Tru64 UNIX

X Window System Environment

Part Number: AA-RH9JC-TE

September 2002

Product Version: Tru64 UNIX Version 5.1B or higher

This manual contains information for system administrators and programmers about the Tru64 UNIX implementation of the X Window System Release 6.5 (X11 R6.5). This manual also contains information about customizing the Tru64 UNIX window system workstation environment.

Hewlett-Packard Company
Palo Alto, California

Contents

3 Programming in the Tru64 UNIX X Window Environment

Index

Examples

Figures

Tables

About This Manual

The *X Window System Environment* manual discusses various aspects of the X Window System (X11) Release 6.5 implementation on the HP Tru64 UNIX operating system.

Audience

This manual provides general information as well as specific information about the X Window System as supplied by the Tru64 UNIX operating system. The audience for this information includes end users, system administrators, and applications programmers.

New and Changed Features

This manual has been revised to document X11 R6.5.

Organization

This document has three chapters:

Chapter 1	Discusses X Window System system administration topics for Tru64 UNIX systems.
Chapter 2	Explains how to customize X environment resources and `keysyms` on Tru64 UNIX systems.
Chapter 3	Discusses X server extensions that are part of the Tru64 UNIX X Window System environment as well as other programming topics that apply to Tru64 UNIX systems.

Related Documents

The information in the *X Window System Environment* manual supplements information found in the following book:

- *X Window System User's Guide OSF/Motif 1.2 Edition*, Valerie Quercia and Tim O'Reilly, O'Reilly & Associates, Inc.
- *X Window System*, Scheifler and Gettys
- *X Window System Toolkit*, Asente and Swick, Digital Press
- Additional information can be found at the X.Org Web site at **http://www.x.org**.

Icons on Tru64 UNIX Printed Manuals

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

G	Manuals for general users
S	Manuals for system and network administrators
P	Manuals for programmers
R	Manuals for reference page users

Some manuals in the documentation help meet the needs of several audiences. For example, the information in some system manuals 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 manuals in the Tru64 UNIX documentation set.

Reader's Comments

HP 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`

Please include the following information along with your comments:

- The full title of the manual and the order number. (The order number appears on the title page of printed and PDF versions of a manual.)
- 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 HP technical support office. Information

provided with the software media explains how to send problem reports to HP.

Conventions

This document uses the following typographical and symbol conventions:

`%` `$`	A percent sign represents the C shell system prompt. A dollar sign represents the system prompt for the Bourne, Korn, and POSIX shells.
`% 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.

1

X Window System Administration in the Tru64 UNIX Environment

This chapter provides information about administering the X Window System environment for systems running the Tru64 UNIX operating software.

This chapter includes information on the following topics:

- Display managers (Section 1.1)
- Locations of the X Window System files on Tru64 UNIX (Section 1.2)
- X Display Manager (`xdm`) and the login process (Section 1.3)
- Security and `xdm` authorization (Section 1.4)
- X Window System login problems (Section 1.5.1)
- X server management (Section 1.6)
- Graphics adapters (Section 1.7)
- Font server management (Section 1.8)
- X terminal management (Section 1.9)
- X server memory utilization (Section 1.10)

1.1 Choosing the xdm or the dtlogin Display Manager

You can configure your system to run either the standard X11 R6 display manager, `xdm`, or the Common Desktop Environment (CDE) display manager, `dtlogin`. Run the `/usr/sbin/xsetup` script to switch between CDE and `xdm`. The `xsetup` script sets the value of the `/etc/rc.config` variable `XLOGIN` to be `xdm` or `cde` and will optionally restart your X display manager using the `/sbin/init.d/xlogin` script. When your system boots, the `/sbin/init.d/xlogin` script uses the value of the `/etc/rc.config` `XLOGIN` variable to determine whether to start `xdm` or CDE `dtlogin`(1).

If for any reason you need to restart your X display manager, use `xsetup`, whether or not you choose to switch from one display manager to another. Alternatively, the X display manager can be stopped, started, or restarted using the `/sbin/init.d/xlogin` command with `stop`, `start`, or `restart` specified as the parameter.

For more information on configuring CDE and `dtlogin`(1), see the *Common Desktop Environment: User's Guide*.

The information in the rest of this chapter primarily applies if you choose to run `xdm`. While CDE is similar to `xdm` and uses the same methods and concepts, there are important differences in the details.

1.2 Locations of the X Window System Files

The file locations shown in the following list reflect the locations of the X Window System files as established by the installation kits.

Files	Contents
`/usr/bin/X11`	X binaries. (In some previous X implementations, some X binaries were located in `/usr/bin`.)
`/usr/bin/X11/demos`	Binaries of X demo programs.
`/usr/examples`	Example files and (possibly) program sources.
`/usr/include/DXm`	DECwindows Motif widget header files.
`/usr/include/Mrm`	Motif resource manager header files.
`/usr/include/uil`	User Interface Language (UIL) header files.
`/usr/include/X11`	X11 header files.
`/usr/include/X11/bitmaps`	Bitmaps used by various window managers and applications.
`/usr/include/X11/extensions`	Header files for extensions to X11 R6. (The extensions are discussed in Chapter 3.)
`/usr/include/X11/ICE`	InterClient Exchange library header files.
`/usr/include/X11/SM`	Session management library header files.
`/usr/include/X11/Xaw`	Athena widget header files.
`/usr/include/X11/Xmu`	X utility header files.
`/usr/include/X11/Xserver`	Header files for loadable X server libraries.
`/usr/lib/lib*`	Developers' libraries (static versions).
`/usr/lib/X11/app-defaults`	Application default files used by applications to define default interface configurations and, in some cases, the layout of applications.

Files	Contents
/usr/lib/X11/locale/C	Internationalization files.
/usr/lib/X11/config	Configuration files that can be used to build Makefiles from Imakefiles so that developers can use more generic build configurations for their applications. These configuration files define the proper configuration parameters for the system.
/usr/lib/X11/fonts/100dpi	The 100 dpi fonts from X.Org.
/usr/lib/X11/fonts/75dpi	The 75 dpi fonts from X.Org.
/usr/lib/X11/fonts/decwin/100dpi	The 100 dpi DECwindows fonts.
/usr/lib/X11/fonts/decwin/75dpi	The 75 dpi DECwindows fonts.
/usr/lib/X11/fonts/misc	Fonts from X.Org.
/usr/lib/X11/fonts/Speedo	Speedo scalable fonts.
/usr/lib/X11/fonts/Type1	Type1 scalable fonts.
/usr/lib/X11/fonts/user/100dpi	The 100 dpi fonts from layered products and local installations.
/usr/lib/X11/fonts/user/75dpi	The 75 dpi fonts from layered products and local installations.
/usr/lib/X11/fonts/user/misc	Other fonts from layered products and local installations.
/usr/lib/X11/fs	Font server configuration and error log files.
/usr/lib/X11/help	Directories in this directory contain the help files for various applications.
/usr/lib/X11/ja	Internationalization files.
/usr/lib/X11/japan	Internationalization files.
/usr/lib/X11/locale	Internationalization files.
/usr/lib/X11/keymaps	Alternate keymaps for different international keyboards.
/usr/lib/X11/nls	Natural language support for native character mappings.
/usr/lib/X11/nls/local_im_tbl	Internationalization files.
/usr/lib/X11/rgb*	Color database used by the server to convert color names to red-green-blue values.
/usr/lib/X11/system.mwmrc	Default systemwide configuration file for mwm.

Files	Contents
/usr/lib/X11/twm	Default configuration information for twm.
/usr/lib/X11/uid	User interface control files used by some applications.
/usr/lib/X11/x11perfcomp	Utility script for reformatting x11perf output.
/usr/var/X11/xkb	XKB keymap files.
/usr/lib/X11/xkb	XKB keymap files.
/usr/bin/X11/xkbcomp	XKB keymap compiler.
/usr/bin/X11/xkbprint	XKB keymap to PostScript generator.
/usr/bin/X11/xkbdfltmap	Determines the default keymap based on the console, language, and keyboard.
/usr/bin/X11/dxkbledpanel	Displays a graphical user interface of the available XKB indicators. Used primarily to show the current keyboard group. Replaces the kb_indicator application.
/var/X11/xdm	X Display Manager configuration and resource files, and the xdm error log. (The file /usr/lib/X11/xdm is a link to /var/X11/xdm.)
/usr/lib/X11/XErrorDB	Error messages used by the X library.
/usr/lib/X11/XKeysymDB	The keysym mappings for X toolkit based applications.
/var/X11/Xserver.conf	Configuration information file for the X server. (The file /usr/lib/X11/Xserver.conf is a link to /var/X11/Xserver.conf.)
/usr/shlib	Run-time shared libraries.
/usr/shlib/X11	Run-time shared libraries for the X server, font server, and xdm.
/usr/shlib/_null	Older versions of shareable libraries.
/var/X11/fs	Font server configuration files. (The file /usr/lib/X11/fs is a link to /var/X11/fs.conf.)

1.3 X Display Manager (xdm) and the Login Process

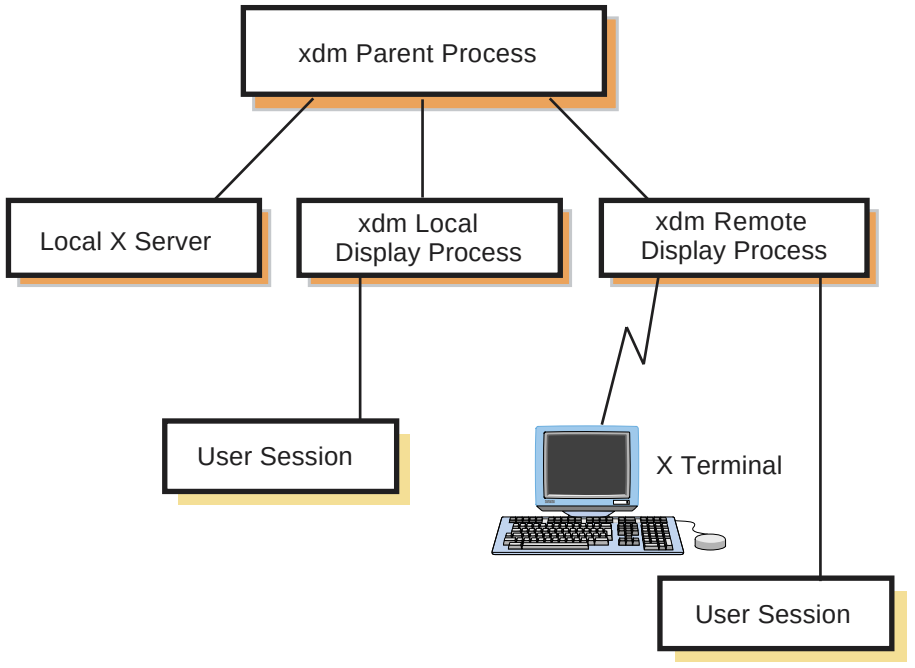
The X Display Manager (xdm) manages user sessions on both local and remote displays. On Tru64 UNIX systems, the xdm utility provides the

mechanism for logging in to the X display and then automatically starts certain client applications .

The xdm utility creates child processes for each display both locally and remotely. The xdm utility is an X client that manages user session elements, such as logging in, authentication, and default resource set up. System administrators can use xdm to make systemwide configurations of the X Window System environment.

Figure 1–1 shows the kinds of processes that xdm manages.

Figure 1–1: The xdm Processes



The xdm daemon starts when the system boots in multiuser mode, so it is ready to manage the login process. The following list shows the steps involved in the login process on Tru64 UNIX systems:

1. The system uses the following command during the booting procedure to start the xdm daemon:

   ```
   /sbin/init.d/xlogin start
   ```

   On Tru64 UNIX systems, the xdm daemon is started by an initialization script that is run using the following link:

   ```
   /sbin/rc3.d/S95xlogin -> ../init.d/xlogin
   ```

2. The xdm program reads its main configuration file:

   /usr/var/X11/xdm/xdm-config

3. The xdm program listens on its socket for requests from any X terminals.
4. The xdm program forks a child process for managing the local display.
5. The xdm program displays the login box (login widget) on the local display. For this procedure, xdm executes the following steps:

   a. Secures the display.
   b. Loads Xresources from the X server resource database utility xrdb. Loading the resources sets the display characteristics for the xdm login box.
   c. Runs the Xsetup_0 setup script on the local display.
   d. Displays the login box (login widget) on the local display.

6. The user enters his or her name and password in the login box. User authentication takes place to ensure that the user is allowed to access this display.
7. The login widget is destroyed and xdm runs the GiveConsole startup script on the local display using the root uid.
8. The X session starts up as a child process using the user's ID (uid). The startup action involves executing the /var/X11/xdm/Xsession script, which either runs the user's $HOME/.xsession script or dxsession.
9. The user exits the session.
10. The xdm program runs the /var/X11/xdm/TakeConsole script on the local display using root's UID.
11. The xdm local display process exits. At this point the workstation returns to the state it was in at step 4, where xdm forks a child process for managing the local display and displays the login widget. The workstation is ready for a user to log in.

