UNICOS® X Window System Reference Manual

SR–2101 8.0

Cray Research, Inc.

Requests for copies of Cray Research, Inc. publications should be sent to the following address:

Cray Research, Inc.
Distribution Center
2360 Pilot Knob Road
Mendota Heights, MN 55120
USA

Order desk (612) 683–5907
Fax number (612) 452–0141

New Features

UNICOS X Window System Reference Manual SR–2101 8.0

The supported X Window system for UNICOS 8.0 is X11 Release 5; all references to X11R4 are removed. X11R5 features that Cray Research does not support include the following:

- X Window System server
- `xinit` and `xmh` clients
- Demo directories
- Example directories
- Shared memory extensions (`xshm`)
- Font server
- `MIT PEX` sample implementation
- `XDM` on UNICOS systems running MLS

Record of Revision

The date of printing or software version number is indicated in the footer. Changes in rewrites are noted by revision bars along the margin of the page.

Version	***Description***
6.0	November 1990. Original printing. This manual contains X Window System information for CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-2 systems running the 6.0 release of the UNICOS operating system.
7.0	September 1992. Reprint with revision to support the UNICOS 7.0 release.
7.0	This restock incorporates the corrected pages from the *UNICOS 7.0 Publications Errata*, publication ER–2124 7.0, which reflect the grave accent printing problem.
8.0	November 1993. Rewrite to support the UNICOS 8.0 release.

Preface

This publication documents the X Window System running on all Cray Research, Inc. systems with UNICOS release 8.0.

The *UNICOS X Window System Reference Manual* provides programming information on X Window System features that are specific to Cray Research, Inc. (CRI) applications. Readers should have a working knowledge of the X Window System, and either the UNICOS or UNIX operating system.

Related publications

The following CRI manuals provide information about the UNICOS operating system:

- *CVT 2.0 Release Notice and System Installation Bulletin*, publication R0–5070 2.0
- *UNICOS Visual Interfaces User's Guide*, publication SG–3094
- *UNICOS User Commands Reference Manual*, publication SR–2011
- *UNICOS Administrator Commands Reference Manual*, publication SR–2022
- *X Window System Resources Ready Reference*, publication SQ–2123

The following manuals provide information about the X Window System (the list includes ordering information):

- *X Window System – C Library and Protocol Reference*, by Robert W. Scheifler, James Gettys, and Ron Newman. Digital Press, Digital Equipment Corporation, 12 Crosby Drive, Bedford, MA 01730, order no. EY–6737E–DP.

- *X Window System – C Library and Protocol Reference*, by Robert W. Scheifler, James Gettys, and Ron Newman and documentation for X11R3 (together as one package). MIT Software Center, Technology Licensing Office, Room E32–300, 77 Massachusetts Avenue, Cambridge, MA 02139, 1–617–258–8330.

- *Xlib Programming Manual for Version 11 and Xlib Reference Manual for Version 11* (two-volume set), by Adrian Nye. O'Reilly and Associates, Inc., 1988, 981 Chestnut Street, Newton, MA 02164, 1–800–338–NUTS, email address, `uunet!ora!nuts`.

- *X Window System in a Nutshell*, edited by Daniel Gilly and Tim O'Reilly. O'Reilly and Associates, Inc., 981 Chestnut Street, Newton, MA 02164, 1–800–338–NUTS, email address, `uunet!ora!nuts`.

- *X Window System User's Guide for Version 11 (Volume 3)*, by Tim O'Reilly, Valerie Quercia, and Linda Lamb. O'Reilly and Associates, Inc., 1988, 981 Chestnut Street, Newton, MA 02164, 1–800–338–NUTS, email address, `uunet!ora!nuts`.

- *X Toolkit Intrinsics Programming Manual (Volume 4)*, by Adrian Nye and Tim O'Reilly. O'Reilly and Associates, Inc., 1990, 981 Chestnut Street, Newton, MA 02164, 1–800–338–NUTS, email address, `uunet!ora!nuts`.

- *X Toolkit Intrinsics Reference Manual (Volume 5)*, edited by Tim O'Reilly, introduction by Mark Langley. O'Reilly and Associates, Inc., 1990, 981 Chestnut Street, Newton, MA 02164, 1–800–338–NUTS, email address, `uunet!ora!nuts`.

- *Introduction to the X Window System*, by Oliver Jones. Prentice Hall Inc., P.O. Box 105361, Atlanta, GA 30348, 1–201–767–5937.

Conventions

The following conventions are used throughout this manual:

Convention	Meaning
`Courier`	This font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
italic	This typeface denotes variable entries and words or concepts being defined.
`bold Courier`	This font denotes literal items that the user enters in screen drawings of interactive sessions. Output is shown in nonbold Courier font.
[ ]	Brackets enclose optional portions of a command line.
...	Ellipses indicate that a preceding command-line parameter can be repeated.
$\boxed{\text{KEY}}$	Indicates a key on the keyboard.

The following machine naming conventions are used throughout this manual:

Term	Definition
CRAY Y-MP systems	All configurations of CRAY Y-MP systems supported by UNICOS 8.0, including the M90 series (M92, M94, M98); C90 series (C916, C92A, C94, C94A, and C98); E series (2E, 4E, 8E, and 8I); EL series (including CRAY Y-MP EL, CRAY EL92, CRAY EL98, and FAST-J).
CRAY X-MP systems	All configurations of CRAY X-MP systems supported by UNICOS 8.0. This includes CRAY X-MP, CRAY X-MP EA, and X-MP EMA systems.
CRAY-2 systems	All configurations of CRAY-2 systems supported by UNICOS 8.0.

Term	Definition
Cray MPP systems	All configurations of the CRAY T3D series, supported by UNICOS 8.0, including CRAY T3D MC, CRAY T3D MCA, and CRAY T3D SC.
All Cray Research systems	All configurations supported by UNICOS 8.0.

It is the objective of Cray Research to become compliant with IEEE Std 1003.1–1990 (POSIX.1) and IEEE Std 1003.2–1992 (POSIX.2). This manual reflects those ongoing efforts.

POSIX.2 uses *utility* to refer to executable programs that Cray Research documentation usually refers to as *commands*. Both terms appear in this document.

In this publication, *Cray Research*, *CRI*, and *Cray* refer to Cray Research, Inc. and/or its products.

Man page references

Throughout this document, reference is made to the online man pages available under UNICOS through the `man` command. A *man page* is a discussion of a particular element of the UNICOS operating system or a compatible product.

Each man page includes a general description of one or more commands, routines, system calls, or other topics, and provides details of their usage (command syntax, routine parameters, system call arguments, and so on). If more than one topic appears on a page, the entry in the printed manual is alphabetized under its primary name; online, secondary entry names are linked to these primary names. For example, `rc` is a secondary entry on the page with a primary entry name of `brc`. To access `rc` online, you can type `man rc`. To access information about `brc` online, you can type either `man rc` or `man brc`; both commands display the `brc` man page on your terminal.

Section numbers appear in parentheses after man page names. Man pages are referenced in text by entry name and section number, as shown in the following example:

> The `–p` and `–s` options to the `dmput`(1) command require that the caller be super user.

The following lists the type of entry associated with each section number:

Section	Subject
1	User commands
1B	User commands ported from BSD
2	System calls
3	Library routines, macros, and opdefs
4	Devices (special files)
4P	Protocols
5	File formats
7	Miscellaneous topics
7D	DWB-related information
8	Administrator commands

A routine name followed by an empty set of parentheses designates a kernel routine; for example, `ddcntl`(). These routines do not have man pages associated with them.

Printed man pages are published in Cray Research manuals. The following manuals consist of collections of man pages that describe the UNICOS operating system commands, system calls, and file formats:

Publication	Title
SR–2011	*UNICOS User Commands Reference Manual*
SR–2012	*UNICOS System Calls Reference Manual*
SR–2014	*UNICOS File Formats and Special Files Reference Manual*
SR–2022	*UNICOS Administrator Commands Reference Manual*

The *UNICOS User Commands Ready Reference*, publication SQ–2056, accompanies the *UNICOS User Commands Reference Manual*.

The following manuals contain collections of man pages that describe the UNICOS library routines:

Publication	Title
SR–2079	*UNICOS Fortran Library Reference Manual*
SR–2080	*UNICOS C Library Reference Manual*
SR–2081	*Scientific Libraries Reference Manual*
SR–2138	*Math Library Reference Manual*

In some cases, man pages associated with a given product are published in the documentation set for that product, rather than in the UNICOS manuals listed here. For more information about the availability and content of any Cray Research publication, see the *User Publications Catalog*, publication CP–0099.

Ordering publications

The *User Publications Catalog*, publication CP–0099, lists all Cray Research hardware and software manuals that are available to customers.

To order a manual, either call the Distribution Center in Mendota Heights, Minnesota, at (612) 683–5907 or send a facsimile of your request to fax number (612) 452–0141. Cray Research employees may choose to send electronic mail to `order.desk` (UNIX system users) or `order desk` (HPDesk users).

Reader comments

If you have comments about the technical accuracy, content, or organization of this manual, please tell us. You can contact us in any of the following ways:

- Send us electronic mail from a UNICOS or UNIX system, using the following UUCP address:

  ```
  uunet!cray!publications
  ```

- Send us electronic mail from any system connected to Internet, using the following Internet addresses:

  `pubs2101@timbuk.cray.com` (comments specific to this manual)

  `publications@timbuk.cray.com` (general comments)

- Contact your Cray Research representative and ask that a Software Problem Report (SPR) be filed. Use `PUBLICATIONS` for the group name, `PUBS` for the command, and `NO-LICENSE` for the release name.

- Call our Software Information Services department in Eagan, Minnesota, through the Technical Support Center, using either of the following numbers:

  (800) 950–2729 (toll free from the United States and Canada)

  (612) 683–5600

- Send a facsimile of your comments to the attention of "Software Information Services" in Eagan, Minnesota, at fax number (612) 683–5599.

- Use the postage-paid Reader's Comment Form at the back of this manual.

We value your comments and will respond to them promptly.

Contents

Introduction [1]

The X Window System is a portable, network-transparent window system originally developed at the Massachusetts Institute of Technology (MIT) and now an industry standard under control of the X Consortium. The X Window System, also known simply as X, allows separate applications (known as *clients*) that run on one or more hosts to be displayed on the same workstation screen in separate windows. The workstation does this by running an X server. To run X, the workstation must provide a bit-mapped display with a keyboard and pointing device (such as a mouse). If clients will run on a Cray Research system, TCP/IP access must be available between the workstation and Cray Research systems.

Because X is portable, it can be used on a wide variety of workstation hardware (for example, Sun, DEC, IBM, RT, Apollo, HP, Sony, and Apple). An important byproduct of this portability is a standardized method of writing graphics programs independent of hardware or operating system constraints. The low-level C language programming interface to X, called *Xlib*, and the high-level interface, called *Toolkit* (`Xt` library) and the *Athena Widgets* (`Xaw` library), are also standardized.

