X Windows Version 11.5
A Concise Description

Tim Love - CUED

November 2, 1994

This document is an introduction to the X Windows system particularly from the point of view of a programmer in C. It does not constitute a rigorous definition of X – in particular the color and internationalisation features of X11R5 are not described – but instead aims to give a balanced view of the whole system and describe some common programming techniques.
We would be most grateful to hear of any errors you might find in this document, or any ways you think it might be clarified or improved. Contact `tpl@eng.cam.ac.uk`

Contents

List of Examples

Program	**Page**	**Description**
demo1.c	19	lines, text
demo2.c	20	rectangles, cursors
ellipse_demo.c	26	pixmaps
keyboard_demo.c	27	keyboard events
easy_color_demo.c	31	colors
selections.c	39	selections
color_demo.c	55	advanced color
xcolors.c	(online in `/export/Examples/X`)	a complete utility
XR5demo.c	(online in `/export/Examples/X`)	X11 R5 features

List of Figures

1 Overview of X Windows

X11 Release 5 came out in August 1991. The heart of the *X*[1] Windows system consists of a program called *X* which runs on a machine with a display, keyboard, and a mouse. It waits for other programs to tell it what to do or for something to happen to the pointer or keyboard. The programs can be running on the same machine as *X* is or elsewhere on the network, maybe on a machine that hasn't even got a display of its own. This 'network transparency' is one of the strengths of *X*. Graphics programs only have to know about *X*, not about the special low level graphics commands for each type of machine. The client programs communicate to *X*, the server, via a Protocol language that is common across machine types.

All a client program needs to do to use the *X* display is to open up a connection with the server and then send Protocol requests to it. To simplify sending these, an extensive library of about 200 display subroutines is provided and it is this library, Xlib, which this document mainly describes.

Many client programs can simultaneously use the same *X* server. To save each client having its own copy of fonts, color information, etc, (thereby wasting space and causing more data to be passed via the network), the server stores data on behalf of the clients, allowing sharing wherever possible. In order to enable the client to reference these resources the server provides resource codes and these can be used in many of the routines to specify that certain data is to be used.

X is 'event driven'. For each window you create you can select what sort of events (key presses, re-exposure, etc) you want it to respond to. Typically, an X program consists of a set-up sequence followed by an 'event-loop' which waits for events to be reported by the server, determines what sort of event has happened and in which window, then processes the event.

There are 4 types of messages passing between the client and server;

- Requests - the client can ask the server to draw something, or ask for information.
- Replies - the server can reply.
- Events - the server can surprize the client with something
- Errors - the server can report an error

Both display requests and events are buffered and the server executes asynchronously much of the time to maximise efficient use of the network. When the client wishes to establish synchronization it often has to ask for it, though working in synchronized mode can be up to 30 times slower, so only use it for debugging.

X clients programs should be written mindful of the fact that other clients are likely to be running at the same time. Windows should be prepared to be covered over by others then exposed, and they have to redraw themselves. Neither should clients indescriminately use scarce resources like off-screen memory or monopolise the keyboard or pointer. If they can accept cut-and-paste operations, so much the better.

One application that is always likely to be running is a 'window manager'; a program which allows you to resize, move and re-stack windows, pop menus up,

[1]X Window System is a trademark of MIT. Copyright 1989 by the Massachusetts Institute of Technology. Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Figure 1: X system overview

etc. Any *X* application you write will need to work under a window manager. Some window managers are more bossy than others (some won't let you raise windows, for instance), but there are X commands so that a client can at least make suggestions to the window manager.

X can support one or more screens containing overlapping (sub)windows and works on many types of hardware but it doesn't provide high level support. If you want to prefabricate dialog boxes from pre-defined scrollbars, buttons, etc then you should use a 'toolkit' that sits on top of *X*. See the Motif manuals and on-line help for details.

X is extensible. Release 4 of *X* included the `SHAPE` extension which provides non-rectangular, disjoint windows. Neither the use or the writing of such extensions will be covered in this document.

1.1 Using X from a Terminal

There are a number of window managers, terminal emulators and graphics programs supplied with X which enable the user to create text windows and move them about on the screen. All these facilities are described in section (1) of the `Unix` [2] manual.

Program	Function
bitmap	bitmap editor
xterm	terminal emulator
hpterm	HP's terminal emulator
mwm	Motif Window Manager
twm	X Window Manager
xwd	window image dumper.
xwud	window image undumper
xpr	print X window dump
xclock	analog / digital clock
xfc	font displayer
xfd	font displayer
xhost	X window system access control program
xload	load average display
xrefresh	refresh all windows on the screen.
xset	X window system user setup program
xsetroot	sets the root window properties
xdvi	DVI file Previewer
xmodmap	sets the keyboard mapping
xcolors	color displayer
xfonts	font displayer
xcolormix	RGB color blender
xcursors	cursor displayer

To investigate the *X* system on your machine you might first like to try

Program	Function
xdpyinfo	returns info about X
xset q	returns info on mouse set-up, etc
xshowcmap	displays current colors
xgc	a menu-driven X tester

1.2 How to use X in a C program

From the programmer's point of view X consists of a library of subroutines which he may call in order to perform specific functions on the display. These functions are contained in a library, `/usr/lib/libX11.a`, (or `/usr/lib/X11R5/libX11.sl` on HPs) which must be linked into the program by the loader when the executable version of the program is generated. This can usually be done by adding `-lX11` to the end of the compilation command and, on HPs, `-L/usr/lib/X11R5` to the start. The Xlib header file `X11/Xlib.h` (or `/usr/include/X11R5/Xlib.h` on HPs)must also be included at the top of every program file which uses the X library. On the HPs this requires the addition of `-I/usr/include/X11R5`

The coordinate system for *X* windows has its origin at the top left hand corner of any given window. The x coordinate increases from left to right across the page and the y coordinate increases from top to bottom. The global coordinate system is effectively that of the `RootWindow` which covers the whole of the screen. The

[2] Unix is a trademark of AT&T Bell Laboratories.

scaling of coordinates on an X display is never changed, however operations in each window are performed relative to that window's origin.

X functions which are described as returning `Status` produce a non zero value if they have executed successfully and return zero if they have failed. Other routines which return a pointer to something on success usually return the `NULL` pointer on failure.

To save space, we have only declared arguments whose type or use is not self evident. Most particularly, the following variables are not declared :-

```
Display *display;
Window w;
Drawable d;   /* either a window you can draw into or
                 off-screen memory (a pixmap) */
GC gc;        /* a Graphics Context, where current font,
                 foreground color, etc is contained*/
Colormap cmap;
Region r;
unsigned long mask,valuemask;
```

There is a standard order for arguments :-

```
display,resources,gc,
x,y,w,h,
array,array_length,
mask,struct
```

1.3 Useful Online facilities

Some of the most useful information on the X library can be found in the include files `Xlib.h` and `X.h`. These files are usually found in `/usr/include/X11` or `/usr/include/X11R5`.

There are several example programs in this document. More comprehensive examples are in `/export/Examples/X` on the CUED teaching system and in the `X` source tree (`/usr/src/X11R4` on the CUED teaching system). Even more examples are available via `ftp` at

```
ftp.uu.net:/published/oreilly/xbook/xlib
```

Manpages for the routines are online.

The online help facility at CUED contains a file of answers to frequently asked questions and some discussion of advanced topics.

Relevant newsgroups are:- `comp.windows.x`, `comp.sources.x`

1.4 Useful Books and Manuals

C Language X Interface Protocol	J. Gettys, R. Newman, R. W. Scheifler.	
The C Programming Language.	B. W. Kernigan, D. M. Ritchie.	Prentice-Hall.
The UNIX System	S. R. Bourne.	Addison-Wesley.
Programming with Xlib		Hewlett Packard.
Xlib Programming Manual	Adrian Nye.	O'Reilly and Associates.
Xlib Reference Manual	Adrian Nye.	O'Reilly and Associates.

Acknowledgments

David Bijl, Tim Marsland and Richard Prager have all made useful comments and suggestions.

2 Starting Up

2.1 Opening a Display Server

Before any windows can be used it is necessary to establish a connection with a display server. A server is identified by the name of the machine it is on and the display number on that machine. If the server name string is `NULL` then the contents of the `DISPLAY` environment variable is substituted to determine which display to use. The full display name has the form `hostname:display_number.screen`. Multiple screens/terminals can be controlled by a single X server as long as they are on the same display; i.e. they are controlled with a single mouse and a single keyboard.

```
Display *XOpenDisplay(name)  /* open a display on the named server.
                                If successful  then all the screens on
                                the display may be used . */
    char *name;              /* e.g. mit-athena:0.0*/

XCloseDisplay(display)       /* close a display completely */
```

When a client's connection to a display is closed, the resources held by the server solely on the client's behalf are freed. When the last connection to the X server is closed, more resources are freed and the server does a reset.

2.2 Finding out about the display

Some simple routines can be used to discover general characteristics of the display. These shall be described in some detail as an introduction to some X concepts.

```
int DefaultScreen(display)  /* This returns the default screen number; the only */
                             /* one you're likely to use */

int ScreenCount(display)    /* This returns the number of possible screens */

int  DefaultDepth()         /* A graphics display can be thought of as overlying
                              bitplanes where only one bit per plane affects the
                              color of a pixel. This routine returns the number
                              of planes used */

unsigned long AllPlanes()   /* The bits in the returned long
                             represent the planes used */

int DisplayPlanes(display, screen_number) /* returns the number of possible planes*/

Colormap DefaultColormap(display, screen_number) /* Information on the current
                       colors is stored in a table called a colormap */

int DisplayCells(display, screen_number) /* returns number of entries in the
                                          default colormap */

unsigned long BlackPixel(display, screen_number) /* There are permanently */
unsigned long WhitePixel(display, screen_number) /* allocated entries in
    the default colormap for black and white. These routines return
    longs that are used to reference these colors */

GC DefaultGC(display, screen_number)  /* Graphics commands get details about
    the current  foreground color, etc, from a Graphics Context structure.
    This  routine returns the default one */

Window DefaultRootWindow(display)     /* This returns the initial window that
                                       covers the screen */

Visual *DefaultVisual(display,screen_number) /* So that various kinds of hardware
    can be supported ‘invisibly’, some  hardware-specific information
    is stored in the visual structure*/

int DisplayHeight(display,screen_number)    /* height of display in pixels */

int DisplayWidth(display,screen_number)     /* width of display in pixels */
```

2.3 Creating & Destroying Windows

Before drawing or writing on the screen each program which is using the server creates one or more windows. The windows exist in a tree structure growing down from the `RootWindow` which covers the whole display.

Many applications only use windows which have root as the parent. If you create a grandchild window you will find that the baby window behaves as if its parent constituted the whole physical display; all output to the baby window is clipped at the boundaries of its parent.

No window appears on the display until it is mapped, all its ancestors are mapped and an expose event has been received. Furthermore the mapping command must reach the server before anything will happen. When you map a window you must either deliberately flush all buffers (see the section on Events for details of `XFlush`) or draw something on the display to push your mapping command across the network. The subroutines for mapping windows are given later in this section. Creating a window alone is not sufficient to enable you to actually see anything on the display.