On Tru64 UNIX, the /var/X11/xdm directory contains the following files:

- GiveConsole

  This script is run when xdm starts up and changes the ownership of the console, /dev/console, from root to the user. The script only runs on the local display. (It cannot be used with X terminals.)

- TakeConsole

  This script is run when xdm is reset and returns ownership of the console, /dev/console, from the user to root. The script only runs on the local display. (It cannot be used with X terminals.)

- `Xaccess`

  This configuration file controls how `xdm` responds to different queries from the X Display Manager Control Protocol (XDMCP). This file is used to manage X terminals.

- `Xkeymaps`

  This file defines the mapping between language and keyboard and the corresponding keymap file in the `/usr/lib/X11/keymaps` directory. Whenever the server is started or reset, the keymap is loaded into the X server by `xdm` using the `xmodmap` command.

  The value of the console language variable and the keyboard type are retrieved from the kernel and used as an index into the `Xkeymaps` table to define the appropriate keymap.

- `Xresources`

  This file contains resource specifications that are loaded into the X server's resource database before the login widget is displayed. These resources affect the appearance of the login window and screen (either `dxlogin` or `xlogin`), the background color of all clients, as well as the appearance of the clients, which are started by `xdm`: `xconsole`, `dxconsole`, and `chooser`

  The `dxlogin` resources can control the following elements:

  – Compaq logo pixmap, clipmask, foreground color, background color, and login box position

  – Root window color

  – Greeting text, font, and color

  – Prompt text as well as color and font for both the prompt and answer

- `Xservers`

  This file defines the command that starts the server on the local display. More entries for workstations or X terminals can be added as needed.

  The default definition for the UNIX socket transport (`DISPLAY :0`) is:

  ```
  :0 local /usr/bin/X11/X
  ```

  The default definition for the Shared Memory Transport (SMT) (`DISPLAY local:0`) is:

  ```
  local:0 local /usr/bin/X11/X
  ```

  This file can also be used to specify entries for X terminals that do not support XDMCP.

- `Xservers.fs`

The file is read by file server systems that use the `xdm-config.fs` configuration file. This file is used to specify entries for X terminals that do not support XDMCP.

- `Xsession`

  This initial startup script is executed under the user's UID to run the login session. If a `$HOME/.xsession` script is available, it runs. Otherwise, `xdm` runs the default session, `dxsession`, which is the DECwindows Session Manager.

- `Xsetup_0`

  This script is used only to configure the local X server; it cannot be used with remote X terminals. The script attempts to determine the display resolution and uses that information to set the appropriate font path. It also starts `dxconsole`, unless an alternate console is being used.

- `keymap_default`

  The `xdm` program links to the appropriate keymap file in `/usr/lib/X11/keymaps`.

- `xdm-config`

  This configuration file contains the values for a number of DisplayManager resources.

- `xdm-config.fs`

  This version of the `xdm_config` file is for use by server systems that do not have a local graphics display.

- `xdm-errors`

  This file is an error log file. Both `xdm` and the X server write all error messages to this file.

- `xdm-pid`

  Once this file records the `xdm` process ID, it is locked to prevent multiple invocations of `xdm`.

The following files are the default and alternate greeter modules. They are located in the `/usr/shlib/X11` directory:

```
libXdmDecGreet.so
libXdmGreet.so
```

The greeter module presents the login interface and authenticates the user and is dynamically loaded by `xdm`. The `libXdmDecGreet.so` file uses OSF/Motif style widgets. The `libXdmGreet.so` file uses Athena-style widgets.

The OSF/Motif style greeter is the default. It displays the Tru64 UNIX logo and login box. It uses the Security Integration Architecture (SIA) to provide improved security.

The Athena style greeter uses the standard X.Org graphical user interface (GUI). It does not use SIA; therefore, it cannot be used with enhanced C2 security.

1.4 Security and xdm Authorization

Because the X Window System runs in a networked environment, any other host on the network can access an individual workstation unless some kind of security mechanism is in place. The X Window System design makes it possible for any client that is able to connect to a workstation's X server to have complete control over that workstation's display. As a result, a client can take control of the mouse or keyboard, send keystrokes to any application running on the workstation, or kill windows in which other applications are running.

This section presents a summary of the X security environment.

There are two approaches to X Window System security: host-based security and user-based security. The following sections briefly discuss each type.

1.4.1 Host-Based Security

With host-based access control, only local clients are accepted by default. X Window System administrators can use the `/usr/bin/X11/xhost` client application to add or delete host names from a list of those allowed to connect to the X server. The `xhost` program uses host names to limit host connections. Therefore, there is no security among users on an individual host, only security among hosts.

Another host-based security mechanism involves using the `/etc/X`*n*`.hosts` file to list systems that can access the local server, specified by *n*. However, this method is not recommended because it is hard to maintain a truly limited list of hosts if more than one user has access to the workstation where the list resides and because the method allows access to the X server at any time, even when the `xdm` login window appears.

1.4.2 User-Based Security

Tru64 UNIX supports two types of user-based X access control authorization mechanisms: MIT-MAGIC-COOKIE-1 and XDM-AUTHORIZATION-1.

The `xdm-config` resources `DisplayManager.`*`DISPLAY`*`.authorize` and `DisplayManager.`*`DISPLAY`*`.authName` control whether `xdm` uses

authorization for local displays. X terminals using XDMCP negotiate with `xdm` to determine which mechanism to use.

When both the host workstation and the X server are configured to use MIT-MAGIC-COOKIE-1 or XDM_AUTHORIZATION-1, a machine-readable code is placed in the `~/.Xauthority` file in your home directory every time you log in under `xdm` control. The term used for this machine-readable code is the *magic cookie*. The X server is informed of the same *magic cookie* for the current session. The code is stored in a file in the `/usr/lib/X11/xdm/` directory, which the X server reads using its `–auth` capability.

Whenever a client application starts, it must supply the correct *magic cookie* code from the `~/.Xauthority` file to the X server to open the display. Since the permissions on this file are restricted to read/write for the user, only clients that the user starts have permission to read the *magic cookie* code. The assumption is that if the user starts the client application, he or she wants that application to have permission to run on the user's workstation.

You can use the `xauth` program to propagate the *magic cookie* code from one host to another. This feature allows users to run client applications on other workstations that do not share their home directory.

1.5 Solving X Window System Login Problems

This section describes some useful techniques for solving problems you might encounter when trying to log in to the X Window System environment. The first section discusses possible causes for login problems. The second section describes using the failsafe mode to correct various login problems.

1.5.1 Login Problems

If you cannot log in at all to your workstation, check for errors in the following places:

- `$HOME/.xsession-errors`

  This file contains errors generated by your own user account.

- `/usr/lib/X11/xdm-errors`

  This file contains `xdm` errors that are not limited to your own user account.

The following list describes the most common login problems and likely causes:

- After you enter your name and password in the login box, the screen immediately resets, redisplaying the login box.

  Possible causes for these problems are:

- There might be errors in the `$HOME/.xsession` script. For example, if there is an ampersand (&) on the last command line in the file, there would be no controlling process for the X session and the session would exit immediately.
- Your user disk and `/tmp` file system are both more than 100 percent full. You cannot log in because there is no space to write the `.Xauthority` file.
- If the `/usr/lib/X11/xdm/Xsession` script was customized, errors might have been introduced accidentally.

- You are able to log in, but only a single terminal window appears.

  The `Xsession` script has resorted to failsafe mode. Possible causes are as follows:

  - The user disk is more than 100 percent full.
  - The user's home directory is not writable.

- No login box appears on the local display and there is no X server.

  If you encounter this problem, check the `/usr/lib/X11/xdm-errors` file for error messages. Possible causes for these error messages are as follows:

  - There are problems in the `/usr/lib/X11/Xserver.conf` file.
  - There are problems in the `/usr/lib/X11/xdm/xdm-config` file or the files that it references.

- You cannot log in to an X display as root.

  For root login to an X display to be allowed, the name of the display must be listed in the `/etc/securettys` file for your workstation. If the display name is listed in that file, you can log in as root to the X display.

  The `/etc/securettys` file usually includes the entries `:0` and `local:0` to allow root login to the local display. You can add entries for remote X terminals or X displays so you can log in remotely as root from those machines.

1.5.2 Failsafe Mode

When you have problems logging in to your workstation, you can use failsafe mode to bring up a terminal window. You can use this terminal window to perform operations that can solve some of the login problems outlined in the previous section. Failsafe mode bypasses the `.xsession` script and `dxsession` session manager to display a single `dxterm` window. You can use this `dxterm` window to debug your `.xsession` script.

You can invoke the failsafe mode by pressing the F1 or F2 key after typing your password in the login box. Do not press the Return key.

If you are able to log in, but only a single terminal window appears, you are already in failsafe mode.

Once you are in failsafe mode, you can check the errors in the `xdm-errors` file; check for errors in the `xsession` script, `Xsession` file, `Xserver.conf` file, or `xdm-config` file; or delete files if the user disk is full.

1.6 Managing the X Server

The X server consists of loadable libraries. The dynamically loaded libraries are in the following directory:

```
/usr/shlib/X11
```

There are libraries for device support as well as others for X server extensions and font renderers. Section 3.1 provides descriptions of the X server extensions that the Tru64 UNIX operating system supports.

You specify which libraries you want dynamically loaded in the `/usr/var/X11/Xserver.conf` resource file.

On Tru64 UNIX systems, the X Window System programming extensions are built and dynamically loaded as shareable libraries. The X protocol requires that client applications must call the `XQueryExtension` function before using an extension. The `XQueryExtension` function returns extension information such as the base request number, number of requests, base error number, number of errors, and version string.

With this mechanism in place, the X server can defer loading any extension libraries until a client requests a specific extension. When the X server receives an `XQueryExtension` protocol request, it loads and initializes the appropriate extension library if that library has not previously been loaded.

In real time, this loading causes a slight delay in processing the first request for an extension library. However, no such delay is experienced during server start up. When the X server is shut down, it closes all libraries that were loaded on demand.

The extension library on Tru64 UNIX consists of the following shareable libraries. The first nine libraries (`ibxkb.so` through `libxinput.so`) are loaded at server startup time; the remaining libraries are loaded on demand.

- `libxkb.so`
- `liblbx.so`
- `liblbxutil.so`
- `libextshm.so`
- `libextAppgroup.so`

- libextSecurity.so
- libpanoramiX.so
- lib_dec_smt.so
- libxinput.so
- libextdpms.so
- libextMultibuf.so
- libextTOG_CUP.so
- libextshape.so
- libextMITMisc.so
- libextScrnSvr.so
- libextxtest.so
- libextkme.so
- libextSync.so
- libextXCMisc.so
- libextbigreq.so
- libextxtrap.so
- libdixie.so
- libmixie.so
- libdbe.so
- libPcl.so
- libPs.so
- libextXp.so
- libxv.so
- libprinter.so

Example 1–1 shows the default `Xserver.conf` resource file that Tru64 UNIX provides.