X achieves network transparency because it uses its own protocol to communicate between the client and the display server. Most Xlib functions generate some type of protocol requests. The X protocol requires a reliable end-to-end byte stream, which TCP/IP currently provides as the underlying network protocol.

This manual describes aspects of version 11, release 5 (X11R5) of the X Window System that are related specifically to Cray Research applications. Servers with earlier versions of X11 might still work with Cray Research X clients, but it cannot be guaranteed.

"Getting Started," page 3, provides information that enables users to get started using X on a Cray Research system. The rest of the manual provides programming information. "Writing Your Own Clients," page 7, offers programming techniques to ensure efficient use of the system. “Fortran and X,” page 19, describes the process for mixing Fortran and C functions in one binary file. “Cray Research Clients,” page 25, provides a list of the clients that were written to run on Cray Research systems.

Getting Started [2]

This section is for new X Window System users; it describes the following procedures:

- Accessing X on your workstation
- Logging in to a Cray Research system from your workstation
- Setting environment variables
- Running a client on the Cray Research system and displaying the output on your X workstation

The general methods presented in this section apply to all workstations. If you are following along at your workstation, the prompt you see will be one that is unique to your workstation. In the sample sessions that follow, the workstation is a Sun X11 server named `mirror`, and the prompt is `mirror%`. Similarly, in the sample session, the name of the Cray Research system is `sn2003`.

Accessing X on your workstation
2.1

To access X on your workstation, you must access the server by adding a path to the X11 binary files in your `.profile` (for the Bourne shell and Korn shell), `.login` (for the C shell), or `.cshrc` file (for the C shell), as follows:

```
set path=($path /usr/bin/X11)          (C shell)

PATH=$PATH:/usr/bin/X11                (Bourne shell,
export PATH                             Korn shell)
```

You can also do this from the command line.

Next, from your `.profile`, `.login`, or `.cshrc` file, or from the command line (if you are not already in a window system), set the `DISPLAY` environment variable to your server name, as follows:

```
mirror% setenv DISPLAY mirror:0.0    (C shell)

mirror$ DISPLAY=mirror:0.0 (Bourne shell,
mirror$ export DISPLAY                Korn shell)
```

To start your environment, create a `.xinitrc` file. The following is a sample `.xinitrc` file that contains commands to start some of the tools that you will be using. This sample shows `xclock`, which displays time; `xload`, which displays the system load average; `xbiff`, which is a mail notification program; and `xterm`, which is a terminal emulator. It starts the `twm` (tab window manager, formerly known as Tom's window manager), as an example, but other window managers such as `mwm` may be substituted.

```
xhost + sn2003
#
# note: The preceding item enables access for host sn2003
#
twm &
xclock –analog –geometry 64x64–0–0 &
xload –rv –geometry 150x64–64–0  &
xbiff –rv –geometry 64x64–214–0 &
xterm –geometry 80x60–0+0 –fn 8x13 &
#
# note: the last item must not be run in background.
#
xterm –C –g 51x12+0+0
```

You can customize your X environment by changing your standard dot files and adding some new dot files. "Example Files," page 59, contains examples of all of the files you will need.

To bring up the X11 server, add the following entry in your `.login` or `.profile` file:

```
xinit
```

When you log in or execute this entry at the command line, you should see a terminal window in the upper left corner of the screen and additional windows, as specified in your `.xinitrc` file (see the previous sample file).

In addition to using the `xhost`(1X) program, you can use X authority files (`xauth`(1X)) to control access to your display. With authority files, you must create a key, which clients must use to access your display server.

If your workstation is not a UNIX workstation, see the X documentation that is specific to your workstation for instructions on starting your environment.

Logging in to a Cray Research system from your workstation
2.2

In the following sample session, you will be logging in to a Cray Research system (`sn2003`) running UNICOS, to run the client named `xclock`. You will be displaying the clock on your workstation, `mirror`. To begin a remote login to the Cray Research system, use either `telnet`(1B) or `rlogin`(1B) in an `xterm` window, as follows:

```
mirror% telnet sn2003

mirror% rlogin sn2003
```

After you have provided a user ID and password, and standard login messages have appeared, a Cray prompt similar to the following appears:

```
sn2003%
```

Setting environment variables
2.3

Now that you are logged in to a Cray Research system, you must set the environment variables by entering the commands shown in this subsection. You can enter these commands at the prompt, or put them into your `.profile`, `.login`, or `.cshrc` file.

The following commands put `/usr/bin/X11` in your directory search path:

```
sn2003% set path=($path /usr/bin/X11)  (C shell)

sn2003$ PATH=$PATH:/usr/bin/X11       (Bourne shell,
sn2003$ export PATH                    Korn shell)
```

The following command sets the `DISPLAY` environment variable to your server name (`mirror`):

```
sn2003% setenv DISPLAY mirror:0.0    (C shell)

sn2003$ DISPLAY=mirror:0.0           (Bourne shell,
sn2003$ export DISPLAY                Korn shell)
```

The following command sets the terminal type environment variable, `TERM`, to `xterm`:

```
sn2003% set term xterm               (C shell)

sn2003$ TERM=xterm                   (Bourne shell,
sn2003$ export TERM                   Korn shell)
```

Running a client on the Cray Research system 2.4

Now you are ready to run programs (known as *clients*) on the Cray Research system. The following example illustrates the use of a Cray Research system to produce graphics interactively on a server workstation. Run the `xclock` client on the Cray Research system, as follows:

```
    sn2003% xclock &
```

A clock is displayed on your workstation. See `xclock`(1X) for available options; for more details about `xclock`, see the O'Reilly, volume 3, documentation, listed in the preface.

Writing Your Own Clients [3]

This section describes the unique features of X programming on the Cray Research system, and it offers programming techniques to make the most efficient use of the Cray Research system. See the preface for a list of references that describe general X programming techniques.

The following topics are presented in this section:

- “Compiling a client” shows how to write a client.
- “Handling events,” page 10, describes how to minimize network traffic, thus maximizing efficiency.
- “Using colors,” page 11, discusses issues that pertain to color graphics.
- “Using fonts,” page 14, describes techniques for efficient use of fonts.
- “Using images,” page 15, discusses the use of client-side raster images.
- “Debugging tools,” page 17, describes the use of the `cdbx`(1) and `xscope` debugging tools.

Compiling a client 3.1

When you build an X client on a Cray Research system, you must load the correct X libraries with the user-written code to create the executable binary file. The X11R5 libraries are as follows:

`libXext.a`	(Extension library)
`libXaw.a`	(Athena widget library)
`libXt.a`	(Intrinsics toolkit)
`libX11.a`	(also known as `Xlib`)
`libXmu.a`	(MIT utility library)

Writing Your Own Clients [3]

This section describes the unique features of X programming on the Cray Research system, and it offers programming techniques to make the most efficient use of the Cray Research system. See the preface for a list of references that describe general X programming techniques.

The following topics are presented in this section:

- "Compiling a client" shows how to write a client.
- "Handling events," page 10, describes how to minimize network traffic, thus maximizing efficiency.
- "Using colors," page 11, discusses issues that pertain to color graphics.
- "Using fonts," page 14, describes techniques for efficient use of fonts.
- "Using images," page 15, discusses the use of client-side raster images.
- "Debugging tools," page 17, describes the use of the `cdbx`(1) and `xscope` debugging tools.

Compiling a client
3.1

When you build an X client on a Cray Research system, you must load the correct X libraries with the user-written code to create the executable binary file. The X11R5 libraries are as follows:

`libXext.a`	(Extension library)
`libXaw.a`	(Athena widget library)
`libXt.a`	(Intrinsics toolkit)
`libX11.a`	(also known as `Xlib`)
`libXmu.a`	(MIT utility library)

`libXau.a`	(sample authorization protocol for X)
`libXdmcp.a`	(X display manager control protocol library)
`libXi.a`	(`xinput` extension library)
`liboldX.a`	(X10 compatibility library)

Figure 1 shows the relationship of the Xlib, toolkit, widget, and extension libraries to each other.

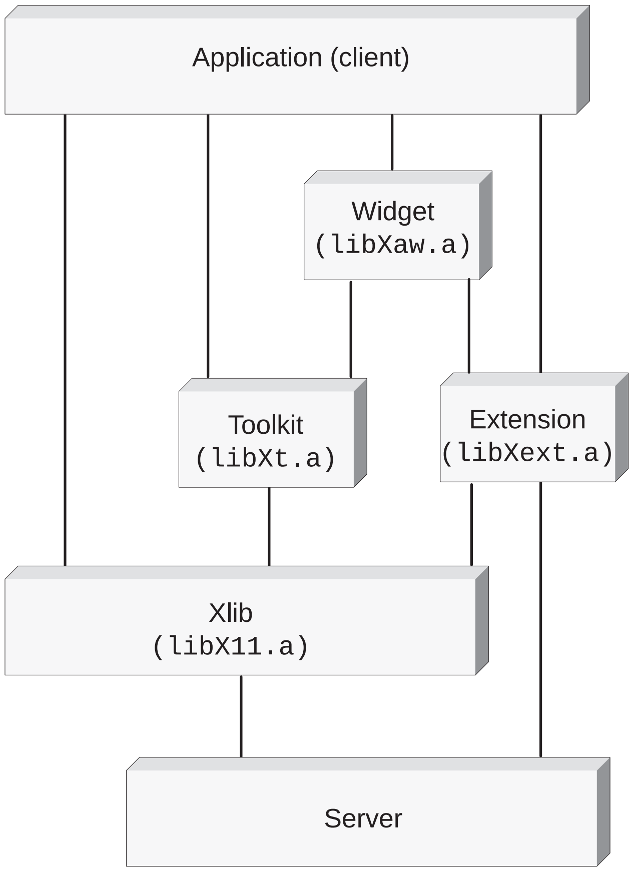

Figure 1. X11R5 libraries

You can use these same libraries when building a client on a workstation.

For a simple X client, you can use the following makefile:

```
# Set CFLAGS = –g for use with debugger
CFLAGS = –g  –DSYSV –DUSG
xclient: xclient.c
cc $(CFLAGS) –o $@ xclient.c \n
    –lXaw –lXt –lXmu –lX11 –lXext
```

Notice that SYSV and USG are defined. This is always a good practice when you are compiling applications that use X on the Cray system. If you are using an Imakefile, the define process is performed automatically. The imake(1) command takes an architecture-independent segment of a makefile and adds the architecture-dependent items to it. This provides an easy way to write a portable program.

If the resulting binary file is used with the debugger cdbx(1), you must use the –g option to cdbx to run the compile and load steps.

