures. Any X clients created with direct references must be recompiled and relinked when XKB Version 1.0 is shipped in a future release. You may also have to modify your source code.

G.5.2 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.6 Commands and Utilities

The following sections describe new or changed commands and utilities that are available in Tru64 UNIX Version 4.0 and Version 4.0B.

G.6.1 Changes to Mtools

Mtools software is included in the OSFDOCTOOLS410 subset. In prior releases, the software was installed by an optional worldwide support subset.

G.6.1.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.6.2 sendmail Utility Supports Configurable GECOS Fuzzy Matching

The sendmail utility now allows the user to configure the fuzzy logic for mail delivery. Previously, if the recipient's address did not precisely match any of the user names on the host, a best-match algorithm was applied against the GECOS field in the passwd file. If a unique best-match was found, the mail was delivered to this user. This behavior can now be configured at run time using the -oG option on the command line. See sendmail.cf(4) for more information.

G.6.2.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.6.3 df Supports Large File Systems

The field width for the Iused and Ifree fields in the output of the df command has been increased to accommodate 12 digits when using the -i switch. This modification was made to support very large file systems where the number of inodes could exceed the field width that was previously set aside for these fields.

G.6.3.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.6.4 Compressed Reference Pages

To economize on disk space, reference pages are now shipped in compressed format. Compressed files were created with the /usr/bin/gzip utility. The man and xman utilities automatically uncompress the reference pages.

The catman command has also been enhanced to work with compressed catman files. All three commands, man, xman and catman, still provide support for uncompressed manpages. The CDE online help viewer also automatically uncompresses reference pages when they are accessed via a hyperlink in a help volume.

For more information, see man(1) and catman(8).

G.6.4.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.6.5 Enhancements to terminfo

Terminal support has been enhanced to support non-Compaq terminals. Entries have been added to the terminfo databases and the termcap file to enable this support. New tools have also been added to assist users in modifying or porting other termcap and terminfo entries to Tru64 UNIX. These include the following:

- captoinfo–Converts termcap files to terminfo entries.
- infocmp–Uncompiles and, if required, compares terminfo entries.

The tput and tic utilities have also been enhanced.

G.6.5.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.6.6 GNU Emacs Version 19.28

GNU Emacs has been updated to Version 19.28. This version is not upwardly compatible with GNU Emacs Version 18.5, the previous version shipped with Tru64 UNIX. See the appropriate GNU Emacs documentation in `/usr/lib/emacs/etc`.

G.6.6.1 ULTRIX Migration Issues

See the GNU Emacs documentation.

G.6.7 Performance Manager

Performance Manager is a real-time performance monitor that allows users to detect and correct performance problems. Graphs and charts can show hundreds of different system values, including CPU performance, memory usage, disk transfers, file-system capacity, and network efficiency. Thresholds can be set to alert you to correct a problem when it occurs, and commands can be run on multiple nodes from the graphical user interface.

G.6.7.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.6.8 Bootable Tape

This release introduces the ability to create a standalone bootable tape of the operating system. You can boot from the bootable tape as easily as you can boot from CD-ROM or a RIS area, but without the overhead of selecting or installing subsets. When you restore your system from the bootable tape, you must reconfigure your system using the System Management applications. You will need to adjust system parameters, such as the host name or IP address.

The binaries and shell scripts needed to create and restore a bootable tape are installed with the base operating system. The files reside in `OSFBINCOM410` and no other subsets are needed. `OSFBINCOM410` is the Kernel Header and Common Files (Kernel Build Environment) subset.

Use the `btcreate` utility to create a standalone bootable tape. To extract and restore file systems from tape at the single-user level, you use the `btextract` utility.

For more information, see `btcreate`(8) and `btextract`(8).

G.6.8.1 ULTRIX Migration Issues

Bootable tape capabilities do not exist on ULTRIX operating systems: there are no ULTRIX migration issues.

G.6.9 Partition Overlap Checks Added to Disk Utilities

Partition overlap checks have been enhanced or added to the following commands:

newfs	ufs_fsck	mount

The checks ensure that partitions will not be overwritten if they are marked in use in the fstype field on the disk label. The overlap checks also ensure that no overlapping partition is marked in use.