Example 1–1: Xserver.conf Resource File

```
! Default configuration file for extensible X server

! no other sysyem files are needed
! no other core files are needed
!

! device <
! >

! You can set alternate library search paths here or supplement the
! default path.
```

Example 1–1: Xserver.conf Resource File (cont.)

```
! library_path < /newserver/fonts/lib/font:/usr/shlib >

! Add a few more extensions
extensions <
        < extdpms       libextdpms.so   DPMSExtensionInit       DPMS >
        < dbe   libdbe.so       DbeExtensionInit        DOUBLE-BUFFER >
        < extshape      libextshape.so  ShapeExtensionInit      SHAPE   >
        < extMultibuf libextMultibuf.so MultibufferExtensionInit Multi-Buffering >
! The KME extension is obsolete functionality that supports lock down
! and latching modifiers.  It has been replaced by the XKB extension
! and is only provided here for interoperability with R5 servers.
        < extkme        libextkme.so    KMEInit Keyboard-Management-Extension >
        < extMITMisc libextMITMisc.so MITMiscExtensionInit MIT-SUND RY-NONSTANDARD >
        < extScrnSvr    libextScrnSvr.so        ScreenSaverExtensionInit
MIT-SCREEN-SAVER >
        < extSync       libextSync.so   SyncExtensionInit       SYNC >
        < extxtest      libextxtest.so  XTestExtensionInit      XTEST >
        < extbigreq     libextbigreq.so BigReqExtensionInit     BIG-REQUESTS >
        < extXCMisc     libextXCMisc.so XCMiscExtensionInit     XC-MISC >
        ! add the xtrap extension
        < extxtrap      libextxtrap.so  DEC_XTRAPInit   DEC-XTRAP >
        < extrecord     libextrecord.so RecordExtensionInit     RECORD >
        < EVI           libEVI.so       EVIExtensionInit
Extended-Visual-
Information >
        < TOG_CUP       libTOG_CUP.so   XcupExtensionInit       TOG-CUP >

        ! add the video extension along with device specific handlers
        ! for the TX device
        < xv    libxv.so        XvExtensionInit XVideo
                < _dec_xv_tx lib_dec_xv_tx.so XvropScreenInit PMAG-RO >
                < _dec_xv_tx lib_dec_xv_tx.so XvropScreenInit PMAG-JA >
        >

        ! add the X imaging extension
!not R6 < _dec_xie      lib_dec_xie.so  Xie3Init        Xie >
        < dixie         libdixie.so     XieInit         XIE
                < mixie libmixie.so >
        >

>

! Load Speedo and Type1 renderers and
! enable communication with a font server
font_renderers <
        < fr_fs         libfr_fs.so     fs_register_fpe_functions >
        < fr_Speedo     libfr_Speedo.so SpeedoRegisterFontFileFunctions >
        < fr_Type1      libfr_Type1.so  Type1RegisterFontFileFunctions >
>

! Enable X Input Devices
!input <
!       Dial and Button Box on port /dev/tty01
!       < _dec_xi_pcm   lib_dec_xi_pcm.so       XiPcmInit /dev/tty01 >
!
!       Serial Mouse.  Use the following format for the last parameter:
!
!       device:type:baud:emulate3:chordmid:samplerate:cleardtr:clearrts:core
!
!       where
!
!       device:         The port the device is connectd to.
!                       For example, /dev/tty00.  /dev/tty00 is the
!                       default.
!
!       type:           The mouse type.  It must be one of the following
!                       strings (case does not matter):  microsoft
!                       mousesystems mmseries logitech busmouse mouseman
!                       ps/2 mmhittab.  mousesystems is the default.
!
!       baud:           The baud rate of the mouse.  Mine is 1200, I think
```

Example 1–1: Xserver.conf Resource File (cont.)

```
!                       others will do 9600.  1200 is the default.
!
!       emulate3:       Either 0 or 1.  1 means emulate a 3 button mouse
!                       with a 2 button mouse.  This is not implemented yet,
!                       though 0 is the default.
!
!       chordmid:       Either 0 or 1.  Some 3 button mice treat MB2 kind of
!                       whacky.  A value of 1 supports those meesers.  0 is
!                       the default.
!
!       samplerate:     The sample rate of the mouse.  I don't have a mouse
!                       that supports this, so I don't know what it does.
!                       150 is the default.
!
!       cleardtr:       Either 0 or 1.  1 means clear the DTR signal for
!                       the port before using the mouse.  0 is the default.
!
!       clearrts:       Either 0 or 1.  1 means clear the RTS signal for
!                       the port before using the mouse.  0 is the default.
!
!       core:           Either 0 or 1.  1 means make this emulate the core
!                       device.  0 means make it a two relative motion
!                       valuator, n-button X Input Device.  0 isn't
!                       implemented yet.  1 is the default.
!
!        < _dec_xi_serial_mouse lib_dec_xi_serial_mouse.so XiSerialMouseInit
/de
v/tty00:microsoft:1200:1 >
! CalComp Tablet entry
!       < _dec_xi_db3   lib_dec_xi_db3.so       XiDb3Init
/dev/tty00:1:12:12:16:
1:8:1000:1:1 >
!>
! transport and auth_protocol library loading is not yet supported

! you specify command line arguments here
args <
        -pn
>
```

The following three `Xserver` command options are especially useful in managing the X server. For more complete information on the Tru64 UNIX `Xdec` and `Xserver` commands, see `Xdec`(1X).

- `–terminate`

  This option causes the server to exit rather than to reset.

  You can also achieve the same effect by setting the following `xdm` resources in the `xdm-config` file:

  – `DisplayManager._0.terminateServer:  true`

  – `DisplayManager.local._0.terminateServer:  true`

- `-edge_left` *`scr1 scr2`*

- `-edge_right` *`scr1 scr2`*

  You use one of these options to connect the edges of screens in a multihead display configuration.

- `-fp`*fontpath*

  You can use this option to set the default font path. The path consists of a comma-separated list of directories for the X server to search to find the font databases. Make sure that all components of the list are valid font directories or else the X server will exit.

The X server has been modified to automatically query the kernel for the language and keyboard of the console. Given this information, the X server will examine the contents of the `/usr/lib/X11/xkb/keymaps.dir` file to determine which default keymap to use. The X server will then compile the keymap, place the compiled version in the directory `/usr/lib/X11/xkb/compiled`, and load it. This feature has been enabled by default. To disable it, add the `-noloadxkb` switch to the command line. If you wish to disable the XKB extension altogether, add the `-kb` switch to the command line. See `Xdec`(1X) or run `Xdec` with the `-?` option for more information.

1.7 Graphics Adapters

This section lists many of the graphics adapters supported by Tru64 UNIX. (Absence of any adapter from this list does not necessarily mean that the adapter is not supported.) Some graphics adapters require their own support kits for full three-dimensional support.

PB2GA-AA QVision Triton EISA graphics adapter	PB2GA-AA Triton 72Hz 1024x768 Supported by `lib_dec_triton.so`
PB2GA-FA ATI Mach64 PCI VGA graphics adapter	PB2GA-FA ATI Mach64 PCI VGA graphics adapter PB2GA-FA Mach64 72Hz 800X600 PB2GA-FA Mach64 70Hz 1024X768 (Default Hz and resolution) PB2GA-FA Mach64 66Hz 1280X1024 PB2GA-FA Mach64 70Hz 1280X1024 PB2GA-FA Mach64 72Hz 1280X1024 Note that 1280X1024 is only available on cards with at least 2 MB RAM Supported by `lib_dec_ati64.so` or `lib_dec_ati64_linear.so`

PB2GA-FB ATI Mach64 ISA VGA graphics adapter	PB2GA-FB Mach64 72Hz 640X480
	PB2GA-FB Mach64 72Hz 800X600
	PB2GA-FB Mach64 70Hz 1024X768 (Default Hz and resolution)
	PB2GA-FB Mach64 66Hz 1280X1024
	PB2GA-FB Mach64 70Hz 1280X1024
	PB2GA-FB Mach64 72Hz 1280X1024
	Note that 1280X1024 is only available on cards with at least 2 MB RAM
	Supported by `lib_dec_ati64.so` or `lib_dec_ati64_linear.so`
Cirrus 5422 VGA graphics adapter (embedded on AlphaServer 1000)	Cirrus 5422 60Hz 640X480
	Cirrus 5422 56Hz 800x600
	Supported by `lib_dec_cirrus.so`
PB2GA-J S3 Trio64 VGA graphics adapter	PB2GA-J Trio64 60Hz 640X480
	PB2GA-J Trio64 72Hz 640X480
	PB2GA-J Trio64 60Hz 800X600
	PB2GA-J Trio64 72Hz 800X600
	PB2GA-J Trio64 60Hz 1024X768
	PB2GA-J Trio64 70Hz 1024X768 (default hz & resolution)
	PB2GA-J Trio64 72Hz 1024X768
	PB2GA-J Trio64 60Hz 1280X1024
	PB2GA-J Trio64 66Hz 1280X1024
	PB2GA-J Trio64 72Hz 1280X1024
	Note that 1280X1024 is only available on cards with at least 2 MB RAM
	Supported by `lib_dec_s3.so` or `lib_dec_s3_linear.so`
PBXGA-A HX+ 8-Plane Smart Frame Buffer Plus for PCI (SFB+)	PBXGA-A HX+ 72Hz 1280X1024

	Supported by `lib_dec_ffb.so`
PBXGA-B HX+ 24-Plane Smart Frame Buffer Plus for PCI with no Z-buffer(SFB+)	PBXGA-BA HX+ 72Hz 1280X1024
	Supported by `lib_dec_ffb.so` or `lib_dec_ffb_ev5.so`
PBXGA-C HX+ 24-Plane Smart Frame Buffer Plus for PCI with Z-buffer(SFB+)	PBXGA-CA HX+ 72Hz 1280X1024
	Supported by `lib_dec_ffb.so` or `lib_dec_ffb_ev5.so`
PBXGB-A TGA2 8 MB Smart Frame Buffer for PCI	PBXGB-AA TGA2 graphics adapter
	Supported by `lib_dec_ffb.so` or `lib_dec_ffb_ev5.so`
PBXGB-C TGA2 Smart Frame Buffer for PCI	PBXGB-CA TGA2 graphics adapter
	Supported by `lib_dec_ffb.so` or `lib_dec_ffb_ev5.so`
PBXGF-AB 3dlabs Oxygen VX1	Supported by `lib_dec_p3.so`
PBXGK-BB Elsa GLoria Synergy	Supported by `lib_dec_comet.so`

The following two families of boards have their own support kits:

Powerstorm 4D40T, 4D50T,4D60T, 4D51T (CatEyes series)	Supported by `lib_dec_e3.so`
Powerstorm 300, 350 (Peregrine series)	Supported by `lib_dec_ri.so`

1.8 Font Server Management

In Tru64 UNIX, `/usr/bin/X11/xfs` is the X Window System font server. The font server supplies fonts to the X Window System display servers.

For X11 R6, the font server was renamed from `fs` to `xfs`. For compatibility, the symbolic link`/usr/bin/X11/fs` —> `xfs` is provided on Tru64 UNIX. Most X11 R5 and X11 R6 X servers can communicate with a font server.

For Tru64 UNIX, the font server loads the following configuration file by default:

```
/var/X11/fs/config
```

Example 1–2 shows the default configuration file. Note that on the Tru64 UNIX system, the `catalogues` and `renderers` lines are not separated as shown in the example.

Example 1–2: Font Server config File

```
# font server configuration file
# $XConsortium: config.cpp,v 1.7 91/08/22 11:39:59 rws Exp $

clone-self = on
use-syslog = off
catalogue = /usr/lib/X11/fonts/decwin/100dpi/,
            /usr/lib/X11/fonts/decwin/75dpi/,
            /usr/lib/X11/fonts/misc/,
            /usr/lib/X11/fonts/75dpi/,
            /usr/lib/X11/fonts/100dpi/,
            /usr/var/X11/fonts/user/misc/,
            /usr/var/X11/fonts/user/100dpi/,
            /usr/var/X11/fonts/user/75dpi/
error-file = /usr/var/X11/fs/fs-errors
# in decipoints
default-point-size = 120
default-resolutions = 75,75,100,100
renderers = libfr_Type1.so;
            Type1RegisterFontFileFunctions,libfr_Speedo.so;
            SpeedoRegisterFontFileFunctions
```

The following list explains the elements in the file:

- `clone-self`

  This line indicates whether the font server should try to clone itself or use delegates when it reaches the limit for number of clients. By default, the Tru64 UNIX font server clones itself when the limit is reached.

- `use-syslog`

  This line indicates whether or not `syslog()` is used for font server error logging. For Tru64 UNIX, the value is set to `off`, which means that, by default, errors are logged to the `error-file` specified in this configuration file.

- `catalogue`

  This line contains the list of font directories that are available by default from the Tru64 UNIX font server.

- `error-file`

  This line lists the pathname of the error log file. This file is used instead of `syslog()`. If you encounter problems after you have modified the configuration file, check the `/usr/lib/X11/fs/fs-errors` log file to debug your changes.

- `default-point-size`

  This line indicates the default point size for any font request that does not specify a point size. Note that the point size is specified in decipoints, so that a value of 120 indicates a point size of 12.

- `default-resolutions`

  This line lists the default resolutions supported by the Tru64 UNIX font server. The values are given in pairs of horizontal and vertical resolutions per inch.

- `renderers`

  This line defines the dynamically loaded renderer libraries for scalable fonts. These renderer libraries are the same font renderer libraries that can be loaded by the X server.

1.8.1 Using the Font Server

To use the font server, you need to add the appropriate port to your font path. For Tru64 UNIX, the default port number is `7100`. The default port number is the registered port `7100`. (Note that many R5 implementations used port `7000`, which was not registered. Use the following syntax to add the font server to your font path:

xset +fp tcp/*hostname*:7100

Replace the *hostname* variable with the name of the system where the font server is running.

You can create a script that automatically starts the font server when you boot your system in multiuser mode. Add a symbolic link to your script in `/sbin/rc3.d`. For example:

```
/sbin/rc3.d/S94fs -> ../init.d/fs
```

For more details, see `rc3`(8).