You can use the imake(1X) or xmkmf(1X) command to generate makefiles from templates known as *Imakefiles*. The following example shows a simple Imakefile:

```
DEPLIBS = $(DEPXLIB)
LOCAL_LIBRARIES = $(XLIB)
SimpleProgramTarget (test)
```

You can use either of the following commands to create a makefile from an Imakefile:

```
xmkmf
```

```
imake –DUseInstalled –DCURDIR=
```

Imakefiles released by Cray Research with UNICOS 8.0 have been upgraded to work with the Standard C preprocessor, usually located in /lib/cpp. Imakefiles obtained from other locations may still use the old format for comments and concatenation and therefore require the portable C preprocessor (pcpp). This preprocessor is located in /lib/pcpp on the UNICOS 8.0 operating system.

To designate a preprocessor other than the default (`/lib/cpp`) for either `imake` or `xmkmf`, set the `IMAKECPP` environment variable as follows:

```
setenv IMAKECPP /lib/pcpp          (C shell)

IMAKECPP=/lib/pcpp;                (Bourne shell,
export IMAKECPP                     Korn shell)
```

Handling events
3.2

The display server generates a packet of information for each occurrence of a user action, such as a keystroke, button press, mouse motion, window exposure, and so on, and each occurrence of interaction among programs. Each packet of information, called an *event*, is put into a protocol packet that consists of 32 bytes and is sent to each client that has requested that specific type of event. Each X protocol packet is also put into a TCP/IP packet, thus increasing the total number of bytes transferred. A client can request a specific type of event to track device input, another type of event to get mouse input, and so on. A client responds to events to control the user interface and to control communication among various clients. The processing that occurs when events are being controlled is known as *event handling*.

Keyboard input can generate a lot of network traffic. Each keystroke generates events `KeyPress` and `KeyRelease`, which typically cause a client to tell the server which character to display. The typing speed of individual users limits the network traffic, which is well below the network bandwidth. However, too many users typing into X clients on a Cray Research system can stress the communication resources.

Mouse motion is more stressful on network and CPU resources than keyboard input. Dragging the mouse across a window can generate numerous `MotionNotify` events. The server bundles together several `MotionNotify` events in one transmission to the client. The receiving client can compress `MotionNotify` events to appear as one event; however, because resources are wasted in generating, transmitting, and discarding them, a client should request `MotionNotify` events only when necessary. Alternatively, you can choose the

PointerMotionHints option, in which the server does the compression for you. Motion events are generated only under special circumstances, reducing the amount of data sent across the network.

The Xlib library provides the XSelectInput call, a mechanism that enables a client to choose the events it wants to see. One of the parameters of XSelectInput specifies the events that the server should generate for this client. Proper use of XSelectInput is the key to reducing unnecessary traffic between client and server. The following example shows the use of XSelectInput, in which keyboard input does not generate events, but mouse button activity does:

```
XSelectInput(display, window,
   ButtonPressMask | ButtonReleaseMask);
```

The following example shows a method for compressing mouse motion events in the client:

```
XEvent report;
while (XCheckTypedEvent(display, MotionNotify, &report));
```

XCheckTypedEvent reads the next MotionNotify event from the queue into the XEvent structure called report. If the event was a MotionNotify event, XCheckTypedEvent returns TRUE. The while loop discards all but the last MotionNotify event. This can be a great traffic reduction if the client has elected to see MotionNotify events.

Using colors
3.3

The visual impact of color allows you to convey a lot of information in a clear, concise manner. A color image can capture the large volume of numbers that a Cray Research system can generate and display it clearly. In the X Window System, colors can be used easily. However, if a client will be communicating with more than one server, you must take great care to ensure portability. The techniques described in this subsection can also greatly decrease the amount of network traffic that a client will generate when color is used.

Every X server has information about its particular color characteristics in a visual structure that you can obtain by using the DefaultVisual macro instruction or the XGetVisualInfo or XMatchVisualInfo functions. Several visual types are available; the most common are StaticGray (the typical visual for monochrome workstations) and PseudoColor (a common color visual in low-performance and medium-performance color workstations). Advanced color workstations may use other visuals, such as TrueColor or DirectColor.

To determine which visual is available for a client, you can use code similar to the following:

```
int my_visual;
Visual *visual;
visual = DefaultVisual(display, screen);
switch (visual–>class) {
     case StaticGray:
        my_visual = StaticGray;
        printf(”This is a StaticGray device\n”);
        break;
     case PseudoColor:
        my_visual = PseudoColor;
        printf(”This is a PseudoColor device\n”);
        break;
     case TrueColor:
        my_visual = TrueColor;
        printf(”This is a TrueColor device\n”);
        break;
     case DirectColor:
        my_visual = DirectColor;
        printf(”This is a DirectColor device\n”);
        break;
     default:
        printf(”ERROR in Visual.  Call programmer\n”);
        exit(1);
        break;
     }
```

To use colors in a client application, you can use the default colormap, or you can define a colormap, allocate one or more colors, and assign each color to a graphics context, which is then used for any drawing. Usually, you can use the server's default colormap (which is preferable), although for some applications, you might have to define a separate colormap; you should base your decision on whether the number of colors available in the default colormap is suffient for your application. A workstation with an 8-bit-per-pixel PseudoColor display is limited to 256 colors per colormap; a window manager might have allocated colors, other clients might have allocated colors, running `xstdcmap` might allocate colors, leaving a client with few colors to allocate.

You can either define colors by specific red, green, and blue intensities, or use any of several hundred predefined named colors. If you need only a few separate colors, it is much easier to use named colors to write a portable client program, although these colors may look somewhat different on various servers. The predefined named colors are defined in a dbm-format color name database on the server. It defaults to `/usr/lib/x11/rgb`. To display the color database, use the `showrgb` client or you may `cat` the database source file `/usr/lib/x11/rgb.txt`. The following example shows a color allocation and setting of a graphics context:

```
Colormap my_map;
XColor my_color;
GC my_gc;

     my_gc = XCreateGC(display, my_window, 0, 0);
     if(my_visual == PseudoColor) {
        my_map = DefaultColormap(display, screen);
        XParseColor(display, my_map, ”Red”, &my_color);
        XAllocColor(display, my_map, &my_color);
        XSetForeground(display, my_gc, my_color.pixel);
        }
     else {  /* use black if not PseudoColor device  */
        XSetForeground(display, my_gc, BlackPixel(display,screen));
        }
```

If the graphics application uses only the X Window System functions for line drawing, polygon drawing and filling, and text drawing, the use of color graphics does not greatly increase network usage. A simple drawing instruction that is sent from the client to the server is the same number of bytes, whether there is 1 bit per pixel, 8 bits per pixel, or 24 bits per pixel on the server screen. Thus, you can quickly draw on a window or a *pixmap* (a drawable mechanism that resides in the server memory) on either a black-and-white (monochrome) or a color workstation.

The additional overhead for allocating colors should be incurred only once during the client initialization. However, if you generate graphics on the client by using an `XImage` structure that resides in the client memory (see “Using images,” page 15), and then copy this image to the server, the network traffic is increased as the number of bit planes per pixel increases. Therefore, to minimize network traffic, an `XImage` structure should be avoided unless complex pixel-by-pixel manipulations are necessary.

Using fonts
3.4

Fonts are server resources that specify the style and size of print. Each server stores its own fonts. A remote client must be able to adapt to the fonts available on any given server. For example, because standard fonts and font names differ among X Window system versions, you must write clients to accommodate each potential server. A server can have both standard and nonstandard fonts installed. To be truly portable, however, a client should not depend on nonstandard fonts. To obtain a listing of the fonts available on a given server, use the `xlsfonts` command.

A client can use `XLoadQueryFont` to check the existence of a font before it tries to use that font. If you use `XLoadFont` with an unknown font name, an X protocol error occurs, but if you use `XLoadQueryFont` with an unknown font name, it does not. The following example uses `XLoadQueryFont`:

```
Display *d;
XFontStruct *theFont;

d = XOpenDisplay(argv[1]);

theFont = XLoadQueryFont(d,”bad_font_name”);
if (theFont == 0) {
    printf(”font name = %s, return = %d\n”, ”bad_font_name”,
theFont);
}
```

When using `XListFonts` in a remote client, you should be careful when using wildcard characters in a font name to match any font with a specific character string in it. For example, the string `*–times–medium–r–normal––*` matches the following font names:

```
–adobe–times–medium–r–normal––10–100–75–75–p–54–iso8859–1
–adobe–times–medium–r–normal––12–120–75–75–p–64–iso8859–1
–adobe–times–medium–r–normal––14–140–75–75–p–74–iso8859–1
–adobe–times–medium–r–normal––18–180–75–75–p–94–iso8859–1
–adobe–times–medium–r–normal––24–240–75–75–p–124–iso8859–1
–adobe–times–medium–r–normal––8–80–75–75–p–44–iso8859–1
```

However, the string `*–times–medium–r–normal––18–*` matches only the following font name:

```
–adobe–times–medium–r–normal––18–180–75–75–p–94–iso8859–1
```

If a client uses `XListFonts` to find all of the `*–times–medium–r–normal––*` fonts, and then examines each font by using `XLoadQueryFont`, a lot of traffic is generated on the network, because `XLoadQueryFont` returns information on every character in the font. Clients can experience several seconds delay by using `XListFonts` and `XLoadQueryFont` inefficiently.

Using images
3.5

An *image* is a client-resident representation of a screen area, not necessarily a window. Images differ from pixmaps or windows in that the image is created and stored on the client side; pixmaps and windows use server resources. The image, really a raster image, is a pixel representation in memory. You must take care when using images because various workstations interpret pixels differently, especially on color displays. (Even the definitions of black pixel and white pixel vary from manufacturer to manufacturer.) Some workstations use bit and byte ordering that is different from that used on other workstations. This further complicates matters.

Fortunately, Xlib provides all of the mechanisms necessary to convert images from client format to server format, so that an image can be sent to the server and displayed properly.

To overcome the problem of which pixel value represents black and which pixel value represents white, Xlib provides the `BlackPixel` and `WhitePixel` macros. In the following example, `XCreatePixmapFromBitmapData` requires values for the foreground and background pixel values. Using the `BlackPixel` and `WhitePixel` macros, you can ensure that the image will be correct, regardless of the server.

```
Display *dp;
Window winp;
Pixmap pixmapp;

dp = XtDisplay(toplevel);
winp = DefaultRootWindow(dp);
pixmapp = XCreatePixmapFromBitmapData(dp, winp, mandelbrot_bits,
       mandelbrot_width, mandelbrot_height,
       BlackPixel(dp, DefaultScreen(dp)),
       WhitePixel(dp, DefaultScreen(dp)), depth);
```

Overcoming the problem of byte and bit ordering requires that you modify the `XImage` structure, which is returned from the `XCreateImage` function call. The `byte_order` and `bitmap_bit_order` fields each contain a value that indicates most-significant bit first (MSBFirst) or least-significant bit first (LSBFirst) ordering. These refer to how programmers chose to represent the image in memory, not to the bit and byte ordering of the client or server hardware. Therefore, an image can be LSBFirst in the client and MSBFirst in the hardware.