If a partition or an overlapping partition has an in-use fstype field in the disk label, the following commands inquire interactively if a partition can be overwritten or not:

newfs	mkfdmn	addvol
swapon	voldisk	voldisksetup

See the reference pages for more information.

Partition overlap checks have been generalized by creating two library functions: check_usage and set_usage. Two new fstype values have been added: FS_RAW and FS_DB. For example, you can use the library function set_usage with database applications to set the fstype field of a disk partition that is in use by the database. Similarly, you can use check_usage to determine the usage of a disk partition or any overlapping partition.

G.6.9.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.6.10 scsimgr Utility for Creating Device Special Files

The scsimgr utility creates device special files for newly attached disk and tape devices. This utility is automatically invoked at system boot time. You can execute the command to add device special files for all disk and tape devices attached to a specified SCSI bus at any time. See the scsimgr(8) reference page for further details.

G.6.10.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.7 Standards

This release complies with many new and changes standards. See `standards`(5) for more information.

G.7.1 Realtime is Compliant with Final POSIX 1003.1b Standard Interfaces

Tru64 UNIX Version 4.0 now completes the implementation of the POSIX 1003.1b standard interface as approved by the IEEE standards board in September 1993 (IEEE Std 1003.1b-1993, Realtime Extension). The new features are described in Section G.8.9, Section G.8.10, and Section G.8.11. See the *Guide to Realtime Programming* for more information.

G.7.1.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.7.2 DECthreads is Compliant with Final POSIX 1003.1c Standard Interfaces

The DECthreads library `libpthread.so` now implements the POSIX 1003.1c standard interface as approved by the IEEE standards board in June 1995 (IEEE Std 1003.1c-1995, POSIX System Application Program Interface). The new POSIX (pthread) interface supported with DECthreads is the most portable, efficient, and powerful programming interface for a multithreaded environment. These interfaces are defined by `pthread.h`. See the *Guide to DECthreads* for more information.

G.7.2.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8 Development Environment

Tru64 UNIX Version 4.0B includes the enhancements to the development environment that are discussed in the following sections.

G.8.1 Tcl/Tk Availability

Tcl/Tk is now available as part of the base operating system. Tcl/Tk is a public domain unencumbered scripting language and graphical tool kit. In addition to Tcl/Tk, a popular extension package, TclX is also included. TclX provides many UNIX extensions to the Tcl command language. Tcl version 7.4, Tk version 4.0, and TclX version 7.4 are included in this release. See the *Installation Guide* for information on how to identify and install the appropriate software subsets.

The available programs are:

- `/usr/bin/tcl`

  A tcl shell with TclX extensions
- `/usr/bin/tclsh`

  A hard link to `/usr/bin/tcl`
- `/usr/bin/wishx`

  A Tcl/Tk/tclX shell
- `/usr/bin/wish`

  A hard link to `/usr/bin/wishx`
- `/usr/bin/tclhelp`

  A graphical help browser for Tcl help

G.8.1.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.2 DEC C++

The following changes have been implemented for DEC C++:

- DEC C++ Version 5.3 Class Library is now threadsafe. See the *DEC C++ Class Library Reference Manual* for details on the threadsafe support, including a new Mutex Package.
- Complex division catches divide-by-zero errors. The division routines within the Complex Library now catch divide-by-zero errors instead of signaling them.
- Iostream assignment operators. For `iostream` assignment operators, there is no longer a memory leak when you use the `*_withassign` assignment operators to initialize an object for which you have called `xalloc()`. Previously, the memory allocated for the object by `xalloc()` was lost.
- String extraction operator. The String extraction operator now takes care of dynamically allocating the String to accommodate the input.
- `ios::ate` mode. When you open a file specifying `ios::ate` but not `ios::app` to the `filebuf open()` function, the file is no longer opened in `O_APPEND` mode. This incorrect behavior caused all data to be written to the end of the file, regardless of the current file position.