The following example shows a sample font server initialization script:

```
#!/sbin/sh
PATH=/sbin:/usr/sbin:/usr/bin
export PATH
#
# Control X font server
#
```

```
case $1 in

\'start\')
    if [ -f /usr/bin/X11/xfs ]
    then
      /usr/bin/X11/fs -config /usr/lib/X11/fs/config -port 7100
&
    else
        echo "WARNING: Font server not found."
        exit 1
    fi
    ;;
\'restart\')

 $0 stop
 sleep 5
 $0 start
    ;;
\'stop\')
    pid=‘/bin/ps -e | grep ’/usr/bin/X11/fs’ |
        sed -e ’s/^  *//’ -e ’s/ .*//’ | head -1‘
    if [ "X$pid" != "X" ]
    then
        /bin/kill $pid
    fi
    ;;
esac
```

1.8.2 Font Server Client Utility Applications

Tru64 UNIX includes several font server client utilities: `fsinfo`, `fslsfonts`, `fstobdf`, and `showfont`. The following list shows how to invoke each utility and provides a brief description. See the reference page for each utility for more information.

- `fsinfo`

  The `fsinfo` utility displays information about an X font server. You can use it to examine the capabilities of the server currently running on your system. The display shows predefined values for various parameters that are used for communication between clients and the server. The display also lists the font catalogues and alternate servers that are available.

  The following example shows the default `fsinfo` display for a Tru64 UNIX system named `coffee`:

  ```
  % fsinfo –server tcp/coffee:7100
  name of server: tcp/coffee:7100
  version number: 2
  vendor string: Hewlett-Packard Company Tru64 UNIX V5.1B
  vendor release number: 6500
  ```

```
maximum request size: 16384 longwords (131072 bytes)
number of catalogues: 1
         all
Number of alternate servers: 0
number of extensions: 0
```

- `fslsfonts`

  You can use the `fslsfonts` utility to display a list of all the fonts served by the current font server. The following example shows a partial display for the default Tru64 UNIX font server on a system named `coffee`:

```
% fslsfonts –server tcp/coffee:7100
adobe-avantgarde-demi-i-normal--0-0-0-0-p-0-iso8859-1
adobe-avantgarde-demi-r-normal--0-0-0-0-p-0-iso8859-1
adobe-avantgarde-medium-i-normal--0-0-0-0-p-0-iso8859-1
adobe-avantgarde-medium-r-normal--0-0-0-0-p-0-iso8859-1
adobe-courier-bold-i-normal--0-0-0-0-p-0-iso8859-1
     .
     .
     .
```

  You can also use the `fslsfonts` utility to list the fonts that match a specified pattern. See `fslsfonts`(1X) for details.

- `fstobdf`

  The `fstobdf` utility reads a font from the font server and creates a bitmap distribution format (BDF) file on the standard output that can be used to re-create the font. You can use this utility to test font servers, debug font metrics, and reproduce lost BDF files. However, be careful to not violate any copyrights or licensing agreements that pertain to the fonts.

  The following command invokes the utility to create a BDF file for a bold font using the font server on system `coffee`:

```
% fstobdf –server tcp/coffee:7100 –fn "*bold*" > boldfont.bdf
```

- `showfont`

  You can use the `showfont` utility to display information about a particular font that is served by the current font server.

  Each of the following commands invokes the utility to display information about the Adobe Avantgarde Demi font available from the font server on system `coffee`:

```
% showfont –server tcp/coffee:7100 -fn\
"-adobe-avantgarde-demi-*-*--*-*-*-*-*-*-*-*"
% showfont –server tcp/coffee:7100 -fn\
"-adobe-avantgarde-demi-r-normal--0-0-0-0-p-0-iso8859-1"
```

1.9 Managing X Terminals

Like workstations, X terminals have monitors, pointers, and keyboards but otherwise they resemble dumb ASCII terminals because they need to be connected to a host computer to function. In most instances, the X terminal reads the X server program at boot time from the host system over the network. However, there are some X terminals that also have the X server built directly into the terminal's ROM.

For X terminals that have X11 R4, R5, or R6 installed, host systems use the X Display Manager (`xdm`) and the X Display Manager Control Protocol (XDMCP) to serve those terminals.

There are three types of XDMCP queries that an X11 R4, R5, or R6 terminal uses to connect to a host:

- Direct

  With a direct query, the X terminal requests a login from only one host. The `xdm` program on the host responds and displays the login window.

- Indirect

  With an indirect query, depending on the host's `Xaccess` file, `xdm` either forwards the query to another host or displays the chooser box, which contains a list of available host nodes. If the chooser box is displayed, the user selects a host. Next, the chooser client forwards the query to that host. In either case, the second host then displays the login window.

- Broadcast

  With a broadcast query, the X terminal requests a response from any `xdm` host on the subnet. The X terminal can either request a direct connection to the first `xdm` host that responds or collect responses for a period of time and offer the list to the user to select one.

Once the connection between the X terminal and the host has been made, the user has access to all the X Window System features that are available on the host system.

You specify access control for XDMCP connections to X terminals in the `/usr/lib/X11/xdm/Xaccess` file. This file is defined in the `xdm-config` file by the `DisplayManager.accessFile` resource. The following list contains examples of different types of connection queries:

- Direct or broadcast queries

```
# disallow direct/broadcast service for xtra
!xtra.lcs.mit.edu

# allow access from this particular display
mars.osf.org
```

```
# allow access from any display in LCS
*.lcs.dec.com
```

- Indirect queries

```
# define a macro, % HOSTS
%HOSTS expo.lcs.dec.com xenon.lcs.dec.com \
excess.lcs.dec.com kanga.lcs.dec.com

# force extract to contact xenon
extract.lcs.dec.com xenon.lcs.dec.com

# disallow indirect access from extra
!xtra.lcs.dec.com dummy

# all others get to choose
*.lcs.dec.com %HOSTS
```

- Indirect queries from the chooser

```
# offer a menu of these hosts
extract.lcs.dec.com CHOOSER %HOSTS

# offer a menu of all hosts
xtra.lcs.dec.com CHOOSER BROADCAST

# offer any host a menu of all hosts
* CHOOSER BROADCAST
```

Older X terminals with X11 R3 can be managed directly without XDMCP. To use these X terminals, you must include a specific entry in the `/var/X11/xdm/Xservers` file. For example, to manage an X terminal named `cream`, include the following line in the `Xservers` file:

```
cream:0 foreign
```

With such a connection, the `xdm` utility immediately displays a login window on the X terminal.

1.10 Memory Utilization by the X Server

Under normal operating conditions, the X server requires large amounts of memory. Once memory is allocated to the X server, it is never freed to the system. It can be reused, but never freed. This means that the X server memory allocation may increase dramatically at startup and then become fairly stable, unless you continue to start new and unique client applications without terminating any of the earlier applications.

2

Customizing the X Environment

With the Tru64 UNIX operating software, you can use resource definitions to customize and manage your workstation environment and certain elements of X Window System, OSF/Motif, and DECwindows applications that you are running. This chapter contains information about how to specify and modify these resource definitions.

2.1 Resource Definition Overview

The term resources file refers to characteristics of X Window System applications or applications built on X Window System technology. Resources values define aspects of the X display on a workstation and the window applications that run in the X Window System environment.

X resources are defined for display aspects of the Tru64 UNIX operating system itself as well as for all the X client applications that are part of the operating system. X applications that are installed on top of the operating system also have resource definitions. Resources characteristics include color specifications for various elements in a window display, presence of scroll bars for a window, location of windows on the desk top area, font used for text, and width of window borders.

Resource definitions are used in all applications based on the X Window System, such as `xterm`, `xclock`, and even the X Display Manager `xdm`. The Tru64 UNIX operating software provides default resource definitions for the X Window System. Users can modify some resource definitions to customize their workstation environment; for example, to set the colors and positions of windows.

2.1.1 Setting Resources

System administrators can set systemwide resources to provide a more uniform environment for the people working at the workstations or X terminals for which they are responsible. Programmers rely on resource specifications to create application windows, dialog boxes, and menus as well as to establish a particular look and feel for their applications' displays.

There are three ways to set resources:

- Using command-line options when invoking a particular client such as `dxterm`, `xterm`, or `xclock`

Only a subset of resources can be set from the command-line, but the advantage of this method is that you do not need to edit any files to apply the definitions. Section 2.2 discusses this method.

- Defining resources in files that are processed whenever an X client application starts

  These files include `$HOME/.Xdefaults-hostname` and files to which the `XENVIRONMENT` variable points. Resource definition files can be located in the user's home directory and in the `/usr/lib/X11/app-defaults` directory, which is part of the operating software.

  The system administrator can use systemwide files to establish uniform settings for small or large groups of users; or special individual settings. Section 2.3 discusses this method.

- Defining resources in client applications

  Programmers who are writing X Window System client applications include resource definitions in their code so that they control the look and feel of the application. Section 2.4 describes some utilities that help users and programmers specify resource definitions.

2.1.2 Resource Definition Precedence

Because of the variety of methods for setting resources, there could be times when there are several definitions for a particular resource. For X Window System environment resources, the definitions are applied in the following order:

1. Systemwide application default resource definitions

   Resource definitions for the Tru64 UNIX operating software clients are located in the `/usr/lib/X11/app-defaults/ClassName` files. These resources are used only by a client that runs on the local host, even if the client appears on a remote X display.

2. User-specific default resource definitions

   These definitions are usually located in files in the user's home directory, `$HOME/ClassName`. If several hosts share the home directory, the definitions in the directory will also be shared by those same hosts.

3. Host-specific default resource definitions

   Host-specific resource definitions are located in either the `$HOME/.Xdefaults-hostname` file or a file pointed to by the `$XENVIRONMENT` variable. These definitions are only used by applications running on the host system and are not specific to the display.

4. Resource database resource definitions

Some users use a resource database loaded by the X Server Resource Database utility (xrdb) to specify display-specific default resource settings.

5. .Xdefaults file resource definitions

   If no resource database exists for the user, the X server applies the resource definitions in the $HOME/.Xdefaults file.

6. Command-line options

   Users can change some resource definitions by specifying the new resource settings on the command-line when they invoke the client application. Section 2.2 and X(1X) provide information on the standard resources that can be set from the command line for most applications. Client applications can create additional options that set resource definitions that are specific to the particular application.

It is important to be aware of which resource definitions take precedence of other definitions; hence, the use of ascending numbers in the preceding list. System definitions are overridden by user definitions, which are, in turn, overridden by host-specific definition. A definition supplied through a command-line option overrides any existing definition for that resource. However, only the 17 standard resources or resources for which the client application has provided a command option can be defined using command-line options. Other resources must be specified in definition files or by using the –xrm option.

Note that host-specific and user-specific resource files do not necessarily have to reside in the user's home directory. There are several environment variables that can be set to specify a search path for default files:

- XFILESEARCHPATH

  This environment variable is used to set the path for systemwide application-specific resource definition files.

- XUSERFILESEARCHPATH

  This environment variable is used in place of $HOME for application-specific user resource definition files.

- XAPPLRESDIR

  If this environment variable is defined and XUSERFILESEARCHPATH is not, the search path becomes:

  ```
  $XAPPLRESDIR/%L/%N:$XAPPLRESDIR/%l/%N:$XAPPLRESDIR/%N:$HOME/%N
  ```

  $XAPPLRESDIR is replaced by the value of that environment variable; $HOME is replaced by the user's home directory. If there is no definition for $XAPPLRESDIR, the path is the user's home directory:

  ```
  $HOME/%L/%N:HOME/%l/%N:$HOME/%N
  ```

The %L element resolves to a full-locale name if one exists; %l resolves to the language component element of the locale; %N resolves to the name of the file being searched for. If no file exists in the locale or if no locale has been defined, the path collapses to the next level.

2.1.3 Loading Resource Definitions

Using xrdb to load resource definitions directly into the X server promotes consistency in the way applications run. In addition, because xrdb runs the resource definition file through a C preprocessor, you can further customize the environment by using #ifdef and #include commands in the resource definition files. You can also use the –D (define symbol) and –U (undefine symbol) options on the xrdb command line to set up different environments on different hosts; so users can move among workstations with different capabilities and maximize the special features on each one. (See Section 2.4.4 for more information.)

To load resources using xrdb, use either the –load option (the default) or the –merge option and specify a new resource definition file. With the –load options, all previous resource definitions in the X server are deleted and replaced with the new definitions in the specified file. If the new file does not contain a definition for a resource that was defined previously, that resource definition is either lost or reverts to a default. The –merge option allows you to change and add resource definitions without losing existing ones that you do not modify in the new definition file that you specify with the xrdb command.

2.2 Using Command-Line Options

When you invoke a client application on your workstation, you can use command-line options to specify certain characteristics for the appearance, location, and features of the window display. There are a number of standard options that are used with X Toolkit or Motif Toolkit applications. Not all such applications use all the standard resource options, but many use most of them. Programmers can also create application-specific options so that users can set other resources for those applications.

Table 2–1 lists the standard command-line options and the resources they modify.