You must know the contents of the XImage structure because the default values for byte_order and bitmap_bit_order are those of the server hardware, and the program must set these values to correspond to the actual byte order and bit order of the image. This can be done immediately after the XCreateImage call, as follows:

```
static XImage *xip = NULL;
Display *draw_d;
Visual *draw_v;
static char *dp = NULL;

dp=malloc (width * height);
xip = XCreateImage(draw_d, draw_v,
       depth, ZPixmap, 0, dp, width,
       height, bitmap_pad, byte_per_line);
xip->byte_order = MSBFirst;
xip->bitmap_bit_order = MSBFirst;
```

Debugging tools
3.6

One of the best tools available for debugging application programs on a Cray Research system is the cdbx(1) program. It has an X Window System interface, which displays the text being debugged, and it has command buttons for common operations such as run, stop at, step, next, and so on. If the DISPLAY environment variable is set to your display server, or the –display option is found on the cdbx command line, the X interface is enabled automatically. See cdbx(1) for details of its use.

Note

Reaching a breakpoint in cdbx during a server grab (a time when only the X program can use the server) hangs your display server. This happens only when cdbx and the X client are connected to the same server.

Another powerful tool for debugging X applications is `xscope`. This tool, which is in the public domain, traces all traffic between client and server. To the client, `xscope` appears as a separate display server because it uses a unique display number, then passes the protocol to the proper display.

A typical way of using `xscope` is to start it on your workstation with no parameters. This defaults to display 1 of the workstation. You must then set the `DISPLAY` environment variable to display 1 of the workstation and start the client.

The `xscope` tool intercepts each packet and displays it in an easy-to-read format. The following example shows this process. You can find an example of `xscope` output in "Example `xscope` Output," page 53.

Workstation	Cray system
`xscope`	–
–	`setenv DISPLAY mirror:1`
–	`xclient`
Protocol is traced here	–
`xclient` displays here	–

Sometimes it is difficult to determine the protocol request that actually caused a problem, because the client requests are buffered up; that is, several requests can be sent in the same transmission. To force Xlib to send requests in synchronous mode (as soon as they are built), you can issue the `XSynchronize` call from the `xclient` program. Using `XSynchronize` can make the `xscope` output easier to read, but this technique should be used only for debugging; unbuffered traffic increases network congestion. The following example shows the use of `XSynchronize`:

```
Display *display;
display = XOpenDisplay(argv[1]);
XSynchronize(display, 1);        /* force synchronous mode */
```

X toolkit clients can use the `–sync` command-line option to force synchronous mode, without editing or recompiling source code.

Fortran and X [4]

Although you cannot call Xlib functions directly from Fortran, you can mix Fortran functions and C functions in one binary file. Using the techniques described in this section, you can use Fortran for computation, and write Fortran-callable C functions that use Xlib to handle the graphics. Similar interfaces can be used for other languages, such as Pascal and Ada.

The following differences between Fortran and C must be considered:

- A Fortran restriction requires that the subroutine or function name be in uppercase letters when Fortran subroutines are called from C, and vice versa.
- The Fortran calling sequence is call-by-address, and the C calling sequence is call-by-value. Therefore, any parameters that are passed from C to Fortran or from Fortran to C must be pointers.
- Fortran and C handle character strings differently.

Figure 2, page 20, shows the architecture of a client that uses both Fortran and C. It consists of a main program that interfaces with functions and subroutines.

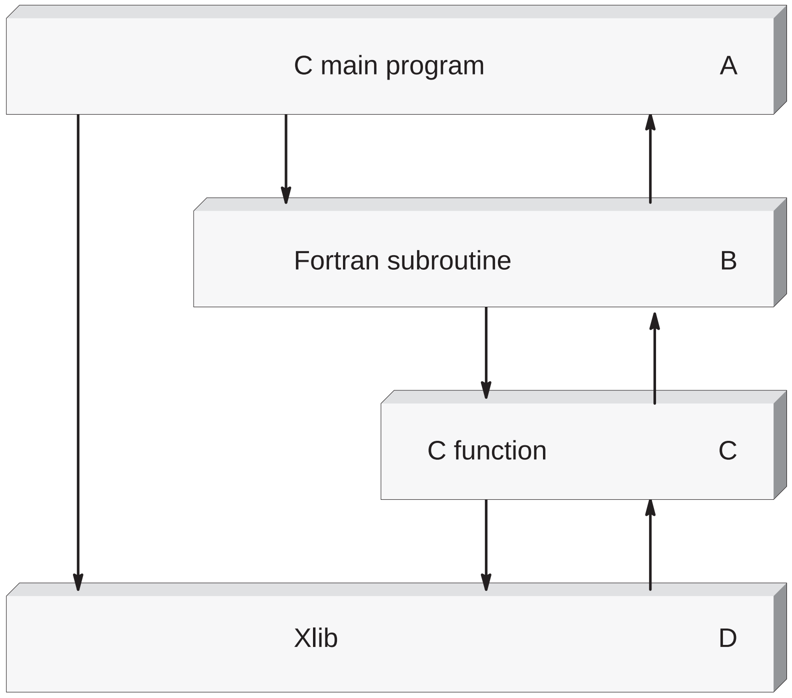

Figure 2. Architecture of C and Fortran client

The main program, represented by box A, performs tasks such as initialization, opening the display, creating windows, creating command buttons, and so on. If the main program is written using only Xlib, it contains its own event-handling loop; if it uses the Intrinsics toolkit, it contains a call to `XtMainLoop`. You can call the Fortran subroutine represented by box B from the main program's event loop if Xlib is being used; it can be called by a *callback procedure* (a routine called when a button is pressed) if the Intrinsics toolkit is being used. To do graphic output, the Fortran subroutine calls a C function (box C) to generate the appropriate Xlib calls. Xlib (box D) generates the protocol requests.

To illustrate these techniques, the remainder of this section contains examples of code. You can find complete listings of the main program (`mandelf`) in "Complete `xmandelf` Source Code," page 27.

The following example shows C calling Fortran and Fortran calling C. The main program uses the Intrinsics toolkit (Xt), and declares a *callback table* (a set of callback procedures).

```
void mandel ();
XtCallbackRec mandel_callbacks[] = {
    { mandel, NULL },
    { NULL, NULL },
},
```

The main program also provides packaging for the rest of the code by performing toolkit initialization, creating widgets such as command buttons, and invoking Xt's main loop, as follows:

```
main(argc,argv)
{
        Widget toplevel, mandelbutton;
        Arg argies[10];
        toplevel = XtInitialize(”Anyname”, ”Anyname”, NULL, 0, &argc, argv);
         –
         –
         –
        XtSetArg( argies[i], XtNcallback, (XtArgVal) mandel_callbacks); i++;
        mandelbutton = XtCreateManagedWidget(”mandel”, commandWidgetClass,
                 form, argies, i);
         –
         –
         –
        XtRealizeWidget(toplevel);
        XtMainLoop();
}
```

The callback table referred to in the previous example is a set of C language routines that will be called when a certain command button named mandel is pressed. The following routine, also called mandel, handles the callback and sets up to call the Fortran subroutine GENERATE, which does the actual calculations.

This mandel routine might be in the same file as the main routine.

Note

Because Fortran is a call-by-address process, all of the parameters on the GENERATE subroutine call are addresses.

```
mandel(w, closure, call_data)
    Widget w;
    caddr_t closure;
    caddr_t call_data;
{
        -
        -
        -

    GENERATE(&wheight,&y0,&incry,&wwidth,&x0,&incrx,&iter);
}
```

The following example is an excerpt from the Fortran subroutine named GENERATE, which the mandel function called. When it completes its calculations, it calls a C function (called FPUTI) to build the Xlib graphics requests.

```
subroutine GENERATE(wheight,y0,incry,wwidth,x0,incrx,iter)
       -
       -
       -
    call FPUTI(ixlo,iylo,index)
```

The following code is the C function `FPUTI`, which must convert parameters from the form that Fortran produces to the form that the call to Xlib requires (in this case, the `XPutImage` call):

```
FPUTI(pix,piy,index)
int *pix,*piy;
int index[64*64];
{
       -
       -
       -
       XPutImage(draw_d,draw_win,draw_gc,xip,0,0,*pix,*piy,VSIZE,VSIZE);
}
```

The following code is the C function `FPUTI`, which must convert parameters from the form that Fortran produces to the form that the call to Xlib requires (in this case, the `XPutImage` call):

```
FPUTI(pix,piy,index)
int *pix,*piy;
int index[64*64];
{
       –
       –
       –
       XPutImage(draw_d,draw_win,draw_gc,xip,0,0,*pix,*piy,VSIZE,VSIZE);
}
```

Cray Research Clients [5]

Client programs that Cray Research, Inc. has developed are documented in the *UNICOS Visual Interfaces User's Guide*, publication SG–3094, *UNICOS CDBX Symbolic Debugger Reference Manual*, publication SR–2091, *UNICOS User Commands Reference Manual*, publication SR–2011, and the *UNICOS Administrator Commands Reference Manual*, publication SR–2022. Brief summaries of all of these client programs are documented in the *X Window System Resources Ready Reference*, publication SQ–2123. The following list contains these client programs, each with the publication number of the manual in which it is found.

Program	Manual
`atexpert`(1)	SR–2011
`cdbx`(1)	SR–2091
`flowview`(1)	SR–2011
`multimeter`(1)	SR–2011
`netperf`(8)	SR–2022
`perfview`(1)	SR–2011
`profview`(1)	SR–2011
`stategraph`(1)	SR–2011
`timeline`(1)	SR–2011
`xbrowse`(1)	SG–3094
`xfm`(1)	SR–2011
`xproc`(1)	SR–2011

As Cray Research continues to develop client programs, additional programs will be added to this list.