- Exception handling. Various problems with exception handling have been fixed. Also, support for exception handling in DEC C++ Version 5.3 has been added.
- Function `exp()` returns zero for underflow errors .When the Complex Library `exp()` function detects an underflow error, the resulting value is now (0,0) instead of (+/- max-float, +/- max-float).
- Use of `clog()` and C++ Class Library iostream `clog`. A single application is restricted from using both the math library function `clog()` and the iostream package's `clog` object. This restriction is due to the fact that `libm` and `libcxx` each contain a definition for the global symbol `clog` and those definitions are incompatible. Furthermore, applications which reference one of the `clog` symbols cannot include both `-lcxx` and `-lm` on their `ld` command line. An error will be generated by `ld` because `clog` is multiply defined.
- `catch(...)`clause. The `catch(...)` clause now catches C structured exceptions.
- `fstream close()` clears the error state. The `fstream`, `ifstream`, and `ofstream close()` member functions now clear the stream's error state when the close succeeds. Call the `clear()` member function after the call to `close`().

G.8.2.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.3 Software Development Environment Repackaging

The Software Development Environment (SDE) has been repackaged to ease installation, simplify licensing, and create a product identity. The current SDE components have been repackaged into a single `OSFSDE` subset, and all of the pieces outside the SDE have been moved into logical subsets, including:

- `OSFINCLUDE` for all include files
- `OSFLIBA` for all static libraries
- `OSFPGMR` for commands outside the scope of the SDE

Because the compiler is needed at installation time, some SDE components have remained in the mandatory `OSFCMPLRS` subset.

The Ladebug debugger subsets have been renamed to the `OSF*` subset name prefix and can now be installed during a custom installation of Tru64 UNIX. These changes have been made on the *Tru64 UNIX Operating System Volume 1 CD–ROM*. The FUSE Porting Assistant has been added to the Tru64 UNIX kit on the *Tru64 UNIX Associated Products Volume 1 CD–ROM*.

This is a tool to help port code to Tru64 UNIX from a variety of platforms and operating systems.

The OSFSDECDE subset was also added to the *Tru64 UNIX Operating System Volume 1 CD–ROM*. It contains the files necessary to access DECladebug and the Porting Assistant from CDE.

G.8.3.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.4 init Execution Order Modified for Static Executable Files

The execution order for init routines in static executable files has been modified to more closely match the execution order for init routines in dynamic executable files. The init routines loaded from an archive library will be executed prior to any init routines loaded from objects and archives occurring earlier on the linker command line. Prior to this change, init routines were executed in the order they were encountered in processing the link command from left to right. As a result, init order for static executable files was much different than the init order for equivalent shared executable files.

For existing applications that rely on the static init order used in prior releases of Tru64 UNIX, you can use the new linker option -old_init_order to restore the strict left-to-right execution order for static executable files.

G.8.4.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.5 PC-Sample Mode of prof Command

The prof command's pc-sampling mode now supports profiling the shared libraries used by a program. Linking a call-shared program with the cc command's -p switch causes the resulting program to profile both the call-shared executable file and all the shared libraries. The following command displays a combined profile:

```
# prof -all
```

New -all, -incobj, -excobj, and -stride switches for the PROFFLAGS environment variable enable you to request per-procedure profiling of the shared libraries or to select particular libraries to profile.

The related enhancements are:

- Extended application programming interfaces (APIs) to `monitor()`, `monstartup()`, and `profil()`
- Use of 32-bit pc-sampling counters instead of 16-bit for `cc -p` and `cc -pg` profiling (`gprof`), except for calls to the traditional `monitor()` API.
- Improved reliability in profiling multithreaded programs, and reference page guidelines for use of `monitor_signal()` with threads.
- `prof` and `gprof` checking.
- Profiling report formats are improved.

See `prof`(1) and `monitor`(3) for further information.

G.8.5.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.6 atom and prof Commands and Threads

Both of the following `atom` and `prof` commands now profile the shared libraries used by a program:

```
# atom -tool pixie -all
```

```
# prof -pixie -all
```

The `threads` environment for `atom` also makes the `pixie` tool thread-safe, though per-thread counts are not recorded.

Additionally, there are new file formats for `.Addrs` and `.Counts` files.

See `atom` (1), `prof` (1), and `pixie` (5) for further information.

G.8.6.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.7 Thread Independent Services Interface

Tru64 UNIX Version 4.0B introduces the Thread Independent Services (TIS) application programming interface in the C run-time library `libc`. TIS provides services that assist in the development of thread-safe libraries.