Table 2–1: Standard Command-Line Options

Option	Resource	Description
–bg –background	background	Sets the background color of the window.
–bd –bordercolor	borderColor	Sets the color of the window border.
–bw –borderwidth	borderWidth	Sets the width of the window border in pixels.
–display	display	Specifies the display on which the client runs.
–fn –font	font	Sets the font used for text display.
–fg –foreground	foreground	Sets the window's foreground color that is used for the text or graphics.
–geometry	geometry	Specifies a geometry string that sets the startup size and placement of the window.
–iconic	iconic	Invokes the application in the iconic state.
–name	name	Specifies the name of the application. This name is used for the window icon.
–rv –reverse	reverseVideo	Reverses the foreground and background colors.
+rv	reverseVideo	Restores the foreground and background colors to their current specifications.
–selectionTimeout	selectionTimeout	Specifies the timeout period in milliseconds. This value determines the timeout period within which two communicating applications must respond to one another after a selection request.
–synchronous	synchronous	Enables synchronous debugging mode.
+synchronous	synchronous	Disables synchronous debugging mode.
–title	title	Specifies the application title that is used in the window's title bar.

Table 2–1: Standard Command-Line Options (cont.)

Option	Resource	Description
–xnllanguage	xnlLanguage	Sets the language, territory, and National Language Support codeset.
–xrm		Allows you to specify a resource name and value to override any defaults.

To modify an application resource definition, include the option on the command line that invokes the application. Most options require a parameter such as the name of a color, a file name, or a text string. The reference page for the command that invokes the application lists the appropriate options and their parameters.

The following examples show how some of these standard options are specified when an application is invoked:

```
dxterm –bg "pale green" –fg "sandy brown" &
```

This example starts a DECterm window with a pale green background. The text and graphics appear in sandy brown.

```
xterm –iconic  –name Letters &
```

This example creates an `xterm` window, but places it immediately in the icon state. The name of the icon is `Letters`.

```
dxcalc –geometry +0-0 &
```

This example invokes the DECwindows Calculator application and places the window in the lower left corner of the screen.

2.3 Using Resource Definitions

Resources are defined in several places in the X Window System environment. There are resource definition files such as local and groupwide `Xdefaults` files that contain resource definitions for your X workstation environment. Then there are resource definitions in window applications based on the X Toolkit (including DECwindows and OSF/Motif Toolkit applications) that determine the various visible aspects of the application.

Programmers need to understand resource definitions so they can use them when they create their applications. System administrators use resource definitions to set up a default working environment for the workstations they maintain. End users can use resource definitions to customize their workstation environment and even to customize some display characteristics of applications they run.

This section explains the structure of resource definitions, gives examples of how to create and modify the definitions, and describes the kinds of resource definition files that you can edit to customize your environment.

2.3.1 Resource Definition Structure

The syntax for resource definitions is as follows:

object.subobject[.subobject]... .attribute: *value*

The parameters have the following definitions:

object	The client program or a specific instance of the client program. This parameter can specify any client, such as a `DECterm` window or the clock application.
subobject	A *subobject* is an element of the *object* client program. A *subobject* corresponds to the widgets that make up the client program. The number of *subobject*s you need to include to reach the particular resource you want to specify is determined by the widget hierarchy of the client program.
attribute	This parameter specifies the characteristic that you want to define. The *attribute* must be a feature of the last *subobject* you listed. The *attribute* refers to such things as font, color, or location of the *subobject*.
value	This parameter specifies the definition for the *attribute*. Definitions can include color names, pixel coordinates, and Boolean values such as `True` or `False`.

Specifying the *object*, *attribute*, and *value* parameters is relatively straightforward. In general, the *object* parameter is the name of the client program. The resource *attribute* refers to the characteristic you want to modify, add, or delete. The second column in Table 2–1 contains the names of some resource attributes. The description gives you an idea of the kinds of values you can specify such as a color name for `foreground`, pixel coordinates for `geometry`, a font string for `font`, and a locale for `xnlLanguage`.

Creating resource definitions can be a bit more complex if you have to deal with *subobject*s. When you want a value to apply to an attribute throughout the application, you can use an asterisk (*) to indicate all the

*subobject*s. For example, if you want the background color to be light blue for every dialog box, menu, message box, and so on in the AccessX client, use the following resource definition:

```
accessx*background:   lightblue
```

This kind of definition is known as a loose binding because the value applies to all appropriate widgets in the hierarchy.

If you want to have a dark-blue background only for the status boxes, you would use the following resource definition:

```
accessx.mousekeys.statusbox:   darkblue
```

This definition requires that you know every element in the widget hierarchy from the main widget, `accessx` to the status box widgets. This kind of definition has a tight binding; that is, each subwidget between the `accessx` widget and the `statusbox` widget is listed in order, separated by periods.

To determine the elements in the widget hierarchy for an application, you need to use the `editres` utility. This utility creates a display of the hierarchy and also provides a way to test your resource definition. See Section 2.4.1 and `editres`(1X) for details.

2.3.2 Resource Definition Files

A resource definition file consists of lists of resource definitions and comments. Comments are prefixed by an exclamation point (!). You can use the exclamation point to disable a definition that you do not want to use, but want to retain in the file.

If your resource definition file will be run through the C language preprocessor, you can use `#ifdef` and `#endif` constructs to deal with definitions that are to be applied under certain circumstances. For example, you might have color definitions that would only be applied when you were working at a workstation with a color monitor.

The `/usr/lib/X11/app-defaults` directory contains resource definition files for many of the window client applications that are included with the Tru64 UNIX operating software. These files are read-only, so users cannot edit the contents to change or add resource definitions. However, you can use some of these definitions as models for your own definitions in a resource file or as part of the command line you issue to invoke the client. Note that many of these definitions specify things that you would not want to customize, such as the alignment of the buttons on the calculator application.

The files in the `/usr/lib/X11/app-defaults` directory do contain some resource definitions that you might find useful as models for definitions you

create. For example, you could use the `Clock-Color` file to get some ideas for color definitions.

More resource definition files are located in your `$HOME` directory. Some of these files can be specific to applications that you run. For example, you could create a file called `XTerm` in your `$HOME` directory that would be read every time a new XTerm window was created on the display. Any definitions for resources already defined in the `/usr/lib/X11/app-defaults/XTerm` file would be overridden by the definitions in your personal `XTerm` file.

Host-specific resource definition files customize your display environment and are read by all client applications running on your host. One such file is called `$HOME/.Xdefaults`. You can set colors for the display background and foreground as well as for various elements of the windows that appear. You use this file to specify your default window manager. If you usually work on a system with a color monitor, but occasionally use a monochrome monitor, you can include color definitions in your `.Xdefaults` file surrounded by `#ifdef` and `#endif` statements. These definitions will only be processed if the COLOR C preprocessor symbol is defined. If you use `dxsession`, the definitions in the `.Xdefaults` file will be loaded into the X servers resource database. Otherwise, you can use the `xrdb` utility, which automatically uses the C preprocessor to deal with such programming constructs. Note that `dxsession` does not use the C preprocessor to process the `.Xdefaults` file and `dxsession` only understands a limited number of C preprocessor directives. You can use the `xrdb -symbols` command to see which symbols `xrdb` has defined.

In general, you will want to use `xrdb` to load one or more resource definition files into the X server's database. The `xrdb` utility is usually invoked by a session script such as `$HOME/.xsession`. See Section 2.4.4 for more information on `xrdb`.

If you do not load the X server's resource database either by using `xrdb` or by using the Session Manager, each time an X application starts up, it reads the `.Xdefaults` file and applies all relevant resource definitions. The `dxsession` program processes the local `.Xdefaults` file and loads the resource definitions into the X server's database.

2.4 Using Client Utilities for Customization

The preceding section referred to some utilities that are useful in creating and processing resource definitions. The following sections describe each of the following client utilities that you can use to customize resources:

- `editres`
- `xset`
- `xsetroot`

- `xrdb`
- `xmodmap`

2.4.1 The editres Utility

The `editres` utility is a dynamic resource editor for use with X Toolkit applications. Motif applications are also X Toolkit applications and also work with `editres`. The utility allows users and application programmers to view the full widget hierarchy of any X Toolkit client that understands the `editres` protocol. You can use `editres` to apply resource definitions to an application and see the results immediately. Users can save these definitions by having `editres` append the definitions to an existing X resource definitions file such as `.Xdefaults`.

The `editres` utility displays the widget hierarchy along with the names and definitions of all the resources for a particular X client application. This information enables a user or programmer to add, modify, or delete resource definitions for the application. The `editres` utility can dynamically apply the resource changes to the application. Thus, the user or programmer can immediately see the results of the new definition and decide whether or not to save the change, restore the original setting, or make another change.

The `editres` main window has four areas: Menu Bar, Panner, Message Area, and Application Widget Tree display. You use the Menu Bar to access the different `editres` features. The Panner provides an intuitive method for scrolling through the Application Widget Tree display. The display area shows the widget tree for the application specified through the Get Widget Tree menu item.

The Show Resource Box menu item creates a pop-up window that contains resource definitions for the widget that is currently selected in the Application Widget Tree display.

You use the Set Resource pop-up window to enter a resource definition for all the widgets currently selected in the Application Widget Tree display. (You can use Tree menu commands to select more than one widget by specifying such keywords as All, Children, Parents, Descendents, or Ancestors; or by specifying a widget class.)

In most instances, you use the Resource Box to determine whether a resource has been defined and what that definition is. You also use this box to add, modify, or delete resource definitions and to indicate to which widgets these changes apply. Once you have made your change, you use the Apply button to see the effects of your change. Press the Save button to save the change you have made. There is also a Save And Apply button, which performs both operations at once.

Note that some client applications have hard-coded the attributes for certain elements rather than use resource definitions. There is no way for `editres` to modify hard-coded attribute specifications.

2.4.2 The xset Utility

The `xset` utility is described in its reference page as the "user preference utility for X". You can use this utility to set various user preference options for your workstation's display. These options include the following:

- Volume, pitch, and duration of the computer's beep sound
- Whether the keyclick sound is enabled or disabled and what volume it has
- Font path that specifies which fonts the X server can use
- Control of the use of LED lights for such things as Shift/Caps Lock
- Control of the mouse for such things as pointer acceleration and the length of the delay time until the maximum acceleration speed is reached
- Pixel color values
- Whether the autorepeat feature for keys is enabled or disabled
- Screen save parameter settings
- Enable and disable DPMS

You can use the –`q` option with the `xset` command to display the current settings for your workstation. To change a setting, issue the `xset` command with the appropriate option. See the `xset`(1X) reference page for the description of each option.

2.4.3 The xsetroot Utility

You can use the `xsetroot` utility to customize the attributes of the display background on your workstation. These attributes include the color and shape that the pointer cursor has, except in client windows where those settings have been defined by the client applications, and the pattern and colors of the display background; that is, the root window. You can use `xsetroot` to do such things as create plaid display backgrounds or change the shape of the pointer cursor to look like a hand or some other object.

The `xsetroot` command has a –`def` option that let you return the display to its default settings. See `xsetroot`(1X) for more details about the utility.

2.4.4 The xrdb Utility

The `xrdb` utility manages the X server resource database. This utility gets and sets the contents of the RESOURCE_MANAGER property for the display

window for screen 0 on your workstation, or the SCREEN_RESOURCES property for the display window of any or all screens. This utility is generally invoked from users' X session scripts. Resource definitions are loaded directly into the X server.

One of the features of the `xrdb` utility is that it uses a C preprocessor when it loads the resource definition file. This feature allows you to have `#include` and `#ifdef` statements and some other programming constructs in your resource definition files. In addition, you can define and undefine symbols by using the `–D` or `–U` options.

The following example shows how you might include an `#ifdef` directive in your resource definition file that defines the colors to use for DECterm windows on color workstation monitors and the black and white values to use with noncolor monitors:

```
#ifdef COLOR
DXterm*background:      lightblue
DXterm*foreground:      darkblue
#else
DXterm*background:      gray
DXterm*foreground:      black
#endif
```

You can use the `xrdb –query` command to see the current settings for your system. If you want to change some of these resources, you can create a resource definition file and use the `xrdb –merge filename` command to add or replace existing definitions with your changes. With the `–merge` option, `xrdb` replaces resource definitions for resources that are already defined for your system with those in the file you specify. If you have included resource definitions in that file for previously undefined resources, those new definitions are added. All other existing definitions remain the same.

There is also an `xrdb –load filename` command that you can use to erase all previous resource definitions and only use those in the file you specify. By default, `xrdb` behaves in this manner. Most of the time, you will probably want to use the `xrdb –merge filename` command because you will not want to lose the default settings for your environment.

For more information on the `xrdb` utility, see the *X Window System User's Guide* and `xrdb`(1X).

2.4.5 The xmodmap Utility

You can use the `xmodmap` utility to modify the mappings for keyboard keys as well as mouse buttons.