If you don’t want to bother setting up too many things, you can create windows that inherit their attributes from their parent.

```
Window XCreateSimpleWindow
(display,parent, x, y, width, height, borderwidth, border,background)
    Display *display           /* The value returned by XOpenDisplay     */
    Window parent;             /* parent window i.d. probably RootWindow */
    int x, y;                  /* specify top LH corner of border        */
    unsigned int width, height; /* specify internal dimensions   */
    unsigned int border;       /* border pixel */
    unsigned long background;  /* background pixel */
```

For more sophisticated windows use,

```
Window XCreateWindow
(display,parent,x,y,width,height,border_width,depth,class,visual,valuemask,attributes)
    Window parent;               /* parent window i.d. probably RootWindow */
    int x, y;                    /* specify top LH corner of border */
    unsigned int width, height;  /* specify internal dimensions */
    unsigned int borderwidth;    /* border width */
    int depth;
    unsigned int class;          /*InputOutput,InputOnly,CopyFromParent*/
                                 /*(InputOnly windows are transparent)*/
    Visual *visual;              /*CopyFromParent, etc */
    unsigned long valuemask;     /* selects attribute fields */
    XSetWindowAttributes *attributes; /* contains the attributes */
```

The following structure is used when initialising windows.

```
typedef struct {
                                /* DEFAULT            MASK              */
Pixmap background_pixmap;       /* None               CWBackPixmap      */
unsigned long background_pixel; /* -                  CWBackPixel       */
Pixmap border_pixmap;           /* CopyFromParent     CWBorderPixmap    */
unsigned long border_pixel;     /* -                  CWBorderPixel     */
int bit_gravity;                /* ForgetGravity      CWBitGravity      */
int win_gravity;                /* NorthWestGravity   CWWinGravity      */
int backing_store;              /* NotUseful          CWBackingStore    */
unsigned long backing_planes;   /* ~0                 CWBackingPlanes   */
unsigned long backing_pixel;    /* 0                  CWBackingPixel    */
Bool save_under;                /* False              CWSaveUnder       */
long event_mask;                /* 0                  CWEventMask       */
long do_not_propagate_mask;     /* 0                  CWDontPropagate   */
Bool override_redirect;         /* False              CWOverrideRedirect */
Colormap colormap;              /* CopyFromParent     CWColormap        */
Cursor cursor;                  /* None               CWCursor          */
} XSetWindowAttributes;
```

`background_pixmap`:- This lets you have a patterned background. It must have the same root and depth as the window. If `None` then there's no background. If `ParentRelative`, the parent's background is used but the parent and child must have the same depth.

`border_pixmap`:- This and the window must have the same root and depth (else a `BadMatch` error). If `CopyFromParent` the parent's pixmap is copied

`bit_gravity`:- Defines which region is retained should the window be resized.

`win_gravity`:- Defines how the window is re-positioned should its parent be resized.

`backing_store`:- `WhenMapped` advises the server to save the contents of the window if it's unmapped or obscured. `Always` advises the server that saving the contents even when unmapped would be useful. Not all servers support backing store. Use `DoesBackingStore()` to find out.

`backing_planes`:- Defines which planes will be saved.

`backing_pixel`:- Defines what value to put in planes not saved.

`save_under`:- `True` advises the server to save regions obscured by the window.

`do_not_propagate_mask`:- Defines which events should not be propagated to ancestors.

`override_redirect`:- If `True` then the window manager won't know about the window.

`colormap`:- This must have the same visual type as the window. If `CopyFromParent` the parent's colormap is copied

There are a couple of calls which can be used to destroy windows. Windows are unmapped automatically before they are destroyed.

```
XDestroyWindow(display,w)  /* destroy the window with the given i.d. */|

XDestroySubwindows(display,w)
    Window w;  /* destroys all the subwindows of the specified window */
               /* the window itself is not destroyed                  */
```

Attributes of a window are returned using the following call and structure :-

```
Status XGetWindowAttributes(display,w,window_attributes)
   XWindowAttributes *window_attributes;  /* RETURN completed structure*/
```

```
typedef struct {
    int x, y;                   /* location of window */
    int width, height;          /* width and height of window */
    int border_width;           /* border width of window */
    int depth;                  /* depth of window */
    Visual *visual;             /* the associated visual structure */
    Window root;                /* root of screen containing window */
    int class;                  /* InputOutput, InputOnly*/
    int bit_gravity;            /* one of bit gravity values */
    int win_gravity;            /* one of the window gravity values */
    int backing_store;          /* NotUseful, WhenMapped, Always */
    unsigned long backing_planes;/* planes to be preserved if possible */
    unsigned long backing_pixel;/* value to be used when restoring planes */
    Bool save_under;            /* should bits under be saved? */
    Colormap colormap;          /* color map associated with window */
    Bool map_installed;         /* is color map currently installed*/
    int map_state;              /* IsUnmapped, IsUnviewable, IsViewable */
    long all_event_masks;       /* events that interest all people */
    long your_event_mask;       /* my event mask */
    long do_not_propagate_mask; /* events that should not propagate */
    Bool override_redirect;     /* boolean value for override-redirect */
} XWindowAttributes;
```

2.4 Mapping, Moving & Uncovering Windows

Map windows using

```
XMapWindow(display,w)
XMapSubwindows(display,w)
XUnmapWindow(display,w)
XUnmapSubwindows(display,w)
```

There are also a few fairly self-explanatory calls which can be used to move the windows around and shuffle them to the top and bottom of the pile like papers on a desk. The window manager may well interfere with your intentions, though – it's up to the user where top-level windows are, and which is on top.

```
XChangeWindowAttributes(display,w,valuemask,attributes)
    XSetWindowAttributes *attributes;

XConfigureWindow(display,w, x, y, valuemask, values)
    unsigned int valuemask    /*  to select relevant fields of value   */
    XWindowChanges *values;    /* contains new details      */

XResizeWindow(display,w,width,height);
XMoveResizeWindow(display,w,x,y,width,height);
XSetWindowBorderWidth(display,w,width);
XRaiseWindow(display,w)       /* raise the window to the front */
XLowerWindow(display,w)       /* lower window to the back      */
XMoveWindow(display,w, x, y)  /* move and raise w   */
XCirculateSubwindows(display,w,direction);
                              /* direction is RaiseLowest or LowerHighest*/
XCirculateSubwindowsUp(display,w)      /* raises the lowest mapped child */
XCirculateSubwindowsDown(display,w)    /* lowers the highest mapped child*/
```

2.5 Defining Bitmaps and Pixmaps

Off screen memory (a pixmap) is often used to define images for later use in graphics operations. Pixmaps are also used to define tiles, or patterns, for use as window backgrounds, borders, or cursors. A single bit plane pixmap is sometimes referred to as a bitmap.

To create a Pixmap, use

```
Pixmap XCreatePixmap(display,d,width,height,depth)
   unsigned int width, height, depth;
```

To read a bitmap from a file (created by the `bitmap` program, perhaps), use

```
int XReadBitmapFile(display,d,filename,width,height,bitmap,x_hot,y_hot)
     /* possible return values are BitmapOpenFailed, BitmapFileInvalid
                                  BitmapNoMemory, BitmapSuccess */
   int *width, *height;    /*RETURN*/
   Pixmap *bitmap;         /*RETURN*/
   int *x_hot,*y_hot;       /*RETURN*/
```

To write a bitmap to a file, use

```
int XWriteBitmapFile(display,filename,bitmap,width,height,x_hot,y_hot)
     /* possible return values are BitmapOpenFailed, BitmapNoMemory
                                  BitmapSuccess */
   int width, height;
   Pixmap bitmap;
   int x_hot,y_hot;
```

If you `#include` a bitmap file in your program you can use

```
Pixmap XCreateBitmapFromData(display,d,data,width,height)
   char *data
```

Pixmaps consume a lot of memory. Use

```
XFreePixmap(display,pixmap)
```

as soon as possible to release memory.

3 Graphics Context

Text and Graphics output routines operate within a graphics context (GC); a structure containing current settings (foreground color, font, etc) for drawing operations. This not only reduces the number of arguments that these routines require, but also (since the GC is held in the server) reduces traffic. .

```
typedef struct {
                          /*DEFAULTS*/               /*MASKS*/
int function;             GXcopy                     GCFunction
unsigned long plane_mask; ~0                         GCPlaneMask
unsigned long foreground; 0                          GCForeground
unsigned long background; 1                          GCBackground
int line_width;           0                          GCLineWidth
int line_style;           LineSolid                  GCLineStyle
int cap_style;            CapButt                    GCCapStyle
int join_style;           JoinMiter                  GCJoinStyle
int fill_style;           FillSolid                  GCFillStyle
int fill_rule;            EvenOddRule                GCFillRule
int arc_mode;              ArcPieSlice            GCArcMode
Pixmap tile;               a pixmap in the fg color  GCTile
Pixmap stipple;            a pixmap  of 1's       GCStipple
int ts_x_origin;           0                      GCTileStipXOrigin
int ts_y_origin;           0                      GCTileStipYOrigin
Font font;                                        GCFont
int subwindow_mode;        ClipByChildren         GCSubwindowMode
Bool graphics_exposures;   True                   GCGraphicsExposures
int clip_x_origin;         0                      GCClipXOrigin
int clip_y_origin;         0                      GCClipYOrigin
Pixmap clip_mask;          None                   GCClipMask
int dash_offset;           0                      GCDashOffset
char dashes;               4                      GCDashList
} XGCValues;
```

These components are quite involved.

function:- When something is drawn on the screen the machine takes the pixel value being drawn and combines it with the pixel which is already on the screen using a display function. The display function produces an output pixel value which becomes the new pixel for that particular point on the screen. If the old pixel value on the screen is ”dst” and the pixel being drawn is ”src” the following table lists the effects of the various display functions available.

Function Name	Operation
GXclear	0
GXand	src AND dst
GXandReverse	src AND NOT dst
GXcopy	src
GXandInverted	(NOT src) AND dst
GXnoop	dst
GXxor	src XOR dst
GXor	src OR dst
GXnor	(NOT src) AND NOT dst
GXequiv	(NOT src) XOR dst
GXinvert	NOT dst
GXorReverse	src OR NOT dst
GXcopyInverted	NOT src
GXorInverted	(NOT src) OR dst
GXnand	(NOT src) OR NOT dst
GXset	1

line_style:- `LineSolid`, `LineDoubleDash`, `LineOnOffDash`

cap_style:- `CapNotLast`, `CapButt`, `CapRound`, `CapProjecting`

join_style:- `JoinMiter`, `JoinRound`, `JoinBevel`

fill_style:- `FillSolid`,`FillTiled`, `FillOpaqueStippled`

fill_rule:- `EvenOddRule`, `WindingRule` (rules determining how to fill a complex polygon.

arc_mode:- `ArcPieSlice`, `ArcChord` (rules determining how to fill an arc).

stipple:- This is a bitmap for use in shading operations.

ts_x_origin, ts_y_origin:- These are the offsets for tile/stipple operations.

subwindow_mode:- `ClipByChildren`, `IncludeInferiors`

graphics_exposures:- A Boolean; should expose events be generated if an attempt is made to draw to a hidden area?

To create a GC use

```
GC XCreateGC(display,d,valuemask,values)
   XGCValues *values
```

Note that although you need to provide a drawable, that doesn't mean that the GC can only be used for operations in that drawable – you can use it for any drawable that's on the same screen and has the same depth as the original drawable.

To change or free GC's, use

```
XCopyGC(display,src,valuemask,dest)
   GC src,dest

XChangeGC(display,gc,valuemask,values)

XFreeGC(display,gc)
```

The GC contents can be changed singley or en masse using these routines

```
XSetState (display,gc,foreground,background,function,planemask)
  This is a packaging of the following 4 routines;
XSetForeground(display,gc,foreground)
XSetBackground(display,gc,background)
XSetFunction(display,gc,function)
XSetPlaneMask(display,gc,planemask)

XSetLineAttributes(display,gc,line_width,linestyle,cap_style,join_style)
XSetDashes        (display,gc,dash_offset,dash_list,n)
XSetFillStyle     (display,gc,fill_style)
XSetFillRule      (display,gc,fill_rule)
XQueryBestSize    (display,class,d,width,height,rwidth,rheight)
XQueryBestTile    (display,gc,d,width,height,rwidth,rheight)
XQueryBestStipple (display,gc,d,width,height,rwidth,rheight)
XSetTile          (display,gc,tile)
XSetStipple       (display,gc,stipple)
XSetFont          (display,gc,font)
XSetClipOrigin    (display,gc,clip_x_origin,clip_y_origin)
XSetClipMask      (display,gc,pixmap
XSetClipRectangles(display,gc,clip_x_origin,clip_y_origin,rectangles,n,ordering)
XSetArcMode       (display,gc,arc_mode)
XSetSubwindowMode (display,gc,subwindowmode);
XSetGraphicsExposures(display,gc,graphics_exposures);
XSetRegion(display, gc, r);
```

Many of these routines you might never use. See the manpages if you're interested.

Xlib caches the contents of the GC. The values in the cache can be requested using

```
Status XGetGCValues(display, gc, valuemask, values_return)
XGCValues *values_return;
```

but note :-

- The values in the cache may not quite be the currently installed values.
- Unless the default values for `CFont`, `GCTile`, and `GCStipple` have been overridden, their values cannot be returned.
- The clip mask and dash list values cannot be returned.