Thread synchronization may involve significant run-time cost, which is undesirable in the absence of threads. TIS enables thread-safe libraries to be built that are both efficient in the nonthreaded environment, yet provide the necessary synchronization in the threaded environment.

When DECthreads (pthreads) are not active within the process, TIS executes only the minimum steps necessary. Code running in a nonthreaded environment does not encounter overhead incurred by the run-time synchronization that is necessary when the same code is run in a threaded environment. When DECthreads are active, the TIS functions provide the necessary thread-safe synchronization.

G.8.7.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.8 High-Resolution Clock

Tru64 UNIX Version 4.0B has an optional high-resolution clock. To enable this option, add the following line to the kernel configuration file and rebuild the kernel:

```
options MICRO_TIME
```

The system clock (CLOCK_REALTIME) resolution as returned by `clock_getres` will not change. Timer resolution remains the same. However, time as returned by the `clock_gettime` routine will now be extrapolated between the clock ticks. The granularity of the time returned will now be in microseconds. The time values returned are SMP safe, monotonically increasing, and have 1 microsecond as the apparent resolution.

G.8.8.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.9 POSIX 1003.1b Realtime Signals

Realtime signals have been implemented to conform to the POSIX 1003.1b standard. This new feature includes queued signals with optional data delivery, and 16 user-definable realtime signals.

The following functions to support realtime signals were implemented:

- `sigqueue`
- `sigtimedwait`
- `sigwaitinfo`
- `timer_getoverrun`

G.8.9.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.10 POSIX 1003.1b Synchronized I/O

Synchronized I/O (file synchronization) has been implemented to conform to the POSIX 1003.1b standard. New functions for synchronized I/O under the UFS and AdvFS file systems include:

- `aio_fsync`

  Asynchronously writes changes in a file to permanent storage

- `fdatasync`

  Writes data changes in a file to permanent storage

The `open` function now takes the following new flags for synchronized I/O:

- `O_DSYNC`

  Ensures synchronized I/O data integrity of the file accessed

- `O_RSYNC`

  Used for synchronized I/O read operations

G.8.10.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.11 POSIX 1003.1b _POSIX_C_SOURCE Symbol

For applications conforming to POSIX 1003.1b, the `_POSIX_4SOURCE` macro is supported for Tru64 UNIX Version 4.0B, but will be retired with the next release of Tru64 UNIX. The macro `_POSIX_4SOURCE` is part of an obsolete draft standard and is supported in this release for compatibility only. When possible, existing applications that use `_POSIX_4SOURCE` should be modified to use `_POSIX_C_SOURCE` instead.

The `_POSIX_C_SOURCE` macro is associated with a value, which allows an application to specify the namespace it requires. However, as a general rule, avoid explicitly defining standards macros when compiling your applications. For most applications, the header file `unistd.h` provides the standards definitions that are needed.

G.8.11.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.8.12 Porting Assistant

The Porting Assistant is a Motif-based tool to help you port your C, C++, and Fortran source code to Tru64 UNIX from other UNIX and proprietary platforms, including OpenVMS. The Porting Assistant includes the following features:

- Uncovers 32-bit dependencies
- Checks your makefile commands and options
- Helps find functions that your application needs
- Helps develop code segments specific to Tru64 UNIX
- Provides additional information on porting your application

The Porting Assistant is licensed and provided to you with the Tru64 UNIX Developers' Toolkit but requires separate installation.

To install Version 2.0 of the Porting Assistant, install subsets `PRTBASE200` and `PRTMAN200` (and their dependencies) from the *Tru64 UNIX Associated Products Volume 1 CD-ROM*.

G.8.12.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.9 Networking

The following sections describe networking enhancements contained in Tru64 UNIX Version 4.0B.

G.9.1 New Version of the gated Daemon

This release includes a new version of the `gated` routing daemon. The update installation procedure will detect if your system is configured to run the `gated` routing daemon. If the Compaq supplied `gated` is detected, then the `/etc/gated.conf` file is moved to `/etc/ogated.conf`. Otherwise, if a user-supplied or customized `gated` is detected, then both the `/etc/gated.conf` and the `/usr/sbin/gated` files are saved with the `.PreUPD` suffix.