The utility has three basic mapping functions:

- It reassigns a modifier function to a different key on the keyboard. For example, to have the Right Shift key perform the Control modifier function, use the following command:

  ```
  xmodmap -e "Control_R = Shift_R"
  ```

- It reassigns a keyboard function to a different key on the keyboard. For example, to have the exclamation point (!) be sent to the computer when you press the vertical bar key, use the following command:

  ```
  xmodmap -e "keycode 243 = slash exclam"
  ```

- It reassigns pointer functions to different mouse buttons. For example, if you are left handed, you could use the following command to change the order of the buttons on the mouse from `1 2 3` to `3 2 1`:

  ```
  xmodmap -e "pointer = 3 2 1"
  ```

You can issue `xmodmap` commands during your work session or include them in an X session script. You can also create `xmodmap` definition files for the utility to read at startup time or when you invoke the utility during your work session.

The `xmodmap` command has the following syntax:

xmodmap [options] [filename]

When you use the `xmodmap` command with no options, it displays the current modifier key map, the keys that can be used to modify other keys. While this information can be helpful in some instances, most of the time you do not want to change these key mappings. The following example shows the `xmodmap` display:

```
xmodmap:  up to 2 keys per modifier, (keycodes in parentheses)

shift       Shift_R (0xab),  Shift_L (0xae)
lock        BadKey (0xb0)
control     BadKey (0xaf)
mod1        Multi_key (0xad),  Multi_key (0xb1)
mod2        Alt_L (0xac),  Alt_R (0xb2)
mod3
mod4
mod5
```

The items in the left column are the logical key names for the modifier keys. The items to the right are the `keysym` specifiers with the hardware hexadecimal keycode in parentheses. For example, the logical key name `shift` has two keys on the keyboard that perform the shift function. Their `keysyms` are `Shift_R` and `Shift_L`. The hardware hexadecimal keycodes for these keys are `0xab` and `0xae` respectively.

Using the xmodmap –pke command, you can see the decimal keycodes and the keysym name or names that have been assigned to each keycode. Note that keycode numbers vary depending on the keyboard model that you have connected to your workstation.

When there are two names, the second one indicates which key function is processed when the shift modifier key is pressed in combination with that physical key. The following example shows a portion of the output:

```
keycode 242 = semicolon colon
keycode 243 = slash question
keycode 244 =
keycode 245 = equal plus
keycode 246 = bracketright braceright
stdin
keycode 247 = backslash bar
keycode 248 =
keycode 249 = minus underscore
keycode 250 = bracketleft braceleft
keycode 251 = apostrophe quotedbl
```

You use keycodes and keysyms in the xmodmap –e command to modify the action that takes place when a particular keyboard key is pressed. For example, you can change the Select key on HP LK201/401 keyboards to perform the Delete function:

```
xmodmap -e "keysym Select = Delete"
```

You can have your own personal keymapping file by creating a file with xmodmap definitions, such as the following:

```
! Make the comma shift be < and the period shift be >.
keysym comma = comma less
keysym period = period greater
! Replace the Help key with the escape function.
keysym 124 = escape
```

To have the file processed whenever you log in, include an xmodmap command in your X session script. For example, if you named your key definition file .Xmodmap and located it in your home directory, you could include the following line in your X session script:

```
xmodmap $HOME/.Xmodmap
```

For more details about the xmodmap utility, see the *X Window System User's Guide* and xmodmap(1).

2.4.6 Utilities Using the X Keyboard Extension

Several applications that make use of XKB features are also new. These applications include the following:

- xkbcomp

  The xkbcomp utility is the XKB keymap compiler and converts XKB keymap source files into one of several output formats. It will also optionally load a keymap directly into the server if you specify the display as the output file. Each of the xmodmap keymaps located in /usr/lib/X11/keymaps for X11 R5 has been converted to XKB format for X11 R6. These new keymaps are located in/usr/lib/X11/xkb. See xkbcomp(1) or run xkbcomp with the -? switch for more information.

- xkbprint

  The xkbprint utility creates a PostScript representation of an XKB keymap. If you specify the display as the input file, it will read the XKB geometry from the server. See xkbprint(1) or run xkbprint with the -? switch for more information.

- xkbdfltmap

  The xkbdfltmap utility queries the kernel for the language and keyboard on the console. Given this information, Xdec will examine the /usr/lib/X11/xkb/keymaps.dir file to determine the default keymap to use. The xkbdfltmap utility will then display the appropriate xkbcomp command to run to download the default XKB map to the server. If xkbdfltmap is run with the -exec switch, it will automatically execute the xkbcomp command for you. See xkbdfltmap(1) or run xkbdfltmap with the -? switch for more information.

- dxkbledpanel

  The dxkbledpanel utility displays the state of the keyboard indicators. This is useful for monitoring and changing the state of indicators that may not have keyboard LEDs. For example, the group indicator does not always have an LED on every keyboard. See dxkbledpanel(1) or run dxkbledpanel with the -? switch for more information.

- dxkeyboard

  The dxkeyboard utility allows you to select a localized keymap based upon your selection of language and keyboard type. The dxkeyboard utility optionally saves your selections and will load them if it is run with the -load switch. The dxkeyboard utility is available as the Keyboard Options object under CDE's Application Manager in the Desktop_Apps folder. See dxkeyboard(1) or run dxkeyboard(1) with the -? switch for more information.

- accessx

  The accessx application for X11 R5 has been ported to use the XKB protocol for X11 R6. See accessx()1 for further information.

2.5 Using an X Session Script

Once you have decided on how you want to customize the X Window System on your workstation, an effective way to preserve that environment is to use an X session script. X session scripts also work with CDE's `dtlogin`(1) manager.

You can use a session script to invoke certain applications when you log in and place various windows on your display in specific positions. You can set the window manager in your session script as well as specify colors, fonts, and window features. The file can also contain `xmodmap` definitions or call an `xmodmap` definition file.

You can use a script to define certain environment variables before the session manager starts. For example, the following script defines the `PRINTER` environment variable, sets the default path, and invokes `dxsession` as the session manager.

```
#!/bin/csh
setenv $PRINTER ln08r
set path=($HOME/bin /bin /usr/bin /usr/bin/mh /usr/bin/X11 \
    /usr/local /usr/local/bin)
exec dxsession
```

The next example invokes the `xconsole` program and starts an `xterm` window as background processes. It then starts the `twm` window manager in the foreground. The `twm` window manager becomes the session's controlling process; that is, the session will last as long as the `twm` process is running. When `twm` exits, the `.xsession` script completes and the user's X session is over. If the last command line in the script had ended with an ampersand (&), the `.xsession` script would immediately complete and exit, the X session would be over, and `xdm` would cause the display to reset to the login box.

```
#!/bin/sh
xconsole –geometry 480x130–0–0 –daemon –notify –verbose
     –fn fixed –exitOnFail
xclock &
xterm –geometry 80x24+10+10 –ls &
exec twm
```

With the `xconsole` program running, messages that are usually sent to `/dev/console` appear in the `xconsole` window on the display. The `xclock` command places a clock client window on the display. The `xterm –geometry –ls` command starts an `xterm` window at the screen location specified with the `–geometry` option and starts the login shell in that window.

You can include a wide variety of customizations in an X session script as shown in the following example. The comments within the example explain the code.

Example 2–1: Session Script

```
#!/bin/csh
#
# Define environment variables, paths, and so on.  Keeping these
# definitions in a separate file is useful.  That way, .login
# and/or .cshrc can reference the same set of definitions.
#
source ~/.environ.csh
#
# Create a pipe for dxconsole to read from, so it can display the
# output of other commands.
#
setenv XSESSION_PIPE .xsession_pipe.$DISPLAY
if  ! { test -p .xsession_pipe.$DISPLAY } then
    /usr/sbin/mknod $XSESSION_PIPE  p
endif
#
# Use xrdb to load the resources in the .Xresources file into the
# X server's resource database.
#
if ( -f .Xresources ) then
    xrdb -load -retain .Xresources
endif
#
# Determine whether the display is the local graphics display,
# that is, :0 or local:0 .
#
if ( "`echo $DISPLAY | cut -d':' -f1`" ==  || \
    "`echo $DISPLAY | cut -d':' -f1`" == "local" ) then
    #
    # These applications are run only if the display is local.
    #
    dxconsole < $XSESSION_PIPE &
    #
    # Figure out how many screens the display has.
    #
    set SCREENS=`xdpyinfo | grep "number of screens" \
                    | cut -f 4- -d " "`
    #
    # The xset b flag sets the bell volume, pitch, and duration.
    # The xset c flag controls the key click.
    # The xset m flag controls the mouse acceleration and
    #               threshold.
    # The xset s flag sets the screen save parameters.
    #
    xset b 18 400 100 c 22 m 7 5 s 600 600 >& $XSESSION_PIPE
    #
    # For each screen, set the background color and the colors
    # and shape of the cursor.  This example uses custom colors
    # defined in an Xcms data file as well as customized bitmaps
    # (created # with /usr/bin/X11/bitmap) to define the shape
    # of the cursor.
    #
    set SCREEN=0
    while ( $SCREEN < $SCREENS )
      xsetroot -solid DarkBlueBackground -fg red -bg yellow  \
        -cursor cursor.bmp cursor_mask.bmp -display $DISPLAY.1 \
        >& $XSESSION_PIPE
```

Example 2–1: Session Script (cont.)

```
      @ SCREEN=($SCREEN + 1)
    end

    # Set the SCREEN variable to the screen number of the highest
    # numbered screen.
    #
    @ SCREEN=($SCREENS - 1)
    #
    # The xbiff command displays a small mailbox image that lets
    # you know when you have mail.  This example uses the 'letters'
    # bitmap from /usr/include/X11/bitmaps as well as custom
    # bitmaps for the full and empty bitmaps and shape masks.
    #
    xbiff -shape -update 120 -geometry 60x60-0+0 \
      -display $DISPLAY.0 -bg black -fg white \
      -bd '#191919195c5c' -xrm "XBiff*fullPixmap: letters" \
      -xrm "XBiff*emptyPixmap:  $HOME/bitmaps/one.xbm" \
      -xrm "XBiff*fullPixmapMask: $HOME/bitmaps/lettersmask.xbm" \
      -xrm "XBiff*emptyPixmapMask: $HOME/bitmaps/one.xbm" \
      >& $XSESSION_PIPE &
    # start oclock on screen 0
    oclock >& $XSESSION_PIPE &
    #
    # Start xcalendar, xload, and dxmail on the highest numbered
    # screen.
    #
    xcalendar -display $DISPLAY.$SCREEN >& $XSESSION_PIPE &
    xload -geometry +0-0 \
      -display $DISPLAY.$SCREEN >& $XSESSION_PIPE &
    dxmail -display $DISPLAY.$SCREEN >& $XSESSION_PIPE &
    #
    # Use xmodmap to reorder the mouse buttons and remap the Shift
    # Lock key on the LK401 or LK201 keyboard to be Escape.
    #
    xmodmap -e 'pointer = 2 3 1' >& $XSESSION_PIPE
    xmodmap -e "clear lock" >& $XSESSION_PIPE
    xmodmap -e "keycode 176 = Escape" >& $XSESSION_PIPE
    #
    # Start the Motif Window Manager as the controlling process.
    # When mwm exits, the X session will be over.
    # Using the shell's built-in exec command saves the cost of
    # creating another process.
    #
    exec mwm -multiscreen >& $XSESSION_PIPE
    #
    # End of Session
    #
else
    #
    # These applications are run only if the display is not local,
    # that is, the session is run on a remote X Terminal.
    #
    # Invoking dxconsole is useful for displaying the stdout of
    # the commands that run, even though as a remote display,
    # the display console output will not actually be displayed.
    #
    dxconsole < $XSESSION_PIPE &
    xset b 18 400 100 c 22 m 7 5 s 600 600 >& $XSESSION_PIPE
    xsetroot -solid DarkBlueBackground -fg red -bg yellow \
        -cursor cursor.bmp cursor_mask.bmp >& $XSESSION_PIPE
        oclock&
```

Example 2–1: Session Script (cont.)

```
    #
    # If the X Terminal is running its own local window manager,
    # mwm is likely to exit immediately, so it is not used as the
    # controlling process.
    #
    mwm >& $XSESSION_PIPE &
    #
    # Instead, xterm is used as the controlling process.  When
    # xterm exits, the X session will be over.
    #
    exec xterm
    #
    # End of Session
    #
endif
```

2.6 Bypassing the Login Manager

Although we do not recommend bypassing the `xdm` or `dtlogin`(1) login manager, there are several ways you could accomplish this. The following steps describe one method that you can use to disable the `xdm` or `dtlogin`(1) login manager:

1. Disable automatic startup of `xdm`.

   ```
   # mv /sbin/rc3.d/S95xlogin /sbin/rc3.d/xS95xlogin
   ```

2. Write a script that will start the X server and then start your application. For example:

   ```
   #!/bin/csh
   #
   # Start the X server.
   # Using the -ac option disables authentication checking.
   #
   /usr/bin/X11/X -ac &
   #
   # define anything you might need in your environment
   #
   setenv DISPLAY :0
   #
   # You may also configure the X server's font path, keyboard, etc.
   # by calling Xsetup_0.  This will also start dxconsole, but if
   # you don't want that make your own customized version of
   # Xsetup_0 and use that.  But bear in mind that the X server
   # will reset when its last connection is closed, so you may need
   # to hold open a connection, something like this:
   #
   /usr/bin/X11/xlogo&
   /var/X11/xdm/Xsetup_0
   #
   # Now start your application
   #
   /path-to-wherever/your-application &
   ```

3. Create a link to your script in `rc3.d` named `S95*`:

```
# ln -s /path-to-wherever/my-startup-script /sbin/rc3.d/S95whatever
```

4. For a clean shutdown, disable stopping of `xdm` or `dtlogin`(1):

```
# mv /sbin/rc0.d/K19xlogin /sbin/rc0.d/xK19xlogin
```

5. Write a shutdown script for your application and create a symbolic link to it in `/sbin/rc0.d/K19whatever`. This step is optional and only required if there is some cleanup you need to do in case the system is shut down.