4 Rectangles, lines, dots, arcs and text

First, some elemental structures.

```
typedef struct
{
    short x, y;
} XPoint;

typedef struct
{
    short x1, y1,x2,y2;
} XSegment;

typedef struct
{
    short x, y;
    unsigned short width, height;
} XRectangle;

typedef struct
{
    short x, y;
    unsigned short width, height;
    short angle1,angle2;
} XArc;
```

The following routines make extensive use of the GC (refer to the manual for further information on exactly which components are relevant). Lines of width 0 come out with a thickness of 1 drawn using a different, faster, algorithm.

They write to `drawables` (pixmaps or windows that can accept graphic input).

```
XDrawPoint(display,d,gc,x,y)
XDrawPoints(display,d,gc,points,npoints,mode)
     int mode; /* CoordModeOrigin or CoordModePrevious */

XDrawLine(display,d,gc,x1, y1, x2, y2)
XDrawLines(display,d,gc,points.npoints,mode)
     int mode; /* CoordModeOrigin or CoordModePrevious */

XDrawSegments(display,d,gc,segments,nsegments)

XDrawRectangle(display,d,gc,x,y,width,height)
XDrawRectangles(display,d,gc,rectangles,nrectangles)

XDrawArc(display,d,gc,x,y,width,height,angle1,angle2)
    unsigned int width,height /*major and minor axes of arc*/
    int angle1 /*start of arc, clockwise from 3 o'clock,scaled by 64 */
    int angle2 /*path and extent of arc, relative to angle1, scaled by 64*/
XDrawArcs(display,d,gc,arcs,narcs)
```

```
XFillRectangle(display,d,gc,x,y,width,height)
XFillRectangles(display,d,gc,rectangles,nrectangles)
```

The last 2 routines fill rectangles that are 1 pixel shorter and narrower than the `XDrawRectangle` routines.

```
XFillPolygon(display,d,gc,points,npoints,shape,mode)
    int shape /*Complex,Convex,NonConvex ;
               helps select best fill algorithm */
    int mode; /* CoordModeOrigin or CoordModePrevious */
XFillArc(display,d,gc,x,y,width,height,angle1,angle2)
XFillArcs(display,d,gc,arcs,narcs)

XMoveArea(display, w, srcX, srcY, dstX, dstY, width, height)
    int srcX, srcY;    /* position of top LH corner of area    */
    int dstX, dstY;    /* where top LH corner must end up      */
    int width, length; /* size of area to be moved             */

XCopyArea(display,src,dest,gc,src_x, src_y,width, height, dest_x,dest_y)
    Drawable src_x, src_y;    /* position of top LH corner of area    */
    int src_x,src_y,dest_x, dest_y; /* where top LH corner must end up */
    unsigned int width, height; /* size of area being copied           */
             /* The depth of the source and destination must be the same. */

XCopyPlane(display,src,dest,gc,src_x,src_y,width,height,dest_x,dest_y,plane)
             /* src and dest are drawables with the same root */
    unsigned int width,height;
    unsigned long plane;

XClearArea(display,w,x,y,width,height,exposures)
    int width, height;/* if width=0 then width is set to window_width-x
                         and similarly for y */
    Bool exposures;   /* if True then exposure events generated */

XClearWindow(display,w) /* clears the window */
```

4.1 Drawing Text in a Window

The simplest way to use a font is to call `XLoadQueryfont` which loads the font and returns a pointer to a `XFontStruct` structure which it fills. The `fid` field is what you use when setting the current font.

```
typedef struct {
    XExtData    *ext_data;      /* hook for extension to hang data */
    Font        fid;            /* Font id for this font */
    unsigned    direction;      /* direction the font is painted */
    unsigned    min_char_or_byte2;/* first character */
    unsigned    max_char_or_byte2;/* last character */
    unsigned    min_byte1;      /* first row that exists */
    unsigned    max_byte1;      /* last row that exists */
    Bool        all_chars_exist;/* flag if all characters exist*/
    unsigned    default_char;   /* char to print for undefined character */
    int         n_properties;   /* how many properties there are */
    XFontProp   *properties;    /* pointer to additional properties*/
    XCharStruct min_bounds;     /* minimum bounds over all existing char*/
    XCharStruct max_bounds;     /* minimum bounds over all existing char*/
    XCharStruct *per_char;      /* first_char to last_char information */
    int         ascent;         /* log. extent baseline for spacing */
    int         descent;        /* log. descent baseline for spacing */
} XFontStruct;

XFontStruct *XLoadQueryFont(display,name)  /* prepare a font for action       */
    char *name;          /* name of the font */

XFreeFont(display,font_struct)    /* tells the server that this font is no */
    XFontStruct *font_struct;     /* longer needed, and frees resources.   */
```

Figure 2: Font dimensions

4.2 Finding the Width of Characters

This can be done for 8 and 16 bit characters.

```
int XTextWidth(font_struct,str, count)
    char *str;           /* string */
    XFontStruct *font_struct;  /* font-id of font to use */

int XTextWidth16(font_struct,str, count)
    char *str;           /* string */
    XFontStruct *font_struct;  /* font-id of font to use */

XTextExtents(font_struct,string,nchars,direction,ascent,descent,overall)
    XFontStruct *font_struct;
    char *str;           /* string            */
    int *direction, *ascent, *descent; /*returned dimensions */
    XCharStruct     *overall; /* overall retured dimensions */
```

```
XTextExtents16(font_struct,string,nchars,direction,ascent,descent,overall)
```

4.3 Printing Characters on the Screen

For strings of a single font use

```
XDrawString(display,d,gc,x, y,string,length)
XDrawImageString(display,d,gc,x, y,string,length) /*draws background too*/
XDrawImageString16(display,d,gc,x, y,string,length) /*draws background too*/
```

Otherwise the text to be output needs to be held in the following structure:-

```
typedef struct {
    char *chars;    /* pointer to string */
    int nchars;     /* number of characters */
    int delta;      /* delta between strings */
    Font font;      /* font to print it in, None don't change */
} XTextItem;

XDrawText(display,d,gc,x, y,items,nitems)
    XTextItem *items;
XDrawText16(display,d,gc,x, y,items,nitems)
    XTextItem *items;
```

4.4 Creating Cursors

X provides a automatic facility whereby a cursor may be registered for each window and whenever the mouse is in that window it will take on the correct shape. If a special cursor has not been registered for a window the mouse will use the parent-window's cursor.

There are three cursor creating routines. The 1st is much the easiest:- (check in `<X11/cursorfont.h>` for shape names, or use the `xcursors` program if available)

```
Cursor XCreateFontCursor(display,shape);
```

The utility `bitmap` is useful for designing bitmaps if you want a cursor which is not in the standard library. Such cursors are defined in terms of two bitmaps and two pixel values. The cursor bitmap determines the shape of the cursor using the display function and the foreground and background pixel values, while the mask bitmap determines the area which the cursor is permitted to modify. Every cursor has a hot-spot which determines the coordinates of the point on the screen where the X server considers it to be. The relative coordinates of this point are also given at this stage.

```
Cursor XCreateGlyphCursor(display,sfont,mfont,schar,mchar,scolor,bcolor)
   Font sfont;              /* font glyph for cursor*/
   Font mfont;              /* mask font*/
   unsigned int schar;      /* shape of cursor*/
   unsigned int mchar;      /* mask character */
   XColor *scolor;           /* RGB foreground */
   XColor *bcolor;           /* RGB background */

Cursor XCreatePixmapCursor(display,source,mask,scolor,bcolor,x,y)
   Pixmap source;
   Pixmap mask;
   XColor *scolor;
   XColor *bcolor;
   unsigned int x,y;

XFreeCursor(display,cursor) /* REMOVE A CURSOR PREVIOUSLY GENERATED */
    Cursor cursor;

XDefineCursor(display,w, cursor)
    Cursor cursor; /*use this cursor when pointer is in window */

XUndefineCursor(display,w) /*mouse will use parent's cursor  */
```

4.5 Examples

Hit any key to quit from this program.

```
/* compile using cc -I/usr/include/X11R5 -L/usr/lib/X11R5 -o demo demo.c -lX11 */
#include <stdio.h>
#include <X11/Xlib.h>
 /* This program draws a red line and some text in a chosen font.
  *
  */
     Display *display;
     Window  window;
     XSetWindowAttributes attributes;
     XGCValues gr_values;
     XFontStruct *fontinfo;
     GC gr_context;
     Visual *visual;
     int depth;
     int screen;
     XEvent event;
     XColor    color, dummy;

main (argc, argv)
char   *argv[];
int     argc;
{
     display = XOpenDisplay(NULL);
     screen = DefaultScreen(display);
     visual = DefaultVisual(display,screen);
     depth  = DefaultDepth(display,screen);
     attributes.background_pixel = XWhitePixel(display,screen);

     window = XCreateWindow( display,XRootWindow(display,screen),
                            200, 200, 350, 200, 5, depth,  InputOutput,
                            visual ,CWBackPixel, &attributes);
     XSelectInput(display,window,ExposureMask | KeyPressMask) ;
     fontinfo = XLoadQueryFont(display,"6x10");

     XAllocNamedColor(display, DefaultColormap(display, screen),"red",
                      &color,&dummy);

     gr_values.font = fontinfo->fid;
     gr_values.foreground = color.pixel;
     gr_context=XCreateGC(display,window,GCFont+GCForeground, &gr_values);
     XFlush(display);
     XMapWindow(display,window);
     XFlush(display);

     while(1){
        XNextEvent(display,&event);

        switch(event.type){
        case Expose:
             XDrawLine(display,window,gr_context,0,0, 100, 100);
             XDrawString(display,window,gr_context,100,100,"hello",5);
             break;
        case KeyPress:
             XCloseDisplay(display);
             exit(0);

        }
     }
}
```

Here is a program that uses more of the aforementioned routines.

```
#include  <X11/cursorfont.h>
#include <stdio.h>
#include <X11/Xlib.h>

 main (argc, argv)
 char   *argv[];
 int     argc;
 {
      Display *display;
      Window  win1;
      XEvent event;
      XSetWindowAttributes attributes;
      Cursor cursor_shape;
      XFontStruct *fontinfo;
      GC gr_context1, gr_context2;
      XGCValues gr_values;
      int     screen;
      int     i;

      display = XOpenDisplay(NULL);
      screen  = XDefaultScreen(display);

      attributes.background_pixel      = XWhitePixel(display,screen);
      attributes.border_pixel          = XBlackPixel(display,screen);

      win1= XCreateWindow( display,XRootWindow(display,screen),0,200,
                           XDisplayWidth(display,screen)-400,
                           XDisplayHeight(display,screen)-400,5, 6,
                           InputOutput, XDefaultVisual(display,screen),
                           CWBackPixel| CWBorderPixel, &attributes);

      XSelectInput(display,win1,ExposureMask | KeyPressMask) ;

      gr_values.function =   GXcopy;
      gr_values.plane_mask = AllPlanes;
      gr_values.foreground = BlackPixel(display,screen);
      gr_values.background = WhitePixel(display,screen);
      gr_context1=XCreateGC(display,win1,
                  GCFunction | GCPlaneMask | GCForeground | GCBackground,
                 &gr_values);

      gr_values.function =   GXxor;
      gr_values.foreground = WhitePixel(display,screen);
      gr_values.background = BlackPixel(display,screen);
      gr_context2=XCreateGC(display,win1,
                 GCFunction | GCPlaneMask | GCForeground | GCBackground,
                 &gr_values);

      fontinfo = XLoadQueryFont(display,"6x10");

      cursor_shape=XCreateFontCursor(display,XC_heart);
      XDefineCursor(display,win1,cursor_shape);

      XSetFont(display,gr_context1,fontinfo->fid);
      XSetFont(display,gr_context2,fontinfo->fid);

      XMapWindow(display,win1);

      while(1){
        XNextEvent(display,&event);

        switch(event.type){
        case Expose:
             XClearWindow(display,win1);
             XDrawString(display,win1,gr_context1,50,50,"Hello",5);
             XDrawImageString(display,win1,gr_context2,20,20,"Hello",5);

             XFillRectangle(display,win1,gr_context1,150,150,111,111);
             XFillRectangle(display,win1,gr_context2,200,180,111,111);
             break;
        case KeyPress:
             XCloseDisplay(display);
             exit(0);
        }
      }
}
```

5 Events: Input from a Window

If a key on the server keyboard is pressed or part of a window is uncovered the server can inform the client which is effected by sending events. Events are usually delivered to windows and each window has to tell the server which events it is interested in. It does this using the `XSelectInput` call. If an event hasn't been selected then it will either be discarded or be passed on to another window, depending on how the window's `do_not_propagate_mask` has been set up.