When the system is installed, the new gated R3.5 is the default version in `/usr/sbin/gated`. The old `gated` Version 1.9 is supplied in `/usr/sbin/ogated`. Also, corresponding, older `gated` reference pages are saved with an `o` prefix.

G.9.1.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.9.2 Dynamic Host Configuration Protocol

This release contains both a client and a server Dynamic Host Configuration Protocol (DHCP) daemon. For DHCP client configuration, use the `netconfig` utility. For configuration of client parameters on the DHCP server, use the `/usr/bin/X11/xjoin` utility, which provides a graphical user interface to the `/etc/bootptab` file.

For more information on DHCP, see `joinc`(8) and `joind`(8).

G.9.2.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.9.3 Point-to-Point Protocol

This release supports Point-to-Point Protocol (PPP), including support for BSD-style compression of entire packets. This is a negotiated option. If a foreign peer cannot handle this, it should be gracefully rejected via the `Protocol-Reject` of the Link Control Protocol (LCP).

When using PPP with modems doing compression, it may be desirable to force no BSD-style compression. To do this, put `-bsdcomp` in either `/etc/ppp/options`, or on the `pppd` command line.

PPP now has a configurable (at boot time) number of interfaces. The default is 1. To specify a higher value, add the following line to the `/etc/sysconfigtab` file and reboot the system:

```
ppp:nppp=x
```

PPP documentation is available in `pppd`(8), `pppstats`(8), and `chat`(8), and in the *Network Administration* manual.

G.9.3.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.9.4 Extensible Simple Network Management Protocol

A new Simple Network Management Protocol (SNMP) architecture is present in this release. The SNMP daemon, `snmpd`, is now an extensible master agent. End-user programmers can develop subagent programs that communicate with `snmpd` to implement their management information bases (MIBs) on Tru64 UNIX systems.

The base operating system MIB support is implemented in a subagent program called `os_mibs`, which is started or stopped automatically with `snmpd`.

G.9.4.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.9.5 SNMP MIB Support

This release supports the Host Resources MIB (RFC 1514). The MIB support daemon must query the system's devices to retrieve information required for this MIB. This query occurs when the daemon starts, and subsequently whenever a relevant SNMP request arrives.

This device querying is the default behavior, and may be configured off. See `snmpd`(8) for more information about configuring the SNMP agent.

G.9.5.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.10 Enhanced Security

This release provides the following new enhanced security features:

- Support for per-user resource limits in user profiles, using `setrlimit`.
- Nonshadowed passwords are allowed, while using other extended profile features.
- The system administrator can control whether the `ttys` database is updated on logins.
- Wildcard support for `ttys` has been extended to X displays.
- User profiles and `ttys` information are stored in database files for faster access and update (resulting in faster logins).
- The new utilities `edauth` and `convuser` are available.

See the *Security* manual and `setrlimit`(2), `edauth`(8), and `convuser`(8).

G.10.1 ULTRIX Migration Issues

ULTRIX has had enhanced security since 1990; now Tru64 UNIX has it. Differences that affect migration are discussed in the *Security* manual, in an appendix on migration.

G.11 File Systems

The following sections describe file system enhancements have been implemented in Tru64 UNIX Version 4.0B.

G.11.1 Advanced File System

The following sections describe Advanced File System (AdvFS) enhancements have been implemented in Tru64 UNIX Version 4.0B.

G.11.1.1 New Tuning Parameters for AdvFS

There is a new mechanism for limiting the amount of kernel memory that AdvFS uses for its access structures. This may be necessary only for systems with 64 MB or less memory, and AdvFS as the default file systems. This is applicable to all hardware configurations.

There are two new kernel parameters relevant to AdvFS that you can modify using the `sysconfig` or `sysconfigdb` commands. They are `AdvfsAccessMaxPercent` and `AdvfsAccessCleanupPercent`.

G.11.1.2 AdvFS Now Supports Directory Truncation

Traditionally, AdvFS directories were never truncated, even though many of the files in the directory had been deleted. This created a problem if the directory file became very big. For example, if several hundred thousand files were added to a directory, then the directory file itself grew very large. Even though most of the files in that directory were subsequently deleted, operations that required scanning the directory remained inefficient because the entire directory file still needed to be read.