Complete xmandelf Source Code [A]

The xmandelf program is a Mandelbrot set generator that uses Fortran and the X Window System.

xmandelf.c
A.1

The following xmandelf.c program is the main xmandelf program. It creates the window and all of the buttons for the Mandelbrot process.

```
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/Cardinals.h>
#include <X11/Xaw/Label.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Logo.h>
#include <X11/Xaw/Paned.h>
#include <X11/Xaw/Box.h>
#include <X11/cursorfont.h>
#include <stdio.h>
#include <sys/utsname.h>
#include <unistd.h>

Widget drawform;
Widget detaillabel, mflopslabel, ncpulabel;      /* in widget ibox */
Widget xlabel, ylabel;   /* in widget bbox */
Cursor crosshair_xcr;
int depth;
int iter;
```

(continued)

```
int maxcpus = 8;
int ncpus = 4;

char ncputext[12];
void mandel();

void clear_callback(w, closure, call_data)
     Widget w;
     XtPointer closure;
     XtPointer call_data;
{
     Display *draw_d;
     Window draw_win;

     draw_d = XtDisplay(drawform);
     draw_win = XtWindow(drawform);

     XClearWindow(draw_d, draw_win);
}
void quit_callback(w, closure, call_data)
     Widget w;
     XtPointer closure;
     XtPointer call_data;
{
     exit(0);
}
void ncpu_callback(w, closure, call_data)
     Widget w;
     XtPointer closure;
     XtPointer call_data;
{
     Arg arg;

     ncpus++;
     if (ncpus > maxcpus) ncpus = 1;
```

(continued)

```
     sprintf(ncputext, ”%d cpus”, ncpus);
     arg.name = XtNlabel; arg.value = (XtArgVal) ncputext;
     XtSetValues(ncpulabel, &arg, 1);
}

XtCallbackRec clear_callbacks[] = {
  { clear_callback, NULL },
  { NULL, NULL },
};

void julia();
XtCallbackRec julia_callbacks[] = {
  { julia, NULL },
  { NULL, NULL },
};
void zoom();
XtCallbackRec zoom_callbacks[] = {
  { zoom, NULL },
  { NULL, NULL },
};
void mooz();
XtCallbackRec mooz_callbacks[] = {
  { mooz, NULL },
  { NULL, NULL },
};
void redo();
XtCallbackRec redo_callbacks[] = {
  { redo, NULL },
  { NULL, NULL },
};
void resetmandel();
XtCallbackRec mandel_callbacks[] = {
  { resetmandel, NULL },
  { NULL, NULL },
};
```

(continued)

```
XtCallbackRec quit_callbacks[] = {
  { quit_callback, NULL },
  { NULL, NULL },
};

void detail();
XtCallbackRec detail_callbacks[] = {
  { detail, NULL },
  { NULL, NULL },
};

void reset();
XtCallbackRec reset_callbacks[] = {
  { reset, NULL },
  { NULL, NULL },

};
void ncpu_callback();
XtCallbackRec ncpu_callbacks[] = {
  { ncpu_callback, NULL },
  { NULL, NULL },
};

main(argc,argv)
int argc;
char **argv;
{
    Widget toplevel, pane;
    Widget box, ibox, bbox;                      /* in widget pane */
    Widget hostlabel;                            /* in widget box  */
    Widget juliabutton, quitbutton, mandelbutton; /* in widget box  */
```

(continued)

```
      Widget moozbutton, zoombutton, redobutton, clearbutton;/*in box */
      Widget detailbutton, resetbutton, ncpubutton; /* in widget ibox */


      Display *dp;
      int i;
      char siter[20];
    Arg argies[10];
    struct utsname U;

    toplevel = XtInitialize(”Anyname”,”Xmandelf”,NULL,0,&argc,argv);
    dp = XtDisplay(toplevel);

    depth = DisplayPlanes(dp,DefaultScreen(dp));

    pane = XtCreateManagedWidget(”pane”, panedWidgetClass, toplevel,
      NULL, ZERO);

    box = XtCreateManagedWidget(”box”, boxWidgetClass, pane, NULL,
      ZERO);
    ibox = XtCreateManagedWidget(”ibox”, boxWidgetClass, pane, NULL,
      ZERO);

      uname (&U);
      i = 0;
    XtSetArg( argies[i], XtNborderWidth, (XtArgVal) 0); i++;
      hostlabel = XtCreateManagedWidget(U.nodename, labelWidgetClass,
            box, argies, i);

    i = 0;
    XtSetArg( argies[i], XtNcallback, (XtArgVal) mandel_callbacks); i++;
      mandelbutton = XtCreateManagedWidget(”mandel”, commandWidgetClass,
            box, argies, i);
```

(continued)

```
   i = 0;
   XtSetArg( argies[i], XtNcallback, (XtArgVal) zoom_callbacks); i++;
     zoombutton = XtCreateManagedWidget(”mandelzoom”, commandWidget-
 Class,
           box, argies, i);

     i = 0;
   XtSetArg( argies[i], XtNcallback, (XtArgVal) mooz_callbacks); i++;
     moozbutton = XtCreateManagedWidget(”unzoom”, commandWidgetClass,
           box, argies, i);

     i = 0;
   XtSetArg( argies[i], XtNcallback, (XtArgVal) redo_callbacks); i++;
     redobutton = XtCreateManagedWidget(”redo”, commandWidgetClass,
           box, argies, i);

   i = 0;
   XtSetArg( argies[i], XtNcallback, (XtArgVal) julia_callbacks); i++;
     juliabutton = XtCreateManagedWidget(”julia”, commandWidgetClass,
           box, argies, i);

   i = 0;
   XtSetArg( argies[i], XtNcallback, (XtArgVal) clear_callbacks); i++;
     clearbutton = XtCreateManagedWidget(”clear”, commandWidgetClass,
           box, argies, i);

   i = 0;
   XtSetArg( argies[i], XtNcallback, (XtArgVal) quit_callbacks); i++;
     quitbutton = XtCreateManagedWidget(”quit”, commandWidgetClass,
           box, argies, i);

     i = 0;
   XtSetArg( argies[i], XtNborderWidth, (XtArgVal) 0); i++;
     mflopslabel = XtCreateManagedWidget(”   0 megaflops”,
     labelWidgetClass, ibox, argies, i);
```

(continued)

```
  i = 0;
    if (depth > 1) iter = 256; else iter = 64;
    sprintf(siter, ”%d iterations”, iter);
  XtSetArg( argies[i], XtNborderWidth, (XtArgVal) 0); i++;
    detaillabel = XtCreateManagedWidget(siter, labelWidgetClass,
          ibox, argies, i);

    i = 0;
    ncpus = maxcpus = sysconf(_SC_CRAY_NCPU);
    sprintf(ncputext, ”%d cpus”, maxcpus);
  XtSetArg( argies[i], XtNborderWidth, (XtArgVal) 0); i++;
    ncpulabel = XtCreateManagedWidget(ncputext, labelWidgetClass,
          ibox, argies, i);

  i = 0;
  XtSetArg( argies[i], XtNcallback, (XtArgVal) detail_callbacks); i++;
detailbutton = XtCreateManagedWidget(”increase iterations”,
commandWidgetClass,ibox, argies, i);

  i = 0;
  XtSetArg( argies[i], XtNcallback, (XtArgVal) reset_callbacks); i++;
resetbutton = XtCreateManagedWidget(”reset iterations”,
commandWidgetClass, ibox, argies, i);

    i = 0;
    XtSetArg( argies[i], XtNcallback, (XtArgVal) ncpu_callbacks); i++;
       ncpubutton = XtCreateManagedWidget(”cpus”, commandWidgetClass,
          ibox, argies, i);

       i = 0;
       XtSetArg( argies[i], XtNheight, (XtArgVal) 512); i++;
       XtSetArg( argies[i], XtNwidth, (XtArgVal) 512); i++;
       XtSetArg( argies[i], XtNbackground, (XtArgVal)
          BlackPixel(dp, DefaultScreen(dp))); i++;
```

(continued)

```
  drawform = XtCreateManagedWidget(”DrawForm”, logoWidgetClass, pane,
          argies, i);

  bbox = XtCreateManagedWidget(”box”, boxWidgetClass, pane, NULL,
ZERO);

  i = 0;
  XtSetArg( argies[i], XtNwidth, (XtArgVal) 512); i++;
  XtSetArg( argies[i], XtNresize, (XtArgVal) False); i++;
XtSetArg( argies[i], XtNborderWidth, (XtArgVal) 0); i++;
  xlabel = XtCreateManagedWidget(””, labelWidgetClass,
          bbox, argies, i);

  i = 0;
  XtSetArg( argies[i], XtNwidth, (XtArgVal) 512); i++;
  XtSetArg( argies[i], XtNresize, (XtArgVal) False); i++;
XtSetArg( argies[i], XtNborderWidth, (XtArgVal) 0); i++;
  ylabel = XtCreateManagedWidget(””, labelWidgetClass,
          bbox, argies, i);

    i = 0;
    XtSetArg( argies[i], XtNcallback, (XtArgVal) reset_callbacks);
i++;
  resetbutton = XtCreateManagedWidget(”reset iterations”,
commandWidgetClass,
          ibox, argies, i);

  XtRealizeWidget(toplevel);
  crosshair_xcr = XCreateFontCursor(dp, XC_crosshair);
XDefineCursor(dp, XtWindow(drawform), crosshair_xcr);

  XtMainLoop();
}
```

mandel.c
A.2

The following mandel.c routines handle all actions that result from the pushing of buttons. One routine calls the Fortran mandelbrot routine.

```
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <stdio.h>
extern Widget detaillabel, drawform, mflopslabel;
extern Widget xlabel, ylabel;
struct zoomd {
    struct zoomd *zp;
    double lx, ux;
    double ly, uy;
};
struct zoomd *zoomp = NULL, *zoompn;
char *malloc();
double GENERATE();

extern int depth;
extern long iter;
extern int ncpus;
long mwidth;
long bias = 1;                /* bias into color table */
long wwidth = 0;
long wheight = 0;

Arg arg;
XImage *xip = NULL;
char *dp = NULL;
Window draw_win;
GC draw_gc;
Screen *draw_Screen;
Display *draw_d;
Visual *draw_v;
```

(continued)

```
#define MIN(a,b)((a) < (b) ? (a) : (b))
#define MAX(a,b)((a) < (b) ? (b) : (a))

  void resetmandel(w, closure, call_data)
       Widget w;
       caddr_t closure;
       caddr_t call_data;
  {
       if (zoomp) { /* reset to beginning by popping zps off stack */
          while (zoompn = zoomp->zp) {
          free(zoomp);
          zoomp = zoompn;
       }
  } else {                                   /* This is first time */
          zoomp = (struct zoomd *) malloc(sizeof (struct zoomd));
       zoomp->zp = NULL;  /* NULL means last in stack - don't pop */
       zoomp->lx = -2.25; zoomp->ux = .75;
       zoomp->ly = -1.5; zoomp->uy = 1.5;
  }
  setxylabels();
  setiter();
  mandel(w, closure, call_data);
}

mandel(w, closure, call_data)
    Widget w;
    caddr_t closure;
    caddr_t call_data;
{
    double wx, wy, oldwx, oldwy;
    double x, y, x0, y0;
    double incrx, incry;
    double mflops;
    long ix, iy;
    long i;
    long zero = 0;
```