It is possible for a window to hijack all events from the keyboard or the mouse. This is not to be encouraged and has been relegated to a later chapter.

These events are queued by the window system and subroutines are listed below which show how to ask if events are waiting for you and remove them from the queue. In order to find out what an event means it is necessary to be able to read the event structure. An `XEvent` structure always has `type` as the first entry. This uniquely identifies what kind of event it is. The second entry is always a pointer to the display the event was read from. The third entry is always a window of one type or another, (except for keymap events, which have no window.) The pointer to the generic event must be cast before use to access any other information in the structure.

`<X11/Xlib.h>` contains all the event structures with enough explanation for all but `FocusIn`, `FocusOut`, `EnterNotify` and `LeaveNotify` events. For precise details of these, you should consult the manual. `XEvent` is the union of all Event Structures, one of which is:-

```
typedef struct {
        int type;               /* of event */
        Display *display;       /* Display the event was read from */
        Window window;          /* window it is reported relative to */
        Window root;            /* root window that the event occured on */
        Window subwindow;       /* child window */
        unsigned long time;     /* milliseconds */
        int x, y;               /* pointer coordinates in event window */
        int x_root, y_root;     /* coordinates relative to root */
        unsigned int state;     /* key or button mask */
        unsigned int keycode;   /* detail */
        Bool same_screen;       /* same screen flag */
} XKeyEvent;

typedef XKeyEvent XKeyPressedEvent;
typedef XKeyEvent XKeyReleasedEvent;
```

Another useful event deals with button presses.

```
typedef struct {
        int type;               /* of event */
        unsigned long serial;   /* # of last request processed by server */
        Bool send_event;        /* true if this came from a SendEvent request */
        Display *display;       /* Display the event was read from */
        Window window;          /* "event" window it is reported relative to */
        Window root;            /* root window that the event occured on */
        Window subwindow;       /* child window */
        Time time;              /* milliseconds */
        int x, y;               /* pointer x, y coordinates in event window */
        int x_root, y_root;     /* coordinates relative to root */
        unsigned int state;     /* key or button mask */
        unsigned int button;    /* detail */
        Bool same_screen;       /* same screen flag */
} XButtonEvent;
typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleasedEvent;
```

5.1 Event types

There now follows a complete list of events

Mask	**Event Type**	**Union field**	**Structure**
NoEventMask			
KeyPressMask	KeyPress	xkey	XKeyPressedEvent
KeyReleaseMask	KeyRelease	xkey	XKeyReleasedEvent
ButtonPressMask	ButtonPress	xbutton	XButtonPressedEvent
ButtonReleaseMask	ButtonRelease	xbutton	XButtonReleasedEvent
EnterWindowMask	EnterNotify	xcrossing	XEnterWindowEvent
LeaveWindowMask	LeaveNotify	xcrossing	XLeaveWindowEvent
PointerMotionMask	MotionNotify	xmotion	XPointerMovedEvent
PointerMotionHintMask	MotionNotify	xmotion	XPointerMovedEvent
Button1MotionMask	MotionNotify	xmotion	XPointerMovedEvent
Button2MotionMask	MotionNotify	xmotion	XPointerMovedEvent
Button3MotionMask	MotionNotify	xmotion	XPointerMovedEvent
Button4MotionMask	MotionNotify	xmotion	XPointerMovedEvent
Button5MotionMask	MotionNotify	xmotion	XPointerMovedEvent
ButtonMotionMask	MotionNotify	xmotion	XPointerMovedEvent
KeymapStateMask	KeymapNotify	xkeymap	XKeymapEvent
ExposureMask	Expose	xexpose	XExposeEvent
VisibilityChangeMask	VisibilityNotify	xvisibility	XVisibilityEvent
StructureNotifyMask	CirculateNotify	xcirculate	XCirculateEvent
	ConfigureNotify	xconfigure	XConfigureEvent
	DestroyNotify	xdestroywindow	XDestroyWindowEvent
	GravityNotify	xgravity	XGravityEvent
	MapNotify	xmap	XMapEvent
	ReparentNotify	xreparent	XReparentEvent
	UnmapNotify	xunmap	XUnmapEvent
ResizeRedirectMask	ResizeRequest	xresizerequest	XResizeRequestEvent
SubstructureNotifyMask	CirculateNotify	xcirculate	XCirculateEvent
	ConfigureNotify	xconfigure	XConfigureEvent
	CreateNotify	xcreatewindow	XCreateWindowEvent
	DestroyNotify	xdestroywindow	XDestroyWindowEvent
	GravityNotify	xgravity	XGravityEvent
	MapNotify	xmap	XMapEvent
	ReparentNotify	xreparent	XReparentEvent
	UnmapNotify	xunmap	XUnmapEvent
SubstructureRedirectMask	CirculateRequest	xcirculaterequest	XCirculateRequestEvent
	ConfigureRequest	xconfigurerequest	XConfigureRequestEvent
	MapRequest	xmaprequest	XMapRequestEvent
FocusChangeMask	FocusIn	xfocus	XFocusInEvent
	FocusOut	xfocus	XFocusOutEvent
PropertyChangeMask	PropertyNotify	xproperty	XPropertyEvent
ColormapChangeMask	ColormapNotify	xcolormap	XColormapEvent
OwnerGrabButtonMask	***	***	***
(*set in GC*)	GraphicsExpose	xgraphicsexpose	XGraphicsExposeEvent
(*set in GC*)	NoExpose	xnoexpose	XNoExposeEvent
(*always*)	SelectionClear	xselectionclear	XSetSelectionClearEvent
(*always*)	SelectionNotify	xselection	XSelectionEvent
(*always*)	SelectionRequest	xselectionrequest	XSelectionRequestEvent
(*always*)	MappingNotify	xmapping	XMappingEvent
(*always*)	ClientMessage	xclient	XClientMessageEvent

The usual call to get events is the following.

```
XNextEvent(display,rep)      /* flush output then fill in struct with event */
```

```
    XEvent *rep;     /* RETURN next event removed from the queue
                        If queue is empty it waits until an event arrives */
```

Note that the returned event might well need to be cast before its fields are accessed. If, for example, you declare `XEvent event` and find by looking at `event`'s `type` field that it's a `ButtonEvent` then you can use `C`'s `union` concept to correctly access the `x` field by doing `event.xbutton.x`.

```
XSelectInput(display,w, mask)/* declare which events you wish to receive */
    int mask;        /* event mask: each bit permits 1 event type*/

XFlush(display)              /* flush all output buffers */

XSync(display,discard)       /* flush output buffers & wait till all events */
    int discard;     /* have been delivered.  If discard <> 0
                        throw all the events away. */


XPeekEvent(display,rep)      /* flush output then fill in struct with event */
    XEvent *rep;     /* RETURN next event but DONT remove from queue
                        if queue is empty it waits until an event arrives */

int XQLength(display)        /* returns the length of the waiting event
                                queue as an integer */

XPutBackEvent(display,event) /* push an event back unto the top of the queue */
    XEvent *event;   /* the event which you want to push back    */

XWindowEvent(display,w, mask, rep) /* flush bufs then look down the queue for*/
    long mask;       /* event which matches this event mask */
    XEvent *rep;     /* RETURN the first event of this sort found
                        If there's no suitable event, wait */

Bool XCheckWindowEvent(display,w, mask, rep) /* flush bufs then look down the
                        queue for event which fits mask and window */
    long mask;
    XEvent *rep;     /* RETURN the first event of this sort found.
                        If there is no suitable event, 0 is returned */

XMaskEvent(display,mask, rep)/* flush buffers then look down the queue for   */
    int mask;        /* an event which matches this event mask   */
    XEvent *rep;     /* RETURN the first event of this sort found
                        If there's no suitable event, wait */

Bool XCheckMaskEvent(display,mask, rep)/* flush buffers then look down the queue for   */
    int mask;        /* an event which matches this event mask   */
    XEvent *rep;     /* RETURN the first event of this sort found
                        If there's no suitable event, 0 is returned */

int XPending(display) /* flush output & return no of events in input queue */
```

5.2 KeyBoard Encoding

Each key produces a fixed `keycode` in the range [7,255]. A list of `keysyms` is associated with each KeyCode. The keysyms take into account whether Shift keys etc have been pressed. The list of defined KeySyms is in `<X11/keysym.h>`. The mapping from keycodes to keysyms can be changed server-wide. From the command line you can use `xmodmap -pk` to see the current mapping. The mapping that matches the `keysym` to a string can be changed for just one client.

`XLookUpString` is used to interpret a key event, producing both a keysym and an ASCII string

To get the current mapping use

```
KeySym *XGetKeyBoardMapping(display,first_keycode,count,keysyms_per_keycode);
   int first_keycode;
   int count;         /* how many keycodes to check */
   int *keysyms_per_keycode;  /*RETURN*/
```

To change the mapping globally use

```
XChangeKeyBoardMapping(display,first_code,keysyms_per_code,keysyms,num_codes);
   KeySym *keysyms;
```

to change it locally use

```
    /* ALTER KEYBOARD MAPPING FOR A GIVEN KEYCODE                */
    /* WARNING : THIS MAY NOT BE IMPLEMENTED */
XRebindKeySym(display,keysym,list,mod_count,string,num_bytes)
    KeySym keysym;   /* the keycode we wish to rebind  */
    KeySym *list;           /*these are used as modifiers*/
    int mod_count;
    char *string;       /* string we wish to associate with it      */
    int nun_bytes;      /* number of bytes in the string            */
```

To see which keycodes are to be used as modifiers, this structure is needed.

```
typedef struct{
   int lock;
   int shift_a,  shift_b;
   int control_a,control_b;
   int mod1_a,    mod1_b;
   int mod2_a,    mod2_b;
   int mod3_a,    mod3_b;
   int mod4_a,    mod4_b;
   int mod5_a,    mod5_b:
}XModifierKeys;


GetModifierMapping(display,modifier_keys)
   ModifierKeys *modifier_keys;  /*RETURN*/

SetModifierMapping(display,modifier_keys)
   ModifierKeys *modifier_keys;

    /* RETURN A CHAR STRING GIVEN AN ARRAY OF {KeyPressedEvent}s */
int XLookupString(event,buffer,num_bytes,keysym,status); nbytes) */
    XKeyEvent *event; /* array of events to be interpreted*/
    char *buffer;
    int numbytes;      /* number of bytes in the char string      */
    KeySym *keysym;        /*RETURN*/
    XComposeStatus *status;
    /* default mapping is in /usr/lib/keymap.txt but user may    */
    /* specify his own in ~/.Xkeymap - see keycomp(1)            */
```