AdvFS now truncates directory files when all of the entries at the end of the directory have been deleted. This truncation is done on 8 KB boundaries, so the size of a directory is always a multiple of 8192.

One ambiguity of directory truncation is that the truncation is done when files are created and not when they are deleted. This is done because of the efficiency of underlying algorithms, and is the same model used by UFS for directory truncation. For example, after most files in a given directory are deleted, the size of the directory file itself will not decrease until a new file is inserted into that directory.

G.11.1.3 ULTRIX Migration Issues

The AdvFS file system does not exist on ULTRIX systems, so there are no migration issues.

G.11.2 File System Access Control Lists

Access Control Lists (ACLs) on files and directories are a new feature in this release. They are manipulated with the `getacl` and `setacl` commands. See the *Security* manual and the reference pages for more information.

G.11.2.1 ULTRIX Migration Issues

The ULTRIX operating system does not support ACLs or property lists (ACLs are implemented as a specific type of property list), so there are no ULTRIX migration issues.

G.11.3 Logical Storage Manager

Tru64 UNIX now provides the following new features for the Logical Storage Manager (LSM):

- Two new LSM commands, `volsave` and `volrestore`, provide an easy way to back up and restore the LSM configuration database. See the reference pages for these commands.
- The Basic Operations menu in LSM's graphical interface, `dxlsm`, now provides support for disk operations. For example, how to add a disk to LSM.
- The LSM limits have increased as follows:
  - The maximum number of LSM volumes on a system has increased from 256 to 4093.
  - The maximum number of plexes on a system has increased from 256 to 4096.
  - The maximum number of subdisks in a plex has increased from 256 to 4096.
  - The maximum number of disks that can be added to LSM has increased from 128 to 256.
  - The maximum size of an LSM volume has increased from 128 GB to 512 GB.

The functionality and syntax of the LSM commands used for encapsulation, unencapsulation, and mirroring have changed in this release, as follows:

- The `volencap` command now supports the following features and functions. For details, see `volencap`(8).
  - Allows the initialization of LSM and encapsulation of the system disk in one step. This requires the use of a free partition table entry.
  - Can be used to encapsulate all partitions on a disk. This requires the temporary use of a free partition table entry if the system disk is being encapsulated.
  - Can be used to encapsulate only the root and swap partitions.
  - Automatically creates a new disk group if specified.
  - Subsumes the functionality of the `voladvdomencap` command.

- Takes multiple arguments.
- Uses a simple disk instead of a sliced disk for system disk encapsulation.
- For disk label characteristics, assumes that partition `c` maps the entire disk, and that an in-use partition has an *fstype* field other than UNUSED. (If a partition's *fstype* field is UNUSED, then `volencap` may allocate that partition table entry for its use.)

- The `volrootmir` command now supports the following features and functions. For details, see `volrootmir`(8).
  - Can be used to mirror all volumes on the system disk by specifying the `-a` option. This option requires the target disk to be of the same type as the source disk.
  - Can be used to encapsulate only the root or swap partition by omitting the `-a` option. This procedure requires that the target root and swap partitions are large enough to hold `rootvol` and `swapvol`, but the target and source disks need not be of the same type.
- When used with the `-a` option, the `volunroot` command unencapsulates all LSM volumes on the system disk, not just `rootvol` and `swapvol`. The requirements for unencapsulation are:
  - The partition associated with the volume must have been initialized as a `nopriv` disk.
  - The volume must map directly to the partition (that is, the volume size must be equivalent to the partition size).
  - The volume must not be mirrored.

  For details, see `volunroot`(8).

G.11.3.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.11.4 Overlap Partition Checking

Two new functions, `check_usage` and `set_usage`, are available for use by applications. These functions check whether a disk partition is marked for use and set the `fstype` of the partition in the disk label. See the reference pages for these functions for more information.

G.11.4.1 ULTRIX Migration Issues

There are no ULTRIX migration issues.

G.12 Internationalization and Language Support

The following sections describe the new features implemented in Tru64 UNIX Version 4.0B to support internationalization. There are no ULTRIX migration issues.