(continued)

```
    draw_d = XtDisplay(drawform);
    draw_win = XtWindow(drawform);
    draw_Screen = XtScreen(drawform);
    draw_gc = draw_Screen->default_gc;
    draw_v = draw_Screen->root_visual;

    if (dp == NULL) {
       mwidth = (depth > 1)? 64: 8;
       dp = malloc(mwidth*64);
       if (dp == NULL) {printf("malloc failed\n"); return; }
       if (xip) XDestroyImage(xip);
       xip = XCreateImage(draw_d, draw_v, depth, ZPixmap, 0, dp,
          64, 64, 8, 0);
       xip->byte_order = MSBFirst;
       xip->bitmap_bit_order = MSBFirst;
       xip->bits_per_pixel = 8;

    }
    XtSetArg(arg, XtNwidth, &wwidth);
    XtGetValues(drawform, &arg, 1);
    XtSetArg(arg, XtNheight, &wheight);
    XtGetValues(drawform, &arg, 1);

    incrx = (zoomp->ux - zoomp->lx)/wwidth;
    incry = (zoomp->uy - zoomp->ly)/wheight;

    y0 = zoomp->uy-incry;
    x0 = zoomp->lx+incrx;

    if (closure == 0) XClearWindow(draw_d, draw_win);

    mflops =
    GENERATE(&wheight,&y0,&incry,&wwidth,&x0,&incrx,&iter,&ncpus);
    setmflops(mflops);
 }
```

(continued)

```
  FPUTI(pix,piy,index)
  int *pix,*piy;
  int index[64*64];
  {
    int i,j;

  #define VSIZE 64

    if (depth == 1) {
        int *idp = (int *) dp;
        long shiftw;
        for (i = 0; i < VSIZE; i++)
           *(idp + i) = 0;
        for (i = 0; i < VSIZE; i++, idp++) {
           shiftw = 0x8000000000000000;
           for (j = 0; j < VSIZE; j++) {
              if ((index[i* VSIZE + j] & 7)  != 0)
                *idp |= shiftw;
              shiftw >>= 1;
           }
        }
  #if 0
    } else if (depth == 4) {
        for (i = 0; i < VSIZE; i+=2) {
          for (j = 0; j < VSIZE; j++) {
          dp[i * VSIZE + j/2] = ( ((index[i * VSIZE + j] & 0x0f) <<4)
|                                (index[i * VSIZE + j + 1] & 0x0f)  );
           }
        }
  #endif
    } else {                         /* depth == 8 */
        for (i = 0; i < VSIZE; i++) {
        for (j = 0; j < VSIZE; j++) {
```

(continued)

```
            dp[i * VSIZE + j] = index[i * VSIZE + j] + bias;
         }
      }
   }

   XPutImage(draw_d,draw_win,draw_gc,xip,0,0,*pix,*piy,VSIZE,VSIZE);
}

void zoom(w, closure, call_data)
     Widget w;
     caddr_t closure;
     caddr_t call_data;
{
     Window draw_win;
     GC draw_gc;
     Screen *draw_Screen;
     Display *draw_d;
     Window Root;
     int S;

     draw_d = XtDisplay(drawform);
     if (zoomp == NULL) { XBell(draw_d, 0); return; }
     draw_win = XtWindow(drawform);
     draw_Screen = XtScreen(drawform);
     draw_gc = draw_Screen->default_gc;
     S = DefaultScreen(draw_d);
     Root = RootWindow(draw_d, S);

     XSetForeground(draw_d, draw_gc, 0x55);
     XSetSubwindowMode(draw_d, draw_gc, IncludeInferiors);
     XSetFunction(draw_d, draw_gc, GXxor);

     XSelectInput(draw_d, draw_win,
        ButtonPressMask | ButtonReleaseMask | PointerMotionMask);
     while (1) {
```

(continued)

```
static int rubberband = 1;
XEvent report;
XNextEvent(draw_d, &report);
switch(report.type) {
   int winx0,winy0,winx1,winy1,width,height;
   int x0, y0, x1, y1;
   double tw, th;

   case ButtonPress:
      XGrabServer(draw_d);
      x0 = winx0 = report.xbutton.x;
      y0 = winy0 = report.xbutton.y;
         rubberband = 0;
         width = height = 0;
         break;

      case ButtonRelease:
         winx1 = report.xbutton.x;
         winy1 = report.xbutton.y;
XDrawRectangle(draw_d,draw_win, draw_gc, x0, y0, width, width);
         if ((width = winx1 – winx0) < 0)
                 width = – width;
         if ((height = winy1 – winy0) < 0)
                 height = – height;
         if (width < height)
                 width = height;
         x0 = (winx1 > winx0) ? winx0 : winx0 – width;
         y0 = (winy1 > winy0) ? winy0 : winy0 – width;
         x1 = x0 + width;
         y1 = y0 + width;
XDrawRectangle(draw_d,draw_win, draw_gc, x0, y0, width, width);
         rubberband = 1;
```

(continued)

```
            XSetFunction(draw_d, draw_gc, GXcopy);
            XUngrabServer(draw_d);
            XFlush(draw_d);
            tw = zoomp->ux – zoomp->lx;
            th = zoomp->uy – zoomp->ly;
            zoompn = (struct zoomd *) malloc(sizeof (struct zoomd));
            zoompn->zp = zoomp;                 /* push onto stack */
            zoompn->ux = zoomp->lx + ((double)x1/(double)wwidth) *tw;
            zoompn->lx = zoomp->lx + ((double)x0/(double)wwidth) *tw;
            zoompn->ly = zoomp->uy – ((double)y1/(double)wheight)*th;
            zoompn->uy = zoomp->uy – ((double)y0/(double)wheight)*th;
              zoomp = zoompn;        /* zoomp is current pointer */
              setxylabels();
              mandel(w, closure, call_data);
              return;       /* call mandel with new width, height */

          case MotionNotify:
            if (rubberband) break;
            while (XCheckTypedEvent(draw_d, MotionNotify, &report));
       XDrawRectangle(draw_d,draw_win, draw_gc, x0, y0, width, width);
              winx1 = report.xbutton.x;
              winy1 = report.xbutton.y;
              if ((width = winx1 – winx0) < 0)
                      width = – width;
              if ((height = winy1 – winy0) < 0)
                      height = – height;

              if (width < height)
                      width = height;
              x0 = (winx1 > winx0) ? winx0 : winx0 – width;
              y0 = (winy1 > winy0) ? winy0 : winy0 – width;
       XDrawRectangle(draw_d,draw_win, draw_gc, x0, y0, width, width);
```

(continued)

```
                break;
            default:
                break;
        }
    }
}

void mooz(w, closure, call_data)
    Widget w;
    caddr_t closure;
    caddr_t call_data;
{
    draw_d = XtDisplay(drawform);
    if (zoomp == NULL) { XBell(draw_d, 0); return; }
    if (zoomp->zp == NULL) { XBell(draw_d, 0); return; }

    if (zoompn = zoomp->zp) {                    /* last on stack */
          free(zoomp);
          zoomp = zoompn;
    }
    setxylabels();
    mandel(w, closure, call_data);
}
void redo(w, closure, call_data)
    Widget w;
    caddr_t closure;
    caddr_t call_data;
{
    draw_d = XtDisplay(drawform);
    if (zoomp == NULL) { XBell(draw_d, 0); return; }
    mandel(w, 1, call_data);
}

void detail(w, closure, call_data)
    Widget w;
```

(continued)

```
    caddr_t closure;
    caddr_t call_data;
{

    if (depth > 1) iter += 256; else iter += 64;
    setiter();
    /*mandel(w, 1, call_data);*/
}
void reset(w, closure, call_data)
    Widget w;
    caddr_t closure;
    caddr_t call_data;
{
    if (depth > 1) iter = 256; else iter = 64;
    setiter();
}

setiter()
{
    char siter[20];
    sprintf(siter, "%d iterations", iter);
    arg.name = XtNlabel; arg.value = (XtArgVal) siter;
    XtSetValues(detaillabel, &arg, 1);
}

setmflops(r)
float r;
{
    char string[20];
    sprintf(string, "%.1f megaflops", r);
    arg.name = XtNlabel; arg.value = (XtArgVal) string;
    XtSetValues(mflopslabel, &arg, 1);
}
setxylabels()
{
```

(continued)

```
    char xtext[80], ytext[80];
    sprintf(xtext,”%15.10f < x < %15.10f”, zoomp−>lx, zoomp−>ux);
    arg.name = XtNlabel; arg.value = (XtArgVal) xtext;
    XtSetValues(xlabel, &arg, 1);
    sprintf(ytext,”%15.10f < y < %15.10f”, zoomp−>ly, zoomp−>uy);
    arg.name = XtNlabel; arg.value = (XtArgVal) ytext;
    XtSetValues(ylabel, &arg, 1);
}
```

julia.c
A.3

The following `julia.c` routines handle the `julia` button by generating a julia set.

```
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <stdio.h>
extern Widget drawform;
struct zoomd {
    struct zoomd *zp;
    double lx, ux;
    double ly, uy;
};
extern int depth;
extern struct zoomd *zoomp;
extern long iter;
XWindowAttributes draw_wattr;

void julia(w, closure, call_data)
    Widget w;
```

(continued)

```
    caddr_t closure;
    caddr_t call_data;
{

    double a,b;
    double wx, wy, oldwx, oldwy;
    double sx, x, y;
    double incrx, incry;
    long ix, iy;
    static long width = 0;
    static long mwidth = 0;
    static long bias = 1;

    Arg arg;
    int wwidth, wheight;
    static XImage *xip1 = NULL, *xip2 = NULL;
    static char *dp1 = NULL, *dp2 = NULL;
    char *malloc();
    char ab[80];
    Window draw_win;
    static Window s_win = NULL;
    GC draw_gc;
    Screen *draw_Screen;
    Display *draw_d;
    Visual *draw_v;
    Window Root;
    int S;                                      /* Screen */

       draw_d = XtDisplay(drawform);
       if (zoomp == NULL)                { XBell(draw_d, 0); return;}
       draw_win = XtWindow(drawform);
       draw_Screen = XtScreen(drawform);
       draw_gc = draw_Screen->default_gc;
       draw_v = draw_Screen->root_visual;
```

(continued)

```
         XtSetArg(arg, XtNwidth, &wwidth);
         XtGetValues(drawform, &arg, 1);
         XtSetArg(arg, XtNheight, &wheight);
         XtGetValues(drawform, &arg, 1);

      XSelectInput(draw_d,draw_win,ButtonPressMask|ButtonReleaseMask);
         while (1) {
            int winx, winy;
            XEvent report;
            XNextEvent(draw_d, &report);
            if (report.type == ButtonRelease) {
    /* if (report.xbutton.x <= 0 || report.xbutton.y <= 0) {  */
    /* make sure button release is in drawform window         */
               if (report.xbutton.window != draw_win ||
                  report.xbutton.x & 0x8000 ||
                  report.xbutton.y & 0x8000 ||
                  (report.xbutton.x == 0 && report.xbutton.y == 0) ||
                  report.xbutton.x > wwidth ||
                  report.xbutton.y > wheight) {
                     XBell(draw_d, 0);
                     continue;
            }
            a = ( (double)report.xbutton.x/ (double)wwidth);
            b = ( (double)report.xbutton.y/ (double)wheight);
            a = (a * (zoomp->ux - zoomp->lx)) + zoomp->lx;
            b = zoomp->uy - (b * (zoomp->uy - zoomp->ly));
            /*printf("julia: a = %f, b = %f\n", a, b);*/
            break;
         }
         continue;
      }
      S = DefaultScreen(draw_d);
      Root = RootWindow(draw_d, S);
      if (s_win == NULL) {
```