5.3 Examples

This uses pixmaps, some graphics routines, and events.

```
#include  <X11/cursorfont.h>
#include <stdio.h>
#include <X11/Xlib.h>
      Display *display;
      Window  win1;
      XSetWindowAttributes attributes;
      XFontStruct *fontinfo;
      GC gr_context1;
      XArc arcs[10];
      Pixmap pixmap;
      Visual *visual;
      int screen;
      int depth;
      int i;
 main (argc, argv)
 char   *argv[];
 int     argc;
 {
      XGCValues gr_values;
      XEvent event;

      setbuf (stdout, NULL);
      setbuf (stderr, NULL);
      display = XOpenDisplay(NULL);
      screen = DefaultScreen(display);
      visual = DefaultVisual(display,screen);
      depth  = DefaultDepth(display,screen);
      attributes.background_pixel      = XWhitePixel(display,screen);
      attributes.border_pixel          = XBlackPixel(display,screen);
      attributes.override_redirect     = 0;

      for(i=0;i<10;i++){
          arcs[i].x = 100;arcs[i].y = 50;
           arcs[i].width = 100;arcs[i].height = 50;
      }
      for(i=0;i<5;i++){
           arcs[i].angle1 = 72*64*i;
           arcs[i].angle2 = 35*64;
      }
      for(i=5;i<10;i++){
           arcs[i].angle1 = 72*64*i + 36*64;
           arcs[i].angle2 = 35*64;
      }

      win1= XCreateWindow(display, XRootWindow(display,screen),
                   200,200, 300,200,5, depth, InputOutput, visual,
                   CWBackPixel | CWBorderPixel | CWOverrideRedirect,&attributes);

      XSelectInput(display,win1,ExposureMask | ButtonPressMask | KeyPressMask);
      pixmap = XCreatePixmap(display,win1,200,100,depth);
      fontinfo = XLoadQueryFont(display,"6x10");
      gr_values.font =   fontinfo->fid;
      gr_values.function =   GXcopy;
      gr_values.plane_mask = AllPlanes;
      gr_values.foreground = BlackPixel(display,screen);
```

```
     gr_values.background = WhitePixel(display,screen);
     gr_context1=XCreateGC(display,win1,
             GCFont | GCFunction | GCPlaneMask | GCForeground | GCBackground,
             &gr_values);

     XDefineCursor(display,win1,XCreateFontCursor(display,XC_heart));
     XMapWindow(display,win1);

     do{
       XNextEvent(display,&event);
       if (event.type == Expose){
             draw_ellipse();
             XCopyArea(display,win1,pixmap,gr_context1,50,25,200,100,0,0);
             XSetFunction(display,gr_context1,GXinvert);
             XDrawImageString(display,pixmap,gr_context1,80,45,"pixmap",6);
             XDrawImageString(display,pixmap,gr_context1,90,60,"copy",4);
             XSetFunction(display,gr_context1,GXcopy);
             XDrawString(display,win1,gr_context1,10,20,
                 "Press a key in this window",26);
       }
     }while  (event.type !=KeyPress);

     XCopyArea(display,pixmap,win1,gr_context1,0,0,200,100,100,125);
     XDrawString(display,win1,gr_context1,10,32,
                 "Now press a key to exit",23);
     XFlush(display);

     do{
       XNextEvent(display,&event);
     }while  (event.type !=KeyPress);

     printf("closing display\n");
     XCloseDisplay(display);
 }

 draw_ellipse()
 {
     XSetArcMode(display,gr_context1,ArcPieSlice);
     XFillArcs(display,win1,gr_context1,arcs,5);
     XSetArcMode(display,gr_context1,ArcChord);
     XFillArcs(display,win1,gr_context1,arcs+5,5);
 }
```

This example deals with keyboard input.

```
#include  <X11/cursorfont.h>
#include <stdio.h>
#include <X11/Xlib.h>
#include<X11/keysym.h>
 char workstation[] = {""};
 char str[80];
     Display *display;
     Window  win1;
     XSetWindowAttributes attributes;
     XFontStruct *fontinfo;
     GC gr_context1;
     Visual *visual;
     int screen;
     int depth;
```

```
main (argc, argv)
char   *argv[];
int     argc;
{
     XGCValues gr_values;
     XEvent event;

     setbuf (stdout, NULL);
     setbuf (stderr, NULL);
     display = XOpenDisplay(workstation);
     screen = DefaultScreen(display);
     visual = DefaultVisual(display,screen);
     depth  = DefaultDepth(display,screen);
     attributes.background_pixel      = XWhitePixel(display,screen);
     attributes.border_pixel          = XBlackPixel(display,screen);
     attributes.override_redirect     = 0;

     win1=  XCreateWindow(display, XRootWindow(display,screen),
            200,200, 300,200,5, depth, InputOutput, visual,
            CWBackPixel | CWBorderPixel | CWOverrideRedirect,&attributes);

     XSelectInput(display,win1,ExposureMask | ButtonPressMask | KeyPressMask);

     fontinfo = XLoadQueryFont(display,"6x10");
     gr_values.font =   fontinfo->fid;
     gr_values.function =   GXcopy;
     gr_values.plane_mask = AllPlanes;
     gr_values.foreground = BlackPixel(display,screen);
     gr_values.background = WhitePixel(display,screen);
     gr_context1=XCreateGC(display,win1,
                GCFont | GCFunction | GCPlaneMask | GCForeground | GCBackground,
                &gr_values);

     XDefineCursor(display,win1,XCreateFontCursor(display,XC_heart));
     XMapWindow(display,win1);
     XFlush(display);

     do{
         XNextEvent(display,&event);
         switch(event.type){
         case ButtonPress:
           sprintf(str, "x=%d, y=%d", event.xbutton.x, event.xbutton.y);
           XDrawImageString(display,win1,gr_context1,event.xbutton.x,
               event.xbutton.y, str, strlen(str));
           break;
         case KeyPress:
           deal_with_keys(&event);
           break;
         case Expose:
           XClearWindow(display,win1);
           XDrawImageString(display,win1,gr_context1,0,20,
                             "Press a key or Button. Use Break to exit", 40);
         }
     }while  (1);
}
```

```
deal_with_keys(event)
XKeyEvent *event;
{
int count;
int buffer_size = 80;
char buffer[80];
KeySym keysym;
/* XComposeStatus compose; is not used, though it's in some books */
   count = XLookupString(event,buffer,buffer_size, &keysym);

   if ((keysym >= XK_space) && (keysym <= XK_asciitilde)){
      printf ("Ascii key:- ");
      if (event->state & ShiftMask)
             printf("(Shift) %s\n", buffer);
      else if (event->state & LockMask)
             printf("(Caps Lock) %s\n", buffer);
      else if (event->state & ControlMask)
             printf("(Control) %c\n", 'a'+ buffer[0]-1) ;
      else printf("%s\n", buffer) ;
   }
   else if ((keysym >= XK_Shift_L) && (keysym <= XK_Hyper_R)){
      printf ("modifier key:- ");
      switch (keysym){
      case XK_Shift_L: printf("Left Shift\n"); break;
      case XK_Shift_R: printf("Right Shift\n");break;
      case XK_Control_L: printf("Left Control\n");break;
      case XK_Control_R: printf("Right Control\n"); break;
      case XK_Caps_Lock: printf("Caps Lock\n"); break;
      case XK_Shift_Lock: printf("Shift Lock\n");break;
      case XK_Meta_L: printf("Left Meta\n"); break;
      case XK_Meta_R: printf("Right Meta\n"); break;

      }
    }
   else if ((keysym >= XK_Left) && (keysym <= XK_Down)){
      printf("Arrow Key:-");
      switch(keysym){
      case XK_Left: printf("Left\n");break;
      case XK_Up: printf("Up\n");break;
      case XK_Right: printf("Right\n");break;
      case XK_Down: printf("Down\n");break;
      }
    }
   else if ((keysym >= XK_F1) && (keysym <= XK_F35)){
      printf ("function key %d pressed\n", keysym - XK_F1);

      if (buffer == NULL)
         printf("No matching string\n");
      else
         printf("matches <%s>\n",buffer);
   }

   else if ((keysym == XK_BackSpace) || (keysym == XK_Delete)){
      printf("Delete\n");
   }

   else if ((keysym >= XK_KP_0) && (keysym <= XK_KP_9)){
       printf("Number pad key %d\n", keysym -  XK_KP_0);
   }
   else if (keysym == XK_Break) {
        printf("closing display\n");
        XCloseDisplay(display);
        exit (0);
   }else{
      printf("Not handled\n");
    }
}


/*
  if one wants to find out if another client has changed the key mappings,
  select MappingNotify and do
      XRefreshKeyboardMapping(Event *event);

 */
```

6 Using Colour

Windows in X on a color display always have an associated colormap, a collection of color cells (i.e. triples of R, G and B values).

On some high end displays it may be possible to interpret the contents of the colormap in more than one way. These 'ways' are called `Visuals`. The possible classes of visual are:-

- `Direct Color`:- A pixel value is separated into 3 fields, each used for indexing separate R,G,B arrays.
- `True color`:- Like DirectColor, except that colormap has predefined read-only, server-dependent, values.
- `PseudoColor`:- The pixel value indexes the colormap (an array of triples) to produce RGB values.
- `Static Color`:- read-only PseudoColor.
- `GrayScale`:- A degenerate case of PseudoColor, when within a triple, R=G=B.
- `StaticGray`:- read-only GrayScale.

If you just use

```
Visual XDefaultVisual(display,screen)
```

to find the default `Visual` and use that, you can't go far wrong. Use

```
Colormap XDefaultColorMap(display,screen)
```

to get the default colormap. Colormaps are local to a particular screen and if read-only, may be shared among multiple applications.

Much hardware built today have a single color map, so the primitives are written to encourage sharing of color map entries between applications. If there is not sufficient colormap resources in the display, then some windows may not be displayed in their true colors. If you ask for colors that are named then you're more likely to be able to share an already used colorcell.

Colors can be specified by RGB values or by name. Color information is held in the following structure.

```
typedef struct         /* the colour structure is used to set  */
{                      /* colour cell mappings for the screen  */
    unsigned long pixel;           /* pixel value    */
    unsigned long red,green,blue;  /* intensity (0 to 65535) */
    char flags;                    /* DoRed, DoGreen, DoBlue */
    char pad;
} XColor;
```

6.1 Allocating Color cells

To see how to use Read/Write cells, see a later section. Use `Read Only` cells if at all possible.

```
Status XAllocColor(display,cmap,screen_def)
    XColor *screen_def;        /* supplies RGB values.
           RETURNS pixel value closest to the RGB specified
           and the RGB values actually used */

Status XAllocNamedColor(display,cmap,colorname,screen_def,exact_def);
   char *colorname;    /* supplies name of color                         */
   XColor *screen_def; /* RETURNS the values actually used in the colormap */
   XColor *exact_def;  /* RETURNS the closest color provided by the hardware*/
```

6.2 Read Only Colorcells: An easy Example

```
#include <stdio.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#define NAME "red"
 main (argc, argv)
 char   *argv[];
 int     argc;
 {
     XColor   color_cell, truecolor;
     XSetWindowAttributes attributes;
     Visual *visual;
     int depth;
     int which_color,which_cell,which_plane;
     Display *display;
     Window  win;
     GC gc;
     Colormap cmap;
     XEvent event;
     int screen;
     setbuf (stdout, NULL);
     setbuf (stderr, NULL);

     display = XOpenDisplay(NULL);
     screen = DefaultScreen(display);
     visual = DefaultVisual(display,screen);
     depth  = DefaultDepth(display,screen);
     cmap= XDefaultColormap(display,screen);

     attributes.background_pixel      = XWhitePixel(display,screen);
     attributes.border_pixel          = XBlackPixel(display,screen);
     attributes.override_redirect     = 1;
     win = XCreateWindow(display,XRootWindow(display,screen),0,0,507,426,5,
           depth, InputOutput,visual,
           CWBackPixel | CWBorderPixel | CWOverrideRedirect,&attributes);

     XSelectInput(display,win,ButtonPressMask+ KeyPressMask) ;


     gc=XCreateGC(display,win,NULL,NULL);

     XMapWindow(display,win);

       /* first by RGB values */
       color_cell.flags= DoRed | DoGreen | DoBlue;
       color_cell.red = 10000;
       color_cell.green = 40000;
       color_cell.blue = 60000;
       if (XAllocColor(display,cmap,&color_cell)==0)
           fprintf("XAllocColor failed\n");
       XSetForeground(display,gc,color_cell.pixel);

       XDrawImageString(display,win,gc,250,50,"RGB color",9);
       XFlush(display);

       /* now by name */

       if (XAllocNamedColor(display,cmap,
                          NAME, &color_cell, &truecolor) == 0) {
            fprintf(stderr, "Color '%s' unknown\n", NAME);
       }
       if (truecolor.red != color_cell.red ||
               truecolor.green != color_cell.green ||
               truecolor.blue != color_cell.blue) {
            fprintf(stderr, "Warning: %s color may be wrong\n", NAME);
       }
       XSetForeground(display,gc,color_cell.pixel);
       XDrawImageString(display,win,gc,250,80, "named color",11);
       XFlush(display);

       printf("now press button: \n");
       do{
         XNextEvent(display,&event);
       }while(event.type != ButtonPress);

       printf("closing display\n");
       XCloseDisplay(display);
       exit(0);
 }
```

The following routine consults the database `/usr/lib/X11/rgb.txt` which contains RGB info on the named colors

```
Status XLookupColor(display,cmap,spec,screen_def,exact_def);
   char *spec;         /* case ignored */
   XColor *screen_def; /* RETURNS the values actually used in the colormap */
   XColor *exact_def;  /* RETURNS the closest color provided by the hardware*/
```

3 useful little routines are

```
Status XParseColor(display,colormap,spec, screen_def)
    char *spec;        /* name of colour or spec string        */
         /* a spec string must begin with a hash sign and then */
         /* contain 3,6,9 or 12 hex digits with no spaces      */
         /* these are taken in groups of 1,2,3 or 4 respectively*/
         /* to give the desired RGB value for the colour       */
    Color *screen_def;    /* RETURN nearest RGB supported by hardware */

XQueryColor(display,cmap,def);
   XColor *def; /*supplies pixel value, RETURNS RGB */
XQueryColors(display,cmap,defs, ncolors);
   XColor defs[ncolors]; /*supplies pixel values, RETURNS RGB */
```

X11R5 introduced a more sophisticated color system called `Xcms` which is beyond the scope of this handout.