G.12.1 Internationalization Configuration Utility for CDE

The Internationalization (I18N) Configuration Tool, available through the CDE Application Manager, is one of the SysMan system administration configuration tools. The I18N Configuration Tool provides a graphical interface that enables you to configure internationalization-specific settings. It also provides a convenient way to see which countries, locales, fonts, and keymaps are currently supported on your system. Use this tool to remove unused fonts and unrequired country support.

G.12.2 Unicode Support

This release provides a new set of locales and codeset converters that support the Unicode and ISO 10646 standards. The codeset converter modules enable an application to convert between other supported codesets and UCS-4.

Tru64 UNIX also provides a function called `fold_string_w()` that maps one Unicode string to another, performing the specified Unicode character transformations. For more information, see `fold_string_w`(3).

For more information on the Unicode support, see `Unicode`(5).

G.12.3 The Worldwide Mail Handler No Longer Exists

Worldwide support subsets no longer install internationalized Mail Handler (MH) software in the `/usr/I18N/bin/mh` directory. In Tru64 UNIX Version 4.0B, internationalization features have been merged into the default Mail Handler (MH) whose files are located in `/usr/bin/mh`. Check the value for the `mhpath` resource used to find the DECwindows Mail application. If necessary, change this value to be `/usr/bin/mh`.

G.12.4 Multilingual Emacs (mule)

The `mule` editor is a multilingual version of GNU Emacs and supports the following kinds of characters:

- ASCII (7-bit)
- ISO Latin-1 (8-bit)
- Japanese, Chinese, and Korean (16-bit) as specified by the ISO 2022 standard and its variants (EUC, Compound Text, and so on)

- Chinese in both GB and Big 5 encodings
- Thai as specified by the TIS 620 standard

The IOSWWMULE400 subset installs Version 2.3 of the GNU mule editor and associated software. Corresponding sources are available in the IOSWWMULESRC400 subset.

Tru64 UNIX does not include public domain fonts that you can use with mule. See the mule-2.3/README.Mule file installed by the IOSWWMULESRC400 subset to find out how you can obtain public domain fonts.

The Tru64 UNIX software is enhanced with lisp libraries that support the dechanzi codeset for Simplified Chinese and the dechanyu codeset and tsangchi input method for Traditional Chinese. These libraries are included in the IOSWWMULE400 subset and installed in the /usr/i18n/mule/lib/mule/site-lisp directory.

For more information about mule, see mule(1).

G.12.5 Support for Catalan, Lithuanian, and Slovene

Tru64 UNIX Version 4.0B includes support for Catalan, Lithuanian, and Slovene program localization. See Catalan(5), Lithuanian(5), and Slovene(5) for information about associated codesets, locales, keyboards, and fonts.

G.12.6 man Command Supports Codeset Conversion

The man command can automatically invoke the iconv utility to perform codeset conversion of reference page files. This allows you to install one set of reference pages to support locales that have the same language and territory but different codesets, thereby reducing file redundancy on the system. For more information, see man(1).

G.13 Dynamic Device Recognition for SCSI Devices

Dynamic Device Recognition (DDR) is a framework for describing the operating parameters and characteristics of SCSI devices to the SCSI CAM I/O subsystem. You can use DDR to include new and changed SCSI devices into your environment without having to reboot the operating system. You do not disrupt user services and processes, as happens with static methods of device recognition.

Beginning with Tru64 UNIX Version 4.0, DDR is preferred over the current, static method for recognizing SCSI devices. The current, static method, as described in *System Administration*, is to edit SCSI device customizations into the /sys/data/cam_data.c data file, reconfigure the kernel, and shut down and reboot the operating system.

Note

Support for the static method of recognizing SCSI devices will be retired in a future release of Tru64 UNIX.

Tru64 UNIX Version 4.0 and Version 4.0B support both methods of recognizing SCSI devices. Both methods can be employed on the same system, with the restriction that the devices described by each method are exclusive to that method (nothing is doubly defined).

The information DDR provides about SCSI devices is needed by SCSI drivers. You can supply this information using DDR when you add new SCSI devices to the system, or you can use the /sys/data/cam_data.c data file and static configuration methods. The information provided by DDR and the cam_data.c file have the same objectives. When compared to the static method of providing SCSI device information, DDR minimizes the amount of information that is supplied by the device driver or subsystem to the operating system, and maximizes the amount of information that is supplied by the device itself or by defaults specified in the DDR databases.