(continued)

```
          s_win = XCreateSimpleWindow(draw_d,Root,0,0,256,256,1,1,0);
          XSelectInput(draw_d, s_win, ExposureMask);
          XMapWindow(draw_d, s_win);
          XSync(draw_d, 0);
          while (1) {
             XEvent report;
          XNextEvent(draw_d, &report);
          if (report.type == Expose) break;
       }
     }
     sprintf(ab, "%f + %fi", a,b);
     XStoreName(draw_d, s_win, ab);
     XClearWindow(draw_d, s_win);

     XGetWindowAttributes(draw_d, s_win, &draw_wattr);
     if (width != draw_wattr.width) {
          width = draw_wattr.width;
          mwidth = (depth > 1)? width: (1 + width/8);
          /* XDestroyImage should free dp1 and dp2 as well */
          if (xip1) {XDestroyImage(xip1); XDestroyImage(xip2); }
          dp1 = malloc(mwidth);
          dp2 = malloc(mwidth);
          if (dp2 == NULL) {printf("malloc failed\n"); return; }

          xip1 = XCreateImage(draw_d, draw_v, depth, ZPixmap, 0, dp1,
                width, 1, 8, mwidth);
          xip2 = XCreateImage(draw_d, draw_v, depth, ZPixmap, 0, dp2,
                width, 1, 8, mwidth);
          xip1->byte_order = xip2->byte_order = MSBFirst;
          xip1->bitmap_bit_order = xip2->bitmap_bit_order = MSBFirst;
     }
     sx = x = -1.5; y = -1.5;
     incrx = -x/draw_wattr.width * 2;
     incry = -y/draw_wattr.height * 2;
```

(continued)

```
for (iy = 0; iy < (1 + draw_wattr.height/2); iy++, y+= incry) {
  long i;
  x = sx;
  for (ix = 0; ix < draw_wattr.width; ix++, x+= incrx) {
       wx = x; wy = y;
       for (i = 0; i < iter; i++) {
               oldwx = wx;
               wx = wx * wx – wy * wy + a;
               wy = 2 * oldwx * wy + b;
               if (wx * wx + wy * wy > 4) break;
       }
       if (depth > 1) {                        /* color */
             *(dp1 + ix) = i + bias;
             *(dp2 + width – ix – 1) = i + bias;
       } else {
          if (i == iter) {          /* could also test if (i & 2) */
               dp1[ix/8] |= 1 << (7 – ix&7);
               dp2[(width – ix – 1)/8] |= 1 << (8 – (width – ix)&7);
          }
       }
    }

  XPutImage(draw_d, s_win, draw_gc, xip1, 0,0, 0, iy, width, 1);
if (iy != draw_wattr.height – iy)
  XPutImage(draw_d,s_win,draw_gc,xip2,0,0,0, draw_wattr.height – iy,
    width, 1);
  if (depth == 1) for (i = 0 ; i < mwidth; i++) *(dp1 + i) = *(dp2 +
i) = 0;
}

}
```

generate.f

A.4

The following generate.f routine is a multitasked Fortran program that generates a Mandelbrot set.

```
      function generate(wheight,y0,incry,wwidth,x0,incrx,iter,n)
      parameter (incx=64,incy=64)
      real x0,incrx,y0,incry
      integer wwidth,wheight,index(0:incx–1,0:incy–1),flops
      complex w0(0:incx–1,0:incy–1), w(0:incx–1,0:incy–1)

cmic$ numcpus n
      flops = 0
      t0 = timef()

cmic$ do all autoscope private(index,w0,w)
ccmic$ do all shared(wheight,y0,incry,wwidth,x0,incrx,
ccmic$1              iter,dp,drawd,drawwin,drawgc,xip,flops)
ccmic$1 private(ix,iy,x,y,i,ssum,ixlo,iylo,ixhi,iyhi,index,w,w0,
findex,
ccmic$1          indexsum)

      do 500 iylo = 0,wheight–1,incy
        iyhi = min0(incy,wheight–1+1–iylo)
        do 500 ixlo = 0,wwidth–1,incx
          ixhi = min0(incx,wwidth–1+1–ixlo)

          do 100 iy=0,iyhi–1
            y = y0–iylo*incry – iy*incry
cdir$ shortloop
            do 100 ix=0,ixhi–1
              x = x0+ixlo*incrx + ix*incrx
              w0(ix,iy) = cmplx(x,y)
              w(ix,iy) = w0(ix,iy)*w0(ix,iy) + w0(ix,iy)
              index(ix,iy)=cvmgt(0,1,(real(w(ix,iy))*real(w(ix,iy))
      +                   + aimag(w(ix,iy))*aimag(w(ix,iy)).le.4.))
   100        continue
```

(continued)

```
            do 300 i=1,iter-1
              do 200 iy=0,iyhi-1
cdir$ shortloop
                do 200 ix=0,ixhi-1
                  if ( index(ix,iy) .eq. 0) then
                    w(ix,iy) = w(ix,iy)*w(ix,iy) + w0(ix,iy)
                  index(ix,iy)=cvmgt(0,i,(real(w(ix,iy))*real(w(ix,iy))
     +                      + aimag(w(ix,iy))*aimag(w(ix,iy)).le.4.))
                  end if
  200             continue
  300           continue

                indexsum = 0
                do 400 iy=0,iyhi-1
cdir$ shortloop
                  do 400 ix=0,ixhi-1
                    if (index(ix,iy).eq.0) index(ix,iy)=iter
                    indexsum = indexsum + index(ix,iy)
  400           continue

cmic$ guard
      flops = flops + 13*incx*incy + 11*indexsum
      call FPUTI(ixlo,iylo,index)
cmic$ end guard

  500 continue

      telapse = timef() - t0
      generate = 1.e-3*flops/telapse
c     print 990, 1.e-3*flops/telapse
c 990 format(' Mflops: ',f7.2)

            return
            end
```

makefile
A.5

The following makefile program compiles the xmandelf program.

```
F=cf77
CC=cc
CFLAGS = $(INCLUDES) –DUSG –DSYSV
FFLAGS =  –ZP
SHELL=/bin/sh

xmandelf: xmandelf.o mandel.o julia.o generate.o
        $(CF) $(FFLAGS) –o $@ xmandelf.o mandel.o julia.o  generate.o \
                 –lXaw –lXt –lXmu –lX11 –lXext

generate.o:        generate.f
        $(CF) $(FFLAGS) –c generate.f
clean:
        @–rm xmandelf *.o

install:        xmandelf
        if [ ! –d $$HOME/bin ] ; \
        then \
                 echo mkdir $$HOME/bin;\
                 mkdir $$HOME/bin;\
        fi
        cp xmandelf $$HOME/bin
```

Example xscope Output [B]

The xscope program is a tool for debugging X Window System applications. This appendix contains an example of xscope output.

```
0.00: Client -->   12 bytes
                 byte-order: MSB first
              major-version: 000b
              minor-version: 0000
0.02:                                   224 bytes <-- X11 Server
                                   protocol-major-version: 000b
                                   protocol-minor-version: 0000
                                         release-number: 00000003
                                       resource-id-base: 00700000
                                       resource-id-mask: 000fffff
                                     motion-buffer-size: 00000000
                                       image-byte-order: MSB first
                                   bitmap-format-bit-order: MSB first
                                   bitmap-format-scanline-unit: 20
                                   bitmap-format-scanline-pad: 20
                                            min-keycode: 8 (^H)
                                            max-keycode: 129 (\201)
                                                 vendor: ”MIT X
                                                   Consortium”
                                         pixmap-formats: (2)
                                                  roots: (2)
0.06: Client -->   72 bytes
        ............REQUEST: CreateGC
         graphic-context-id: GXC 00700000
                   drawable: DWB 0008006d
                 value-mask: foreground | background
                 value-list:
                        foreground: 00000001
                        background: 00000000
```

(continued)

```
        ............REQUEST: CreateGC
         graphic-context-id: GXC 00700001
                   drawable: DWB 00080070
                 value-mask: foreground | background
                 value-list:
                         foreground: 00000001
                         background: 00000000
        ............REQUEST: GetProperty
                     delete: False
                     window: WIN 0008006d
                   property: <RESOURCE_MANAGER>
                       type: <STRING>
                long-offset: 00000000
0.08:                                    1512 bytes <-- X11 Server
                                   ..............REPLY: GetProperty
                                                 format: 08
                                                   type: <STRING>
                                            bytes-after: 00000000
0.18: Client -->   28 bytes
        ............REQUEST: InternAtom
             only-if-exists: False
                       name: ”WM_CONFIGURE_DENIED”
0.18:                                      32 bytes <-- X11 Server
                                    ..............REPLY: InternAtom
                                                 atom: ATM 00000049
0.20: Client -->   16 bytes
        ............REQUEST: InternAtom
             only-if-exists: False
                       name: ”WM_MOVED”

0.20:                                      32 bytes <-- X11 Server
                                    ..............REPLY: InternAtom
                                                 atom: ATM 0000004c



0.22: Client -->   32 bytes
        ............REQUEST: OpenFont
                    font-id: FNT 00700002
                       name: ”fixed”
        ............REQUEST: QueryFont
                       font: FTB 00700002
```

(continued)

```
0.24:                                    1644 bytes <-- X11 Server
                                   ..............REPLY: QueryFont
                                            min-bounds:
                                            max-bounds:
                                    min-char-or-byte2: 0000
                                    max-char-or-byte2: 007f
                                         default-char: 0000
                                       draw-direction: LeftToRight
                                            min-byte1: 00
                                            max-byte1: 00
                                     all-chars-exist: True
                                          font-ascent: 10
                                         font-descent: 3
                                           properties: (6)
                                           char-infos: (128)

0.28: Client -->   44 bytes
        ............REQUEST: CreateGC
         graphic-context-id: GXC 00700003
                   drawable: DWB 0008006d
                 value-mask: foreground | background | font
                 value-list:
                         foreground: 00000001
                         background: 00000000
                               font: FNT 00700002
        ............REQUEST: NoOperation
        ............REQUEST: QueryBestSize
                      class: Stipple
                   drawable: DWB 0008006d
                      width: 0020
                     height: 0020
0.30:                                      32 bytes <-- X11 Server
                                ..............REPLY: QueryBestSize
                                                     width: 0020
                                                    height: 0020
0.34: Client -->  512 bytes
        ............REQUEST: CreatePixmap
                      depth: 01
                  pixmap-id: PXM 00700004
                   drawable: DWB 0008006d
                      width: 0020
                     height: 0020
```