7 Programming Tips

7.1 Catching Exposure Events

If a window is obscured and then subsequently uncovered it is the clients responsibility to do any repainting which may be needed. X merely sends the correct event type and then leaves it to the application to decide what to do about it. The usual solution is to have a subroutine for redrawing the window which is called whenever the expose events are delivered.

No drawing happens until the first expose event arrives. Ensure that all windows receiving events appear on screen after events have been selected but before events happen. Otherwise the first expose event might be missed.

7.2 Avoiding multiple Exposure

You don't want to redraw the window more than is necessary. If a window receives 2 expose events and nothing happens in between, the window needn't be drawn twice. Strategies are

- Ignore all Expose events whose count field isn't zero.
- Since all expose events generated by a single user action are guarenteed to be contiguous in the queue, you can discard all but the first.

If you want to remove all expose events relating to a certain window, use

```
XEvent event;
...
case Expose:
  /* deal with event */
  while XCheckTypedWindowEvent(display,window,Expose,&event)
        ;
```

7.3 Coping with Exposures

You will have to repair the damage made by occluding windows. Some options are

Display lists:- keep a note of all graphics operations and replay them when necessary.

Backing Store:- If your machine supports it (use `xdpyinfo` from the command line to find out) you could set this facility, but it will tie memory up.

Exposure Events:- the event returns region affected, so you could just redraw this area. Eg, a chessboard could just have damaged squares redrawn.

Clipping using Regions:- redraw only the damaged regions by clipping.

```
XEvent event;
Region region;
XRectangle rectangle;
switch(event.type){
     case Expose:
         region = XCreateRegion();

         /*Make the union of the rectangles into a  Region */
         do{
           rectangle.x = (short) event.xexpose.x
           rectangle.y = (short) event.xexpose.y
           rectangle.width = (unsigned short) event.xexpose.width
           rectangle.height = (unsigned short) event.xexpose.height
           XUnionRectWithRegion(&rectangle,region,region);

         } while(XCheckTypedEvent(display,Expose, &event);

         XSetRegion(display,gc,region);
         redraw(window);
         XDestroyRegion(region);
         break;
```

7.4 Catching Motion Events

These can come in thick and fast. If you find that you can't process them fast enough, try using `PointerMotionHintMask`. This modifies the effect of other 'Pointer Motion' events that you have selected. If the pointer starts moving then only one event will be generated until a button or key changes state or `XQueryPointer` is called.

7.5 Getting arguments

To get a value from `.Xdefaults` use

```
char* XGetDefault(display,program_name, option)
char *program_name;
char *option;
```

To parse a geometry string, use

```
int XGetGeometry(geometry,x,y,width,height);
   int *x,*y;   /*RETURN */
   unsigned int *width, *height;  /*RETURN*/
```

The bits in the returned integer show how many values have been returned. The bits are `XValue`,`YValue`, `WidthValue`, `HeightValue`, `XNegative` and `YNegative`.

If you are likely to have many possible command line arguments, use `XrmInitialise()` to initialise a resource manager, then

```
XrmDatabase XrmGetFileDatabase(resource_file)
char *resource_file;
```

This will create a database from a file than has a format like `.Xdefaults`. To merge in any arguments on the command line, use

```
void XrmParseCommand(db,table, table_length, prog_name, argc, argv);
XrmDatabase db;
XrmOptionDescList table; /* a table of possible options */
int table_length;
char *prog_name;
int *argc /* supplies the number of arguments, returns number unprocessed*/
char **argv;  /* supplies arguments, returns those unprocessed */
```

`table` is a pointer to an array of structures of the following type

```
typedef struct {
   char *option;          /* the command line flag */
   char *resouce;         /* the .X11defaults style string */
   XrmOptionKind argkind; /* XrmoptionSepArg, XrmoptionStickyArg,etc
                             (see /usr/include/X11/Xresource.h  */
   caddr_t value;         /* what to use if argkind is XrmoptionNoArg
                             (use NULL otherwise) */
} XrmOptionDescRec
```

7.6 Colors

There's no way of telling how many colorcells are unallocated.

Your program should work on both monochrome and color screens if possible using something like the following code.

```
Bool iscolor;
Xcolor blue;
iscolor = (XDisplayCells(display, screen)>2);
foreground = (iscolor &&
              XParsecolor(display,cmap, "blue",&blue) &&
              XAllocColor(display,cmap, &blue))
                    ? blue.pixel : BlackPixel (display,screen));
```

7.7 Errors

There is no need for a client program to provide its own error handlers as default handlers exist and these are used automatically if the user does not request otherwise. If you do want your own handlers they can be declared to the X server using these routines. Look out for `Bad Match`, which means your arguments don't match up with each other.

```
char *XDisplayName(string)/* reports an error when the requested display
                             doesn't exist */
char *string /* if NULL, it looks in the enviroment and returns the
                display name that the user was requesting. */


XIOErrorHandler( handler ) /*specify hander for FATAL errors  */
    int handler(Display *);/* user supplied handler routine */
                           /* should not return             */
```

```
XSetErrorHandler( handler )/* specify handler for NONFATAL errors */
    int handler(Display *, XErrorEvent *)
             /* user supplied handler is passed error event */

XSetIOErrorHandler( handler )/* specify handler for NONFATAL errors */
    int handler(Display *)
             /* user supplied handler is passed error event */

typedef struct _XErrorEvent
{
    int serial;        /* serial number of failed request  */
    char error_code;    /* error code of failed request     */
    char request_code;  /* Major op-code of failed request   */
    char minor_code;    /* Minor op-code of failed request */
    XID resourceid;      /* Window of failed request         */
} XErrorEvent;

XGetErrorText(display,code,buffer,length)/* returns a null terminated
                                             string  into buffer*/
    int code;  /* containing a verbal description of the    */
               /* error producing the code specified        */
```

As X is asynchronous errors may not be reported immediately they occur and this sometimes produces confusing effects when one is trying to debug a program. The following routine allows control of the synchronisation

```
int (*XSynchronize(display,onoff))()    /* returns previous state*/
   int onoff;                              /* asynch = 0, else synch */
```

8 Working with other clients and Window Managers

X11 release 4 included routines to implement *The Inter-Client Communications Conventions* which allow clients to cooperate in the areas of selections, cut buffers, window management, session management, and resources. See the online `XR5demo.c` program for details. This chapter covers enough for most needs.

You can suggest to the window manager how it should display and handle your application. The window manager is at liberty to ignore the suggestions. These properties can be set individually or en masse.

8.1 Naming windows and defining icons

```
XStoreName(display,w, name) /* assign a name to a window */
    char *name;          /* name: a null-terminated string      */

Status XFetchName(display,w, name) /* find a window's name */
    char **name;         /* RETURN a pointer to the name array  */

XSetIconName(display,w,icon_name) /* set name used on icon */
    icon_name *char;

Status XGetIconName(display,w,icon_name) /* get name used on icon */
    icon_name **char;   /*RETURN*/
```

To set groups of properties there are 2 structures:-

```
typedef struct{
long flags; /*InputHint,StateHint,IconPixmapHint,IconWindowHint,
            IconPositionHint,IconMaskHint, WindowGroupHint, AllHints */
Bool input;           /* does this application rely on the window manager
                         to get keyboard input ? */
int initial_state; /*NormalState,ZoomState,IconicState,InactiveState,DontCareState*/
Pixmap icon_pixmap;
Window icon_window;   /* window to be used as icon */
int icon_x, icon_y;   /*initial icon position*/
Pixmap icon_mask;
XID window_group;     /* windows can be grouped so that, for example,
                         they could all be de-iconified at once */
} XWMHints;

typedef struct {
  long flags;    /* which fields are defined. Possibilities are
                 USPosition, USSize, PPosition, Psize, PMinSize
                 PMaxSize, PResizeInc, PAspect, PAllHints */
  int x,y;
  int width, height;
  int min_width, max_width;
  int min_height, max_height;
  int width_inc, height_inc;
  struct {
         int x; /* numerator   */
         int y; /* demoninator */
  } min_aspect, max_aspect;
} XSizeHints;
```

Control over the step size of the increment is useful if, say, you want to allow change of a text window by units of a character cell.
These structures are used in these routines:-

```
void XSetNormalHints(display, w, hints)
     Display *display;
     Window  w;
     XSizeHints *hints;

void XGetNormalHints(display, w, hints)
     Display *display;
     Window  w;
     XSizeHints *hints;  /* RETURNED*/

XSetStandardProperties(display,w,window_name,icon_name,
                icon_pixmap,argv,argc,hints)
char *window_name, *icon_name;
Pixmap icon_pixmap;
char **argv;             /* the command line to start the application*/
int  argc;
XSizeHints *hints;

XSetWMHints(display,w,wmhints)
XWMHints *wmhints;

XWMHints *XGetWMHints(display,w,wmhints)
XWMHints *wmhints;
```

If you set maximum size, minimum size and original size to be all the same, and the window manager takes your hints, then your window will be of fixed size.

8.2 Property Lists

Extra information can be associated with windows. This information can be accessed by all clients using the server. Each property has a name and a unique ID called an Atom. Properties have types which are also indicated using Atoms (eg `XA_STRING`, `XA_PIXMAP`). The Atoms for some common properties are predefined in `<X11/Xatom.h>`. To create an Atom with a name, use

```
Atom XInternAtom(display,atom_name,only_if_exists)
   char *atom_name;
   int only_if_exists;/*If False then a new atom will be created if necessary;
                        if True, and there is no corresponding atom then None
                        is returned */
```

The inverse of this is,

```
char *XGetAtomName(display,atom)
```

To obtain the atom type and property format, use

```
int GetWindowProperty(display,w,property,long_offset,long_length,delete,
            req_type,actual_type,actual_format,nitems,bytes_after,prop);
   Atom property;
   long long_offset,long_length; /* the length and offset of data to be
                                    received, in 32-bit quantities     */
   Bool delete;            /*if True then the property is deleted   */
   Atom req_type;          /*AnyPropertyType, etc */
   Atom *actual_type;      /*RETURN*/
   int *actual_format;     /*RETURN data type of returned data*/
   unsigned long *nitems;  /*RETURN*/
   long *bytes_after;      /*RETURN number of bytes left unread */
   unsigned char **prop;   /*RETURN*/
Possible return values are None, Success, BadWindow, BadMatch

Atom *XListProperties(display,w,num_prop)
   int *num_prop;  /*RETURN*/

XChangeProperty(display,w,property,type,format,mode,data,nelements)
   Atom property,type;
   int format;   /*8,16 or 32*/
   int mode;     /* PropModeReplace, PropModePrePend, PropModeAppend*/
   unsigned char *data;
   int nelements;

XRotateWindowProperties(display,w,properties,num_props,npositions)
Atom properties[];

XDeleteProperty(display,w,property)
```

9 Inter client communication

9.1 Selections

In *X* all data transferred between clients must go via the server. *Selections* are pieces of data held by the server on behalf of clients. The following program lets you select an area of a window then paste it into the window of another instance of the same program. If the area chosen is too big to fit into the pre-set pixmap then a string giving the size of the chosen area is displayed instead.