3

Programming in the Tru64 UNIX X Window Environment

Use the *X Window System* and *X Window System Toolkit* manuals as the primary references for information on how to program X Window System applications.

However, information specific to the Tru64 UNIX X server is not covered in those manuals. This chapter includes information on the following topics:

- Extensions to the X server (Section 3.1)
- X Display Manager greeter module (Section 3.2)
- Programming update (Section 3.3)

3.1 Extensions to the X Server

Tru64 UNIX supports a number of protocol X server extensions. Many of these extensions are built and dynamically loaded as sharable libraries. Section 1.6 lists the components of the Tru64 UNIX extension library and explains the processes for loading and making calls to them.

The following list contains the X11 R6.5 protocol X server extensions that Tru64 UNIX supports:

- Application Group (Section 3.1.1)
- BIG_REQUESTS (Section 3.1.2)
- Display Power Management Signaling (DPMS) (Section 3.1.3)
- Extended Visual Information (EVI) (Section 3.1.4)
- Low Bandwidth X Extension (LBX) (Section 3.1.5)
- MIT-SCREEN-SAVER (Section 3.1.6)
- MIT Shared Memory (MIT-SHM) (Section 3.1.7)
- MIT-SUNDRY-NONSTANDARD (Section 3.1.8)
- Multibuffering (Section 3.1.9)
- OpenGL (Open Graphics Library) — Support available with the Open3D for Tru64 UNIX layered product (Section 3.1.10)
- PanoramiX (Section 3.1.11)

- Remote Execution (RX) (Section 3.1.12)
- Resource Configuration (Section 3.1.13)
- Security (Section 3.1.14)
- SHAPE (X11 Nonrectangular Window Shape) (Section 3.1.15)
- Shared Memory Transport (SMT) (Section 3.1.16)
- Synchronization Extension (SYNC) (Section 3.1.17)
- TOG-CUP (Section 3.1.18)
- XC-MISC (Section 3.1.19)
- X Imaging Extension (XIE) (Section 3.1.20)
- X Input Extension (Section 3.1.21)
- X Keyboard Extensions (`xkb`) (Section 3.1.22)
- X Keyboard Management Extension (XKME) (Section 3.1.23)
- X Print Extension (Xp) (Section 3.1.24)
- XTrap (Section 3.1.25)
- XTEST (Section 3.1.26)
- X Video (XV) (Section 3.1.27)

Documentation on many of the extensions is available from X.org. You can download the extensions from:

http://ftp.x.org/pub/R6.6/xc/doc/hardcopy/Xext.

Header files for several of the extensions are in the `/usr/include/X11/extensions` directory. The following sections provide brief descriptions of each extension.

3.1.1 Application Group

The Application Group extension provides the framework that allows more than one program to manage X applications on the desktop. Use of this extension allows embedding or inserting X programs into the windows of another program, such as a web browser.

An Application Group is a set of one or more applications that are managed by an application that is known as the Application Group Leader. The purpose of the Application Group is to share the `Substructure-Redirect` attribute of the `root` window with the application manager and one or more Application Group Leaders.

3.1.2 BIG_REQUESTS

The standard X protocol only allows requests up to 2^{18} bytes long. BIG_REQUESTS, a new protocol extension, has been added. This extension allows a client to extend the length field in protocol requests to be a 32–bit value. This is useful for extensions that transmit complex information to the server.

3.1.3 DPMS — Display Power Management Signaling

The Display Power Mangement Signaling extension supports powerdown capable monitors.

Not all graphics adapters and monitors are DPMS capable. It is important to check the equipment specifications of your monitors because monitors that do not support DPMS can be damaged by the activation of the DPMS feature.

The time required for a monitor to return from the power saver state depends on the amount of time the monitor has been in power saver state. This is the result of the cooling of the monitor phosphor and the time required to reheat the phosphor. It is not a function of the operating system or the X Window system.

3.1.4 EVI — Extended Visual Information

The Extended Visual Information extension allows a client to determine information about core X visuals, beyond those that the core protocol provides, by quering the X server for additional visual information, specifically for colormaps and framebuffer levels.

This extension exclusively supports X clients. It does not support X extensions. Extensions that have an impact on visual information should provide the mechanisms for delivering that information.

3.1.5 Low Bandwidth Extension

The Low Bandwidth X (LBX) extension defines compression and local caching techniques that improve performance of X applications in wide area networks (WANs) and across slow speed network connections. Performance is improved by reducing the amount of protocol data that is transported over the network and by reducing the number of client to server round trips required for common application start-up operations.

This extension is implemented using an X server extension and a proxy application. The X server extension provides a new optimized protocol. The proxy application, `lbxproxy`, translates a normal client X protocol stream into the LBX stream. This permits an existing application to benefit from the optimized protocol without any changes to the application.

The proxy is useful when multiple applications are running on a local area network that is separated from the X server by a slower network. In this case, the local cache is shared by each application using the same proxy process.

3.1.6 MIT-SCREEN-SAVER Extension

The Screen Saver extension enables a client to receive notification when the screen has been inactive for a specified amount of time or whenever it cycles. The extension is useful to those writing screensaver programs.

3.1.7 MIT-SHM — MIT Shared Memory Extension

This extension allows images to be placed in shared memory segments accessible by both the application and X server. Using shared memory reduces the amount of bandwidth required to transfer the images between the application and the server.

3.1.8 MIT-SUNDRY-NONSTANDARD Protocol Extension

This extension permits tolerance of old X bugs. See `xset`(1X) for a description of the `-bc` option.

3.1.9 Multibuffering Extension

This extension enables a client application to perform the following operations:

- Associate multiple image buffers with a window
- Paint in any image buffer associated with a window
- Display a series of image buffers in a window in rapid succession to achieve smooth animation
- Request simultaneous display of different image buffers in different windows

3.1.10 OpenGL — Open Graphics Library Extension

This extension provides a software interface to graphics hardware. The interface consists of a set of procedures and functions that allows a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

To the programmer, OpenGL is a set of commands that allows the specification of geometric objects in two or three dimensions, together with commands that control how these objects are rendered into the frame buffer.

For the most part, OpenGL provides an immediate-mode interface, so that specifying an object causes it to be drawn.

A typical program that uses OpenGL begins with calls to open a window in the frame buffer into which the program will draw. Then, calls are made to allocate a GL context and associate it with the window. Once a GL context is allocated, the programmer can issue OpenGL commands. Some commands are used to draw simple geometric objects for example, points, line segments, and polygons. Other commands affect the rendering of these primitives, including how they are lit or colored and how they are mapped from the user's two- or three-dimensional model space to the two-dimensional screen. OpenGL also has commands that affect direct control of the frame buffer, such as those that read and write pixels.

In the X Window System, OpenGL rendering is made available as an extension to X in the formal X sense: connection and authentication are accomplished with the normal X mechanisms. As with other X extensions, there is a defined network protocol for the OpenGL rendering commands that are encapsulated within the X byte stream.

Information on OpenGL is provided in the *OpenGL Reference Manual* eddited by Dave Shreiner (Addison-Wesley).

3.1.11 PanoramiX Extension (Xinerama)

The PanoramiX extension allows a system configured with multiple video monitors (a multiheaded system) to operate the monitors as a single large screen. Windows can span multiple screens and can move from one screen to another. This extension is only supported in homogeneous graphics environments. That is, the environment must consist of common devices, visuals, depths, resolutions, and so on.

Monitor configurations can easily be enhanced by enabling the PanoramiX extension in combination with use of the `-screenOrder` option, which allows screen ordering based on physical monitor location.

The Xnest and Xvbf servers are not configured to work with the PanoramiX extension. In addition, the OpenGL layered product is supported with the PanoramiX extension on the Peregrines (Powerstorm 300 and Powerstorm 350). This requires the installation of a separate kit.

The extension causes applications to display on multiple screens of a workstation as if the workstation is supporting only a single screen (`screen :0`). The size of the composite screen equals the size of the multiple screens combined.

It is necessary to create multiple instances of some resources because these resources are screen unique or they contain a back pointer to a `ScreenPtr`.

These resources include GCs, windows, pixmaps, or colormaps. When the server handles a client request that creates a resource, the extension creates an equivalent instance of the resource for each physical screen. PanoramiX uses linked lists to keep track of these resources. Each entry in the list contains the following information:

- The client-requested resource identifier
- Additional resource identifiers created by the extension
- A Boolean used for freeing entries

3.1.12 Remote Execution Extension (RX)

The Remote Execution (RX) extension defines a MIME-type document and defines how the document is used to execute a remote application from a Web Browser. The document is provided to the browser so that the browser can set up an environment for the application to run in. The RX document can list both required and optional services, and allows the preferences of the browser to determine which services to use. The RX plug-in is loaded as a default application helper for the Netscape Web Browser.

3.1.13 RCM — Resource Configuration Management

The Resource Configuration Management extension modifies a resource for a specific widget and each child widget in the hierarchy by changing the X Intrinsics. There is no sourcing of the resource file, the application does not have to be restarted for the new resource values to take effect, and the changes occur immediately.

The RCM customizing hooks reside in the Intrinsics and are linked with other toolkits such as Motif and the Athena widgets. This is the main difference between RCM and the Editres protocol.

The Resource Configuration Management extension is not a standard part of the X Toolkit Intrinsics (libXt).

3.1.14 Security Extension

The Security extension contains a new protocol that provides enhanced X server security. This extension adds to the X protocol the concepts of *trusted* and *untrusted* clients. The trust status of a client is determined by the authorization used at connection setup. All clients using host-based authorization are considered trusted. Clients using other authorization protocols may be either trusted or untrusted, depending on the data included in the connection authorization phase.

The requests in the security extension permit a trusted client to create multiple authorization entries for a single authorization protocol. Each entry

is tagged with the trust status to be associated with any client presenting that authorization.

When a connection identifying an untrusted client is accepted, the client is restricted from performing certain operations that would steal or modify data that is held by the server for trusted clients. An untrusted client performing a disallowed operation will receive protocol errors.

When a client is untrusted, the server will also limit the extensions that are available to the client. Each X protocol extension is responsible for defining what operations are permitted to untrusted clients. By default, the entire extension is hidden.

3.1.15 SHAPE – X11 Nonrectangular Window Shape Extension

This extension provides arbitrary window and border shapes within the X11 protocol. The `oclock` program, for example, uses this extension to produce a round clock-face display.

3.1.16 SMT – Shared Memory Transport Extension

The Shared Memory Transport (SMT) extension provides a completely shared memory transport for requests. For many operations, performance significantly increases when this extension is used. Unlike the MIT-SHM (shared memory transport) extension which supports only image transfers, the HP SMT supports the full protocol.

All requests are passed to the server by means of a shared memory queue. The server and client control the flow by using X protocol requests over UNIX Domain sockets. All events, replies, and errors are returned through UNIX Domain sockets.

This transport is suitable only for high-bandwidth applications that typically use large requests. Short requests may take longer to process with SMT than with UNIX Domain sockets because of synchronization overhead. For example, `XNoOp` requests will take twice as much time to execute.

The `DISPLAY` environment variable must be set to `local:0` when SMT is used.

When using SMT, the X server may not be able to allocate a shared memory segment. This problem occurs if the system shared memory resources are depleted; a warning message appears on the client side.

3.1.17 SYNC – Synchronization Extension

The synchronization extension, SYNC, provides primitive calls that allow synchronization between clients to take place within the X server. This

feature eliminates network errors that can arise when two communicating systems are running a distributed application that requires both systems to be synchronized.