(continued)

```
............REQUEST: CreateGC
         graphic-context-id: GXC 00700005
                   drawable: DWB 00700004
                 value-mask: foreground | background
                 value-list:
                         foreground: 00000001
                         background: 00000000
         ............REQUEST: PutImage
                     format: Bitmap
                   drawable: DWB 00700004
                         gc: GXC 00700005
                      width: 0020
                     height: 0020
                      dst-x: 0
                      dst-y: 0
                   left-pad: 00
                      depth: 01
              data: aa aa aa aa 55 55 55 55 aa aa aa aa 55 55 55 55
         ............REQUEST: FreeGC
                         gc: GXC 00700005
         ............REQUEST: CreateGC
         graphic-context-id: GXC 00700006
                   drawable: DWB 0008006d
        value-mask: foreground | background | fill-style | tile | font
                 value-list:
                         foreground: 00000001
                         background: 00000000
                         fill-style: Tiled
                               tile: PXM 00700004
                               font: FNT 00700002
         ............REQUEST: NoOperation
         ............REQUEST: CreatePixmap
                      depth: 01
                  pixmap-id: PXM 00700007
                   drawable: DWB 0008006d
                      width: 0040
                     height: 0040
         ............REQUEST: CreateGC
         graphic-context-id: GXC 00700008
                   drawable: DWB 00700007
                 value-mask: foreground | background
```

(continued)

```
                   value-list:
                           foreground: 00000001
                           background: 00000000
0.40: Client -->  320 bytes
        ............REQUEST: PutImage
                     format: Bitmap
                   drawable: DWB 00700007
                         gc: GXC 00700008
                      width: 0040
                     height: 0040
                      dst-x: 0
                      dst-y: 0
                   left-pad: 00
                      depth: 01
              data: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        ............REQUEST: FreeGC
                         gc: GXC 00700008
        ............REQUEST: GetGeometry
                   drawable: DWB 00700007
```

Example Files [C]

This appendix contains examples of `.login`, `.cshrc`, `.twmrc`, `.Xresources`, and `.xinitrc` files. Note that the `.login`, `.Xresources`, and `.xinitrc` files should be located on the front-end system, not the Cray Research system. These files were tested on a Sun Workstation. Although efforts were made to make them usable on other Sun servers and other platforms, differences in environments might require modifications.

Example `.login` file
C.1

The following example shows a `.login` file:

```
umask 022
set ignoreeof
setenv host ‘hostname‘

set localpath=(/usr/ucb /usr/bin /bin /usr/local/bin ~/bin .)

if (‘tty‘ == ”/dev/console”) then
   setenv TERM ‘tset – Q – ’?vt100’‘
   switch ( $TERM )
      case ”xterm”:     # X Windowing System
           setenv DISPLAY ”${host}:0.0”
           set path=( /usr/bin/X11 $localpath )
           xinit
           kbd_mode –a
      case ”vt100”:
           set path=( $localpath )
      default:
           exit
           breaksw
      endsw
```

(continued)

```
else if (”$term” == ”dialup” || ”$term” == ”network” || \
        ”$term” == ”unknown”) then
        setenv TERM ‘tset – Q – ’?vt100’‘
        switch ( $TERM )
           case ”vt100”:
                set path=( $localpath )
           default:
                exit
                breaksw
        endsw
     endif
endif
```

Example .cshrc file
C.2

The following example shows a `.cshrc` file:

```
#
set filec
set history=32
set host=‘hostname‘
#
#display prompt in bold
#The sequence ^[ is the escape character
#
set prompt=”^[[5m${host} ! % ^[[m”
#
alias lsf ”ls –CF”
alias lsl ”ls –lg”
if ($TERM == ”xterm || $TERM ==”xterms”) then
   #
   #alias to set title bar to a string
   #alias to set icon label to a string
   #the sequence ^G is a control–G
   #
  alias label ‘echo –n ”^[]2;\\!*^G”‘
  alias icon_label ‘echo –n ”^[]1;\\!*^G”‘
```

(continued)

```
  #
  #set title bar and icon label to directory stack
  #
alias cd ‘cd \\!*; label ${host}:’dirs’‘
alias pushd ‘pushd \\!*; label ${host}:’dirs’‘
alias popd ‘popd \\!*; label ${host}:’dirs’‘
  #
  #first time set up
  #
label ${host}:’dirs’
icon_label ${host}
 endif
```

Example .twmrc file C.3

The .twmrc file manipulates windows on a workstation. The following example shows a .twmrc file:

```
NoTitleFocus
WarpCursor
BorderWidth     2
TitleFont       ”–adobe–courier–bold–r–*––12–120–75–75–m–70–*–*”
MenuFont        ”9x15”
IconFont        ”9x15”
ResizeFont      ”9x15”

#Button = KEYS : CONTEXT : FUNCTION
#–\|––\|––\|––\|––\|––\|––\|––\|––\|––\|––\|––\|––\|––\|––\|––\|––\|––\|–
Button1 =       : root    : f.menu ”button1”
Button2 =       : root    : f.menu ”button2”
Button3 =       : root    : f.menu ”button3”
Button1 =  m    : root    : f.menu ”button1”
Button2 =  m    : root    : f.menu ”button2”
Button3 =  m    : root    : f.menu ”button3”
```

(continued)

```
Button1 =      : icon    : f.lower
Button2 =      : icon    : f.iconify
Button3 =      : icon    : f.move
Button1 =  m   : icon    : f.lower
Button2 =  m   : icon    : f.iconify
Button3 =  m   : icon    : f.move
Button1 =  m   : window  : f.lower
Button2 =  m   : window  : f.iconify
Button3 =  m   : window  : f.move
Button1 =  m | s : window  : f.lower
Button2 =  m | s : window  : f.resize
Button3 =  m | s : window  : f.raise
Button1 =  m   : title   : f.menu ”button1”
Button2 =  m   : title   : f.menu ”button2”
Button3 =  m   : title   : f.menu ”button3”
Button1 =      : title   : f.raise
Button2 =      : title   : f.move
Button3 =      : title   : f.lower
”R2”    =      : icon     : f.lower
”R5”    =      : icon     : f.raise
”R2”    =      : window   : f.lower
”R5”    =      : window   : f.raise
NoTitle
{
  ”xclock”
  ”xping”
  ”xload”
  ”xcalc”
}

menu ”button1”
{
”Other Logins”  f.title
”xterm”    !”xterm –T xterm –geometry 80x34 &”
”sn218”    !”xterm –T sn218 –geometry 80x34 ’#+755+1’ –e telnet sn218
&”
”sn1001”   !”xterm –T sn1001 –geometry 80x34 ’#+845+1’ –e telnet
sn1001 &”
”sn2025”   !”xterm –T sn2025 –geometry 80x34 ’#+1020+1’ –e telnet
sn2025 &”
}
```

(continued)

```
menu ”button2”
{
”Window Ops”          f.title
”Refresh”             f.refresh
”Focus on Root”       f.unfocus
”Source .twmrc”       f.twmrc
”twm Version”         f.version
”(De)Iconify”         f.iconify
”Move Window”         f.move
”Resize Window”       f.resize
”Raise Window”        f.raise
”Lower Window”        f.lower
”Focus on Window”     f.focus
”Destroy Window”      f.destroy
”Kill twm”            f.quit
}
```

Example .Xresources file

C.4

The `.Xresources` file contains client defaults which may be used as input to `xrdb` as shown in the `xinitrc` example on page 65. The following example shows a `.Xresources` file with definitions useful for `csh` and `vi`:

```
*VT100.translations:  #override \n\
                   <Key>L2:string(”history 30”) string(0x0d) \n\
                   <Key>L3:string(”pushd”) string(0x0d) \n\
                   <Key>L4:string(”popd”) string(0x0d) \n\
                   <Key>L5:string(”dirs”) string(0x0d) \n\
                   <Key>L6:string(”lsf”) string(0x0d) \n\
                   <Key>L7:string(”pwd”) string(0x0d) \n\
                   <Key>L8:string(”xrefresh”) string(0x0d) \n\
                   <Key>L9:string(”clear”) string(0x0d) \n\
                   <Key>L10:string(”!!”) string(0x0d) \n\
                   <Key>F1:string(”:q”) string(0x0d) \n\
                   <Key>F2:string(”:w”) string(0x0d) \n\
                   <Key>F3:string(”ZZ”) \n\
```

(continued)

```
                   <Key>F4:string(”:!cc –c –g %”) string(0x0d) \n\
                   <Key>F5:string(”!vi”) string(0x0d) \n\
                   <Key>F6:string(”!make”) string(0x0d) \n\
                   <Key>R7:string(0x15) \n\
                   <Key>R9:string(0x04) \n\
                   <Key>R13:string(0x02) \n\
                   <Key>R15:string(0x06) \n\
 ~Meta Shift  Ctrl <Btn2Down>:mode–menu() \n\
 ~Meta Shift  Ctrl <Btn1Down>:mode–menu() \n\
 ~Meta ~Lock ~Ctrl <Btn1Down>:select–start() \n\
 ~Meta ~Lock ~Ctrl <Btn1Motion>:select–extend() \n\
 ~Ctrl ~Meta       <Btn2Down>:ignore() \n\
 ~Meta            <Btn2Up>:insert–selection(PRIMARY,CUT_BUFFER0) \n\
 ~Ctrl ~Meta       <Btn3Down>:start–extend() \n\
 ~Meta            <Btn3Motion>:select–extend() \n\
 ~Meta            <BtnUp>:select–end(PRIMARY,CUT_BUFFER0) \n\
                   <BtnDown>:bell(0)
xbiff.ReverseVideo:     off
xtroff*geometry:        –100+0
xtroff*scrollbar:       off
xman*topBox.geometry    102x74+643+772
xterm.ReverseVideo:     off
xterm*SaveLines:        330
xterm*TitleBar:         on
xterm*font:         –adobe–courier–medium–r–*––12–120–75–75–m–70–*–*
xterm*boldFont:     –adobe–courier–bold–r–*––12–120–75–75–m–70–*–*
```

Example .xinitrc file
C.5

The .xinitrc file starts the X Window System environment. The following example shows a .xinitrc file:

```
set host=‘hostname‘
xrdb –load .Xresources &
twm &
#
#  Set up for screen 1
#
xterm –geometry 80x34+0+0 –fg white –bg skyblue –display $host:0.1&
xterm –geometry 80x34–0+0 –fg white –bg maroon –display $host:0.1&
#
#  Set up for screen 0
#
xclock –geometry 0+0 &
xterm –geometry 80x60+0–0 –display $host:0.0&
xterm –geometry 80x55–0–0 –display $host:0.0&
#
#  Last item must not run in background
#
xterm –C –geometry 80x10–0+0
```