When the left button is used to select an area, the client claims ownership of the property called `COORDS`. The previous owner is sent a `SelectionClear` event by the server, indicating that it has lost ownership. When the middle button is pressed in a client, that client calls `XConvertSelection` saying which selection, the property to place the data in, the window on which to set this property, and the target type the application wants the data in. Initially it asks for the data to be of type `XA_PIXMAP`. These arguments get put into a `XSelectionRequestEvent` by the server and is sent to the current selection owner. If owner can put the data into the requested form, it does so. It constructs a `SelectionNotify` event containing information on the outcome and sends this to the requesting client using `XSendEvent`. The requestor either reads the property or repeats the request, this time asking for the data in the form `XA_STRING`.

Note that this mechanism can be used between any clients using the selection mechanism. For instance, `xterm` passes strings using the `XA_STRING` type.

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>
 /* This program draws a red line and some text in a chosen font.
  * button1 growbox. button2 dump pixmap
  */
#define PIXMAP_W 200
#define PIXMAP_H 200

static char *event_names[] = {
"",
"",
"KeyPress",
"KeyRelease",
"ButtonPress",
"ButtonRelease",
"MotionNotify",
"EnterNotify",
"LeaveNotify",
"FocusIn",
"FocusOut",
"KeymapNotify",
"Expose",
"GraphicsExpose",
"NoExpose",
"VisibilityNotify",
"CreateNotify",
"DestroyNotify",
"UnmapNotify",
"MapNotify",
"MapRequest",
"ReparentNotify",
"ConfigureNotify",
"ConfigureRequest",
"GravityNotify",
"ResizeRequest",
"CirculateNotify",
"CirculateRequest",
"PropertyNotify",
"SelectionClear",
"SelectionRequest",
"SelectionNotify",
"ColormapNotify",
"ClientMessage",
"MappingNotify" };

     Display *display;
     Window  window;
     XSetWindowAttributes attributes;
     XGCValues gr_values;
     XFontStruct *fontinfo;
     GC gr_context;
     Visual *visual;
     Pixmap pix;
     Atom type_asked_for = -1;
     Atom type_offered;
     int depth;
     int screen;
     XEvent event;
     XColor    color, dummy;
     Atom target_property;
     Bool only_if_exists;
     unsigned char* data;
     int  ret_bytes;
     unsigned long ret_remaining, ret_length;
     XEvent ev;
     XSelectionEvent *ev_ptr;
     Atom ret_atom;
     int ret_format;
     long datalength;
     int ret;
     int x_orig, y_orig, x, y, x_final, y_final;
     int x_paste, y_paste;
     char str[1024];

main (argc, argv)
char   *argv[];
int     argc;
{
     display = XOpenDisplay(NULL);
     screen = DefaultScreen(display);
     visual = DefaultVisual(display,screen);
     depth  = DefaultDepth(display,screen);
     attributes.background_pixel = XWhitePixel(display,screen);

     window = XCreateWindow( display,XRootWindow(display,screen),
                            200, 200, 350, 200, 5, depth,  InputOutput,
                            visual ,CWBackPixel, &attributes);
     XSelectInput(display,window,
                  ExposureMask | KeyPressMask | ButtonPressMask |
                  ButtonMotionMask | ButtonReleaseMask ) ;
     fontinfo = XLoadQueryFont(display,"6x10");

     XAllocNamedColor(display, DefaultColormap(display, screen),"red",
                      &color,&dummy);
     setbuf(stdout,NULL);
     setbuf(stderr,NULL);
```

```
gr_values.font = fontinfo->fid;
gr_values.foreground = color.pixel;
gr_context=XCreateGC(display,window,GCFont+GCForeground, &gr_values);
XFlush(display);
pix=XCreatePixmap(display, window, PIXMAP_W, PIXMAP_H, depth);

/* create Atom type to contain graphics info */
target_property = XInternAtom(display, "COORDS", only_if_exists = False);

ev_ptr= (XSelectionEvent*) &ev;

XMapWindow(display,window);

do{
 XNextEvent(display, &event);
 printf("In main loop, got a %s event\n", event_names[event.type]);
 switch(event.type){
 case ButtonPress:
    /* if middle button, paste */   /*REQUESTOR*/
    if (event.xbutton.button == Button2){
      XStoreName(display, window,"Requestor");
      /*create Atom to contain graphics (for sender)*/
      x_paste = event.xbutton.x;
      y_paste = event.xbutton.y;
      target_property = XInternAtom(display, "COORDS",
                        only_if_exists = False);


      if (target_property == None){
        printf("sender hasn't created Atom\n");
        break;
      }

      /* tell the server to send an event to the present owner
       * of the selection, asking the owner to convert the data in
       * the selection to the required type
       */

      type_asked_for = XA_PIXMAP;
      printf("asking for Pixmap\n");
      XConvertSelection(display, XA_PRIMARY, type_asked_for, target_property,
                   window, event.xbutton.time);


    } /* end of button2 */

    /* if left button, cut */              /*OWNER*/
    if (event.xbutton.button == Button1){
       XSetFunction(display, gr_context, GXxor);
       if (XGetSelectionOwner(display, target_property) == window)
          draw_box(x_orig,y_orig,x_final,y_final);
       x_orig=x=event.xbutton.x;
       y_orig=y=event.xbutton.y;

       draw_box(x_orig,y_orig,x,y);
       do{
          XNextEvent(display, &event);
          if(event.type == MotionNotify){
            draw_box(x_orig,y_orig,x,y);
            draw_box(x_orig,y_orig,event.xmotion.x,event.xmotion.y);
            x=event.xmotion.x;
            y=event.xmotion.y;
          }
        }while(event.type != ButtonRelease);
        x_final= x;
        y_final= y;
        XSetFunction(display, gr_context, GXcopy);

        XSetSelectionOwner(display, target_property, window, CurrentTime);
        printf("claiming selection\n");
        if (XGetSelectionOwner(display, target_property) != window){
            printf("failed to own selection\n");
}
        else
            XStoreName(display, window,"Owner");

     }
   break;

  case SelectionRequest: /*OWNER*/
       /* the owner of the selection receives a request to put info
        * into selection.
        */

       if ((mod(x_orig, x_final)>PIXMAP_W)||
              (mod(y_orig, y_final)>PIXMAP_H)){
          type_offered = XA_STRING;
          printf("offering a string\n");
       }
       else{
          type_offered = XA_PIXMAP;
          printf("offering a pixmap\n");
       }
       if (event.xselectionrequest.target == type_offered){
           printf("requestor wants what's on offer\n");
           if (type_offered == XA_STRING){
              sprintf(str,"%d %d %d %d",x_orig,y_orig,x_final,y_final);
              printf("str in %s\n", str);

              ret = XChangeProperty(display,window,target_property, XA_STRING,8,
                       PropModeReplace, str,
                       (int) strlen(str)+1);

              /* check to see if changed ok */
              ret=XGetWindowProperty(display, window,target_property,
                  0L, strlen(str)+1  ,False, AnyPropertyType, &ret_atom,
                  &ret_format, &ret_length, &ret_remaining, &data);
              if (ret != Success)
                  fprintf(stderr, "XGetWindowProperty failed in selectionrequest\n");
              else
                  fprintf(stderr, "data after being set in selectionrequestis %s\n", data);

           }else if (type_offered == XA_PIXMAP){
              ret= XCopyArea(display,window,pix,gr_context,x_orig, y_orig,
                 mod(x_orig,x_final), mod(y_orig, y_final), 0, 0);

              switch(ret){
              case BadDrawable:
                break;
              case BadGC:
                break;
              case BadMatch:
                break;
              }
              ret=XChangeProperty(display,window,target_property, XA_PIXMAP,8,
                       PropModeReplace, pix,
                       (int) sizeof(pix));

              ret=XGetWindowProperty(display, window,target_property,
                   0L, (int) sizeof(pix)  ,False, AnyPropertyType,
                   &ret_atom, &ret_format, &ret_length, &ret_remaining,
                   &data);
              if (ret != Success)
                   fprintf(stderr, "XGetWindowProperty failed in selectionrequest\n");
              else
                   fprintf(stderr, "data after being set in selectionrequestis %s\n", data);
            }
       }else{
              printf("can't provide the type on offer\n");
              type_offered = None;
       }

       /* tell the requestor if the data is ready to read and is in
        * the requested form.
        */

       ev_ptr->display   = event.xselectionrequest.display;
       if (type_offered == None)
          ev_ptr->property =None;
       else
          ev_ptr->property  = event.xselectionrequest.property;
       ev_ptr->selection = event.xselectionrequest.selection;
       ev_ptr->target    = type_offered;
       ev_ptr->type      = SelectionNotify;
       ev_ptr->requestor = event.xselectionrequest.requestor;
       ev_ptr->time      = CurrentTime; /*event->xselectionrequest.time*/
       ev_ptr->send_event = True;

       printf("Sending the event off\n");
       ret=XSendEvent(display, event.xselectionrequest.requestor,
                  False,0L, (XEvent*) ev_ptr);

       if ((ret==BadValue) || (ret==BadWindow))
            printf("SendEvent failed\n");
       break;

  case SelectionClear: /*OWNER*/
       /* something else is the owner now.
        */
       /* if still transferring, wait */
       XSetFunction(display, gr_context, GXxor);
       draw_box(x_orig,y_orig,x_final,y_final);
       XSetFunction(display, gr_context, GXcopy);
       XStoreName(display, window,"Ex owner");
       break;

  case Expose:
         XDrawLine(display,window,gr_context,0,0, 100, 100);
         XDrawString(display,window,gr_context,100,100,"hello",5);
         break;
  case KeyPress:
         XCloseDisplay(display);
         exit(0);

   case SelectionNotify: /*REQUESTOR*/
       /*the owner of the selection has sent the requestor an
        * event
        */
       if (event.xselection.property == target_property )
          printf("property ok in selectionnotify\n");
       else if(event.xselection.property == None){
          printf("owner couldn't provide requested type\n");
          /* so ask for another type */
          if(type_asked_for == XA_PIXMAP){
               printf("couldn't get pixmap so ask for string\n");
               type_asked_for = XA_STRING;
               XConvertSelection(display, XA_PRIMARY, type_asked_for,
                  target_property, window, event.xbutton.time);
               break;
          }
          else{
               printf("can't provide anything\n");
               exit(1);
          }
      }


       if (type_asked_for == XA_STRING)
           datalength = strlen(str)+1;
       else
           datalength = sizeof(pix);

       ret=XGetWindowProperty(display, window,event.xselection.property,
           0L,datalength,False,
       AnyPropertyType, &ret_atom, &ret_format,
           &ret_length, &ret_remaining, &data);

       if (ret != Success)
          printf("XGetWindowProperty failed in selectionnotify\n");

       if (type_asked_for == XA_STRING)
           XDrawString(display, window, gr_context,
                 x_paste, y_paste, data, strlen(data));
       else{
          ret= XCopyArea(display, *data ,window,gr_context,0,0,
                mod(x_orig,x_final), mod(y_orig, y_final),
                x_paste, y_paste);

          switch(ret){
          case BadDrawable:
                break;
              case BadGC:
                break;
              case BadMatch:
                break;
              }

              /* XFree(data); */

         /* the property in xselection should be deleted using
          * XGetWindowproperty with delete set to true so that
          * the owner knows when data has been transferred.
          */
            }
       } /* end of switch */
   }while(1);
}


draw_box(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
XDrawLine(display,window,gr_context,x1,y1,x2,y1);
XDrawLine(display,window,gr_context,x2,y1,x2,y2);
XDrawLine(display,window,gr_context,x2,y2,x1,y2);
XDrawLine(display,window,gr_context,x1,y2,x1,y1);
}

int mod (a,b)
int a,b;
{
  if (a >=b)
     return a-b;
  else
     return b-a;
}



test_ret_of_XChange_Property(ret)
int ret;
{
         switch(ret){
         case BadAtom: printf("Change Prop BadAtom\n"); break;
         case BadMatch: printf("Change Prop BadMatch\n"); break;
         case BadValue: printf("Change Prop BadValue\n"); break;
         case BadAlloc: printf("Change Prop BadAlloc\n"); break;
         case BadWindow: printf("Change Prop BadWindow\n"); break;
         default: printf("Change Prop ok\n");
         }

         /* should look at propertynotify of requesting window to know when
          * transfer successful
          */

}
```

9.2 Buffers

Use selections instead, if you can. Buffers rely on prior agreement as the the interpretation of passed data.

X provides eight buffers in which a client program may store data. They are numbered 0 – 7 and may be accessed explicitly or one after another in a cycle.