With this extension, clients on different hosts running different operating systems can be synchronized. Multimedia applications can use this extension to synchronize audio, video, and graphics data streams. In addition, the extension provides internal timers within the X server that can be used to synchronize client requests. Using this feature, simple animation applications can be implemented without having to use round-trip requests. The extension allows applications to make the best use of buffering within the client, server, and network.

3.1.18 TOG-CUP

The TOG-CUP extension provides a mechanism for a colormap manager to recognize special colormap requirements, encourages colormap sharing and reduces colormap flashing on low-end 8–bit frame buffers, and defines a behavior in the X server color allocation scheme to reduce colormap flashing, when colormaps are not shared.

A protocol that provides a method to query the server for a list of reserved colormap entries, and one that initializes read-only (shareable) colormap entries at specific locations in a colormap encourage colormap sharing and accommodate special colormap requirements.

If the core protocol does not contain information about the pixel values returned, and the TOG-CUP extension is in effect, the AllocColor and AllocNamedColor request look in the default colormap for a matching color. If a match is found in the default colormap, and the corresponding cell in the private colormap is empty, the color is allocated to the corresponding location in the private colormap rather than the first available location. This minimizes colormap flashing when the root window's default visual is GrayScale, PseudoColor, or DirectColor, and the default visual is using a private colormap.

3.1.19 XC-MISC

The XC-MISC protocol allows clients to get back ID ranges from the server. `Xlib` handles this automatically, making this useful for long-running applications that use many IDs over their lifetime.

3.1.20 XIE — X Imaging Extension

The X Imaging extension provides mechanisms for the transfer and display of virtually any image on any X-capable hardware. Although this extension is not intended to serve as a general-purpose imaging processor, it provides a

large number of primitives for image rendering and image enhancement. These primitives can be combined to form complex expressions. XIE also includes facilities for importing and exporting images between clients and servers, facilities for moving images between client and servers as well as between core X modules and XIE modules, and facilities that enable applications to access images as resources.

X.Org provides documentation for XIE in PostScript format. That documentation is located on the Tru64 UNIX system in the `/usr/doc/xie` directory. The following list describes the documents:

- *X Image Extension Overview*

  This document provides general information about the X Image Extension code. Topics covered are: XIE design goals, XIE historical summary, XIE architecture, element definitions, and subsetting.

- *XIElib Specification*

  This document contains reference descriptions of all the XIElib functions, XIElib events, and XIElib errors. The Functions section covers the following types of functions: startup, LUT, photomap, ROI, photoflo, client data, abort and await, photoflo element, technique, and free.

- *XIE Sample Implementation Architecture*

  This document is for X.Org members who have a working understanding of the X Imaging Extension. It provides an architecture overview as well as chapters on the following topics: extension initialization, memory management, request dispatching, data representation, data structures, protocol requests, DIXIE photoflo management, DDXIE photoflo management, and photo elements.

- *X Image Extension Protocol Reference Manual, Version 5.0*

  This document specifies the X wire protocol for the X Image Extension. It defines the syntax, structure, and semantics of the XIE protocol elements. Topics covered include syntax specification, parameter types, resources, pipelined processing, import elements, process elements, export elements, events and errors, techniques, service class, and protocol encodings.

3.1.21 X Input Extension

This extension supports input devices other than the core X keyboard and pointer. The extension is designed to handle request and event definitions that are similar to core request and event definitions. This design allows extension input devices to be individually distinguishable from each other as well as from core input devices. The extension requests and events use a device identifier and support the reporting of *n*-dimensional motion data as well as other data that is not reportable through core input events.

3.1.22 X Keyboard Extension for X11 R6

The X Keyboard Extension (XKB) server extension 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 X11 R5 (for versions of Tru64 UNIX earlier than Version 4.0) AccessX server extension for people with physical impairments has been incorporated into the XKB server extension. These accessibility features include StickyKeys, SlowKeys, BounceKeys, MouseKeys, and ToggleKeys, as well as complete control over the autorepeat delay rate.

3.1.23 XKME — X Server Keyboard Management Extension

This extension enables an X client application to access the server mode-switch modifier. The mode switch is designed to meet the needs of character sets of languages that require native characters (for example, Hebrew and Japanese). The mode switch enables a client application to switch back and forth between character groups: Group 1 (ASCII characters) and Group 2 (native characters).

The function of the mode switch is similar to that of the Shift or Shift Lock key. These mechanisms both enable multiple symbols (`keysym`) to be generated from single keys, with one symbol for one mode or shift or shift/lock state and another symbol for the other state. For example, on the American keyboard, 3 and # can be switched by the shift state.

The combination of the mode-switch and shift/lock mechanisms allows up to four `keysyms` to be established for a single key.

The entry point `XKMEDoKBModeSwitch` is defined for the mode-switch modifier and can be set to the following modes of operation:

`LockDownModeSwitch`	Locks down the mode-switch modifier; that is, switches to Group 2.
`UnlockModeSwitch`	Unlocks the mode-switch modifier; that is, switches to Group 1.

See `dxkeycaps`(1X) for information on how to access the shift modifier from client applications and contains general information on keyboard mappings.

3.1.24 Xp (X Print Service Extension)

X Print (Xp) service is an X extension that allows X imaging to nondisplay devices such as printers and fax machines. The core of the X Print service is the X Print Server.

Applications that require printing operations can make a connection to the X Print Server and list the available printers using the `GetPrinterList` request. After selecting a printer, an application creates and sets a print context using the `PrintCreateContext` and `PrintSetContext` requests.

The print context is a fundamental X print service concept. The print context:

- Contains printer default capabilities
- Contains printer capabilities
- Maintains the state of the printer settings
- Maintains the state of rendering against a printer
- Maintains rendered output

A print context also affects how the DDX driver generates page description language (PDL), and how the PDL is submitted to the spooler. It may also affect fonts and other elements of the DDX layer of the X print server.

Printer capabilities are determined by attribute pools within the print context. These pools can contain information related to a context's server, printer, job, document, and page options. `PrintGetAttributes` and `PrintSetAttributes` are used to access and modify attribute pools.

`PrintStartJob` and `PrintEndJob` are used to delineate print jobs. A job is a collection of documents defined by `PrintStartDoc` and `PrintEndDoc`. Each document is a collection of pages. Upon completion of a job, the server sends any resulting PDL to a print spooler, or makes it available for retrieval by an application.

3.1.25 XTrap Extension

This extension allows a client application to track and use information about input events occurring on a remote X server. XTrap also allows a client application to provide input to the remote server.

3.1.26 XTEST Extension

This extension contains a minimal set of X client and server extensions that are required to completely test the X11 server with no user intervention. The extension is designed to meet the following goals:

- Minimize portability problems by confining the extension to an appropriate high level within the X server. In practice, this goal means that the extension should be at the DIX level, use the DIX/DDX interface, or both. This specification has effects, in particular, on the level at which *input synthesis* can occur.
- Minimize the changes required in the rest of the X server.
- Minimize the performance penalties that running the test produces on normal X server operation.

3.1.27 XV — X Video Extension

This extension performs the following functions:

- Lists available video adapters
- Identifies the number of ports each adapter supports
- Describes what drawable formats each adapter supports
- Describes what video encodings each adapter supports
- Displays video from a port to a drawable format
- Captures video from a drawable format to a port
- Reserves ports for exclusive use and unreserves them
- Sets and gets port attributes
- Delivers event notification

3.2 X Display Manager Greeter Module

In the X Display Manager (`xdm`), the greeter module is a separate dynamically loadable library. The greeter collects identifying information from the user (for example, name and password), authenticates the user, and optionally starts the login session. Application programmers can customize this module to suit the needs of their application.

The greeter library that is used is determined by the value of the `DisplayManager.greeterLib` resource in the `/var/X11/xdm/xdm-config` file. This library is required to define a function named `GreetUser()`.

The X Display Manager uses `dlopen()` to dynamically load the greeter library. It uses `dlsym()` to find the `GreetUser()` function.

The `GreetUser()` function can either handle the user's session itself or allow `xdm` to do so. The return value of `GreetUser()` indicates to `xdm` whether or not to start a session.

The GreetUser() function is passed the xdm struct display pointer, a pointer to a Display Struct (defined in /usr/include/X11/Xlib.h), and pointers to greet and verify structures. If GreetUser() expects xdm to run the session, it fills in the Display pointer and the fields of the greet and verify structs.

Definitions of struct display, struct verify_info, and struct greet_info are located in /usr/examples/xdm/dm.h. The GreetUser() function prototype is defined in /usr/examples/xdm/greet.h.

Any greeter library compiled on a Tru64 UNIX system prior to Version 4.0 must be recompiled to integrate data structure changes made in X11 R6. The use of a version field on these structs eliminates the need to recompile for future versions of the operating system.

The GreetUser() function is defined in greet.h as follows:

```
int GreetUser(
 struct display          *d,
 Display                 **dpy,
 struct verify_info      *verify,
 struct greet_info       *greet,
 struct dlfuncs          *dlfcns)
```

The parameters for the function are as follows:

- struct display *d [read-only]

  This struct display is defined in /usr/examples/xdm/dm.h.

- Display **dpy [write]

  The parameter returns the Display pointer from XtOpenDisplay() or XOpenDisplay().

- struct verify_info *verify [write]

  This struct is defined in /usr/examples/xdm/dm.h. The GreetUser() function is passed a pointer to an existing verify-info struct. The function is expected to write the fields of this struct. These fields include the uid, gid, arguments to run the session; the environment variable for the session; and the environment variable for startup and reset.

- struct greet_info *greet [write]

  This struct is defined in /usr/examples/xdm/dm.h. The GreetUser() function is passed a pointer to an existing verify-info struct. The function is expected to write the user's name and password into the name and password fields, but these values are really needed only when xdm is compiled with SECURE_RPC defined.

- struct dlfcns

This struct is a set of function pointers to xdm functions that the GreetUser ( ) function is likely to need.

Note that on Tru64 UNIX using these function pointers is not necessary since the symbols will be resolved by the dynamic loader.

The GreetUser() function returns an enumerated type, greet_user_rtn, defined in greet.h.

```
Greet_Session_Over     0    session managed and over
Greet_Success          1    greet succeeded, session not managed
Greet_Failure          -1   greet failed
```

3.3 Programming Updates

This section contains new information about programming in the X Window System environment. The section covers the following topics:

- XChangeProperty and GetWindowProperty functions (Section 3.3.1)
- Link order for static X clients (Section 3.3.2)
- DECnet transport for client/server connections (Section 3.3.3)

3.3.1 XChangeProperty and GetWindowProperty Functions

X.Org has refined the behavior of the XChangeProperty and GetWindowProperty functions. This refinement primarily affects programs that have arrays of integers (int) with format 32. If you have used or plan to use either function, use arrays of longwords (longs) instead.

Until recently, the data type used with a format of 32 was implied, not specified. With the new refinements, the data that is provided in format 32 to the XChangeProperty function or returned from the GetWindowProperty function should be accessed as arrays of longwords or typedefs based on longwords such as Window or Atom.

3.3.2 Link Order for Static X Clients

There are certain steps you must follow when compiling, loading, or linking X client applications against a static or nonshared library.

Specify either the –ldnet_stub or –ldnet option when:

- Using the cc –non_shared command
- Using the ld –non_shared command
- Linking against the libX11.a static library

If you link your X client application to the nonshared version of the /usr/lib/libDXm.a library, you must include libbkr in the link line.

If you omit `libbkr`, the warning messages appear about the following undefined symbols:

```
DXmHelpSystemClose
DXmHelpSystemDisplay
DXmHelpSystemOpen
```

3.3.3 DECnet Transport for X Client/Server Connections

The X server, X libraries, and various X clients use a DECnet transport mechanism for client/server connections when the appropriate DECnet product is installed on the system or on two systems, if the X client and X server are running on different nodes. If DECnet is not installed, attempts to make these client/server connections fail.

The loadable X server, as well as clients and libraries that directly execute calls to DECnet functions, are built using the `libdnet_stub.so` shared library in the `ld` command that links the object files. DECnet functions that are commonly called include `getnodename`, `dnet_addr`, and `dnet_conn`.

X clients that are built fully static and include `libX11.a` or `libXmu.a` must incorporate the `libdnet_stub.a` library if they do not use the DECnet transport. If they do use the DECnet transport, they must incorporate the `libdnet.a` library. One of these `libdnet` libraries must be included to resolve function calls from within the `libX11` or `libXmu` modules. If the X client is not fully static, but is using `libX11.a` or `libXmu.a` for some other reason, `libdnet_stub.so` should be included in the `ld` command information so that the client can be used whether or not DECnet is installed.

Note that DECnet/OSI is not part of the Tru64 UNIX operating system.

Index

F

G

H

I

K

L

M

N

O

P

R

S

T

W

X