You can also use DDR capabilities to convert customizations in the cam_data.c file to information in the DDR /etc/ddr.dbase text database.

For more information about DDR, see *System Administration*, ddr_config(8), and ddr.dbase(4).

G.13.1 ULTRIX Migration Issues

Because dynamic device recognition does not exist on ULTRIX systems, it does not affect migration. However, in a future release of Tru64 UNIX, the name space for SCSI devices will increase, and that change will affect current versions of both operating systems.

G.14 Interfaces Retired from Tru64 UNIX

With the release of Tru64 UNIX Version 4.0, several features of previous versions of the operating system were retired. The documentation for previous versions of Tru64 UNIX announced that these features would be retired. The retired features and their ULTRIX migration issues (if any) are:

- Support for ULTRIX RIS to Tru64 UNIX client functionality

  ULTRIX migration issues: If you have been using ULTRIX systems as RIS servers for Tru64 UNIX client systems, that capability will not work for a Tru64 UNIX Version 4.0 client. You will need to serve Tru64 UNIX

Version 4.0 clients from Tru64 UNIX RIS servers. ULTRIX systems can still RIS serve ULTRIX client systems.

- The `oawk` version of the `awk` command

  There are no migration issues because the `owak` command does not exist in ULTRIX.

- Routines that were duplicated in the`libc` and `libm` libraries have been removed from the `libc` library.

  ULTRIX migration issue: application developers might have to add the `-lm` option to their compiler command line.

- The `-n` option from `/usr/bin/echo` and `/bin/echo`

  There are no ULTRIX migration issues.

- Ethernet trailer encapsulation

  There are no ULTRIX migration issues.

- Linkworks run-time library

  There are no ULTRIX migration issues.

- Logical Volume Manager

  There are no ULTRIX migration issues.

- Obsolete POSIX real-time interfaces

  There are no ULTRIX migration issues.

- XIE V3.0 interface, server support (although run-time support will still be provided transparently through the client)

  There are no ULTRIX migration issues.

- The POLYCENTER Common Agent (extensions to the SNMP V1.0 agent)

  There are no ULTRIX migration issues.

G.15 Features Scheduled for Retirement

Read the Tru64 UNIX release notes for information about those features scheduled for retirement in future releases of the operating system. Knowledge of these pending changes will help you determine wise migration tactics.

Index

D

E

F

G

H

I

M

N

O

P

R

S

T

U

V

W

X

Y

How to Order Tru64 UNIX Documentation

To order Tru64 UNIX documentation in the United States and Canada, call **800-344-4825**. 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/`

If you need help deciding which documentation best meets your needs, see the Tru64 UNIX *Documentation Overview*, which describes the structure and organization of the Tru64 UNIX documentation and provides brief overviews of each document.

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

Name	Order Number
Tru64 UNIX Documentation CD-ROM	QA-MT4AA-G8
Tru64 UNIX Documentation Kit	QA-MT4AA-GZ
End User Documentation Kit	QA-MT4AB-GZ
Startup Documentation Kit	QA-MT4AC-GZ
General User Documentation Kit	QA-MT4AD-GZ
System and Network Management Documentation Kit	QA-MT4AE-GZ
Developer's Documentation Kit	QA-MT5AA-GZ
General Programming Documentation Kit	QA-MT5AB-GZ
Windows Programming Documentation Kit	QA-MT5AC-GZ
Reference Pages Documentation Kit	QA-MT4AG-GZ
Device Driver Kit	QA-MT4AV-G8

Reader's Comments

Tru64 UNIX
ULTRIX to Tru64 UNIX Migration
AA-PS3EE-TE

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

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

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

Please rate this manual:

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

Please list errors you have found in this manual:

Page Description

________ ____

________ ____

________ ____

________ ____

Additional comments or suggestions to improve this manual:

____

____

____

____

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

Name, title, department ____________________________________

Mailing address ____________________________________

Electronic mail ____________________________________

Telephone ____________________________________

Date ____________________________________

Do Not Cut or Tear - Fold Here and Tape

COMPAQ

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

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

POSTAGE WILL BE PAID BY ADDRESSEE

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

Do Not Cut or Tear - Fold Here

Cut on This Line