```
XStoreBytes(display,bytes, length) /* STORE IN BUFFER ZERO             */
    char *bytes;    /* NOT necessarily ascii or null-terminated */
    int length;             /* number of bytes                  */

char *XFetchBytes(display,nbytes)  /* RECALL BUFFER ZERO               */
    int *nbytes;            /* RETURN                           */

XRotateBuffers(display,n) /* MOVE THEM ROUND BY n STEPS 0 -> n MOD 8    */
    int n;        /* 1 -> (n+1) MOD 8    ETC                    */

XStoreBuffer(display,bytes, nbytes, buffer) /* STORE IN SPECIFIED BUFFER*/
    char *bytes;    /* NOT necessarily ascii or null-terminated */
    int nbytes;             /* number of bytes                  */
    int buffer;             /* buffer to use                    */

char *XFetchBuffer(display,nbytes, buffer) /* RECALL SPECIFIED BUFFER   */
    int *nbytes;            /* RETURN                           */
    int buffer;             /* buffer to use                    */
```

10 Miscellaneous Routines

Not all of the following routines are very useful; they are included mainly for the sake of completeness. The following routines return the biggest (in the case of cursors), else the fastest size of the required resource.

```
XQueryBestSize(display,class,d,width, height,rwidth,rheight)
   int class; /* TileShape, CursorShape, StippleShape*/
   unsigned int width, height;      /* requested size              */
   unsigned int *rwidth, *rheight;   /* RETURN of required legal size */

XQueryBestTile(display,d,width, height,rwidth,rheight)
   unsigned int width, height;      /* requested size              */
   unsigned int *rwidth, *rheight;   /* RETURN of required legal size */

XQueryBestStipple(display,d,width, height,rwidth,rheight)
   unsigned int width, height;      /* requested size              */
   unsigned int *rwidth, *rheight;   /* RETURN of required legal size */
    /* FIND NEAREST POSSIBLE CURSOR SIZE TO REQUESTED SIZE      */
XQueryBestCursor(display,d,width, height, rwidth, rheight)
    unsigned int width, height;      /* requested size              */
    unsigned int *rwidth, *rheight;   /* RETURN possible size        */


int XTranslateCoordinates(display,src_w,dest_w,src_x,src_y,dest_x,dest_y,child)
   int *dest_x, *dest_y; /*RETURN src_x, src_y relative to the dest window*/
   Window *child; /*RETURN a child window of dest_w if the point is within it*/

Status XGetGeometry(display,d,root,x,y,width,height,border_width,depth);
   Drawable *root; /*RETURN the root of d*/
   int *x,*y;   /*RETURN coords if d is a window, else 0 */
   unsigned int *width, *height;  /*RETURN*/
   unsigned int *border_width; /*RETURN border_width if d is a window, else 0*/
   unsigned int *depth; /*RETURN*/

    /* LIST THE PARENT AND CHILDREN OF A WINDOW                 */
Status XQueryTree(display,w, parent, children, nchildren)
   Window *parent;      /* RETURN pointer to parent window id  */
   int *nchildren;      /* RETURN pointer to number of children*/
   Window **children;   /* RETURN pointer to array of child ids*/

XSetScreenSaver(display,timeout,interval,prefer_blanking,allow_exposures)
    int timeout;     /* blank screen after this many seconds*/
    int interval;  /* interval between screen saver invocations*/
    int prefer_blanking;/*DontPreferBlanking,PreferBlanking,DefaultBlanking*/
    int allowExposures;/*DontAllowExposure, AllowExposures,DefaultExposures*/
```

10.1 Pointer control

See also the chapter on events.

```
XChangePointerControl(display,do_accel,do_threshold,accel numerator,
  accel denominator, threshold)/* SET MOUSE ACCELERATION */
    int do_accel;        /*True or False. Do you want to set accel ? */
    int do_threshold;    /*True or False. Do you want to set threshold ? */
    int accel numerator, accel denominator; /* -1 restores default*/
    int threshold;   /* ignore moves up to this number of pixels*/

XGetPointerControl(display,accel numerator,accel denominator, threshold)
    int *accel numerator,   /*RETURN*/
    int *accel denominator; /* RETURN*/
    int *threshold;   /* RETURN*/

    /* FIND MOUSE COORDINATES AND THE STATE OF ITS BUTTONS      */
Status XQueryPointer(display,w,root,child,root x,root y,win x win y,mask)
    Window w;       /* relative to which window              */
    Window *root, *child;      /*RETURN*/
    int *root x, *root y;      /* RETURN*/
    int *win x,*win y;    /* RETURN*/
    unsigned int *mask;    /* RETURN button states                  */

XWarpPointer(display,src_w,dest_w,src_x,src_y,src_width,
       src_height,dest_x,dest_y)
   Window src_w,dest_w;
   int src_x,src_y;
   unsigned int src_width,src_height;
   int dest_x,dest_y;
```

10.2 Keyboard Control

Using these flags, KBKeyClickPercent, KBBellPercent, KBBellPitch, KBBellDuration, KBLed, KBLedMode, KBKey, KBAutoRepeatMode and this structure

```
 typedef struct {
        int key_click_percent;
        int bell_percent;/
        int bell_duration;
        int led;
        int led_mode;         /*LedModeOff LedModeOn*/
        int key;
        int auto_repeat_mode; /*AutoRepeatModeOn AutoRepeatModeOff
                                AutoRepeatModeDefault*/
} XKeyboardControl;
```

one can change keyboard settings with

```
XChangeKeyboardControl(display,value_mask,values);
   XKeyboardControl *values

XAutoRepeatOn(display)          /* control keyboard auto repeat        */
XAutoRepeatOff(display)         /* control keyboard auto repeat        */
XBell(display,percent)
```

10.3 Images

A display may support multiple screens, each having several different visual types supported at different depths. To cope with this, the following routines and structures are used.

```
typedef struct {                 /*      MASKS                */
  Visual *visual;                /*    VisualNoMask           */
  VisualID visualid;             /*    VisualIDMask           */
  int screen;                    /*    VisualScreenMask       */
  int depth;                     /*    VisualDepthMask        */
  int class;                     /*    VisualClassMask        */
  int red_mask;                  /*    VisualRedMaskMask      */
  int green_mask;                /*    VisualGreenMaskMask    */
  int blue_mask;                 /*    VisualBlueMaskMask     */
  int colormap_size;             /*    VisualColormapSizeMask */
  int bits_per_rgb;              /*    VisualBitsPerRGBMask   */
} XVisualInfo;                   /*    VisualAllMask          */

typedef struct _XImage {
    int width, height;          /* size of image */
    int xoffset;                /* number of pixels offset in X direction */
    int format;                 /* Bitmap, XYPixmap, ZPixmap */
    char *data;                 /* pointer to image data */
    int byte_order;             /* data byte order, LSBFirst, MSBFirst */
    int bitmap_unit;            /* quant. of scanline 8, 16, 32 */
    int bitmap_bit_order;       /* LeastSignificant, MostSignificant */
    int bitmap_pad;             /* 8, 16, 32 either XY or ZFormat */
    int depth;                  /* depth of image */
    int bytes_per_line;         /* accelarator to next line */
    int bits_per_pixel;         /* bits per pixel (ZFormat) */
    unsigned long red_mask;     /* bits in Z arrangment */
    unsigned long green_mask;
    unsigned long blue_mask;
    char *obdata;               /* hook for the object routines to hang on */
    struct funcs {              /* image manipulation routines */
        struct _XImage *(*create_image)();
        int (*destroy_image)();
        unsigned long (*get_pixel)();
        int (*put_pixel)();
        struct _XImage *(*sub_image)();
        int (*add_pixel)();
        } f;
} XImage;
```

`Drawables` are stored in the server. An image is stored on the client machine. The `XImage` structure contains all the data corresponding to the drawable. Once you know the format of the data within the image, you can manipulate the data using bit-arithmetic.

There is a standard format for coding bitmap data which is used in some of the routines which follow. The bits are stored in scan-line order starting at the top left hand corner bit which is stored in the least significant position in the first memory word being used. Each line begins on a 16 bit boundary and is padded to a 16 bit boundary at the right hand end. The lines follow each other from top to bottom.

There are two standard formats for pixmaps, XY and Z. In XY format the bitmaps in the pixmap occur one after another with the most significant bit plane first. In Z format the pixels are stored individually and come in the same order as the bits in a bitmap. If the hardware supports 2 to 8 bit planes then each pixel is stored in one byte; if the hardware is 9 to 16 planes the pixels have a 16 bit word each. Z format may not be used with black and white display hardware.

```
   To create an XImage structure use
XImage *XCreateImage(display,visual,depth,format,offset,data,width,height,
        xpad,bytes_per_line)
   int xpad; /*8, 16, 32 */
   char * data;
   int bytes_per_line; /* if 0, then data assumed contiguous */
```

To fill this image with data, use

```
XImage *XGetImage(display,d,x,y,width,height,plane_mask,format);
   XImage *image;
   int src_x,src_y; /* offset from top-left of image */
```

To combine an image in memory with a rectangle of a drawable on your display, use

```
XPutImage(display,d,gc,image,src_x,src_y,dst_x,dst_y,width,height);
   XImage *image;
   int src_x,src_y; /* offset from top-left of image */
```

To copy part of an existing image to a new one and create an XImage, use

```
XImage *XSubImage(ximage,x,y,width,height)
```

You can manipulate the data directly or use

```
long XGetPixel(ximage,x,y)
int XPutPixel(ximage,x,y,pixel)
int XAddPixel(ximage, value) /* This adds (int) value to each pixel*/
```

10.4 Regions

Regions are sets of pixels. They are held by the client program and are often used to create masks etc. These are created by

```
Region XCreateRegion();
```

which creates an empty region. This can be assigned to using

```
Region XPolygonRegion(points,n,fill_rule)
XPoint points[];
```

and destroyed by

```
XDestroyRegion(r)
```

To clear a region, destroy and recreate it. To find the smallest enclosing rectangle use

```
XClipBox(r,rect)
XRectangle rect /*RETURN*/
```

To use a region as a GC's mask, use

```
XSetRegion(display,gc,r)
```

To move a region use

```
XOffsetRegion(r,dx,dy)
```

To change the size of a region use

```
XShrinkRegion(r,dx,dy) /* if dx or dy <0 then the shrink becomes expand*/
```

Other operations are:-

```
XIntersectRegion(sra,srb,dr)
Region dr; /*RETURN*/

XUnionRegion(sra,srb,dr)
Region dr; /*RETURN*/

XUnionRectWithRegion(rect, src_region, dest_region)
Rectangle *rect;

XSubtractRegion(sra,srb,dr)
Region dr; /*RETURN (union - intersection)*/

XXorRegion(sra,srb,dr)
Region dr; /*RETURN*/

int XEmptyRegion(r)   /* is the region null ? */

int XEqualRegion(r1,r2)

int XPointInRegion(r,x,y)

int XRectInRegion(r,x,y,width,height)
```

10.5 Access Control

X is not particularly sophisticated about its access control. Each server program looks in `/etc/X?.hosts` when it starts up to find a list of machines which it will accept connections from. (? is the display number) The following calls permit the user to modify the situation. From the terminal it is also possible to add and remove hosts using `xhost(1)`.

All the functions use the XHostAddress structure. DECnet nodes must terminate in '::' to distinguish them from internet nodes.

```
#include <sys/socket.h>
XAddHost (display,host)              /* add a host      */
    XHostAddress *host;    /* network address */

XAddHosts (display,hosts,num_hosts)              /* add hosts      */
    XHostAddress *hosts;    /* network address */
     int num_hosts;

#include <sys/socket.h>
XRemoveHost (display,host)           /* remove a host   */
    XHostAddress *host;    /* network address */

XRemoveHosts (display,hosts,num_hosts)           /* remove a hosts   */
    XHostAddress *host;    /* network address */
     int num_hosts;

#include <sys/socket.h>
struct in_addr *XListHosts (display,nhosts,state)/* RETURN list of hosts */
    int *nhosts;        /* RETURN number of hosts on list */
    Bool *state;        /* RETURN state of connection set up*/

XSetAccessControl(display,mode)
     int mode;

XEnableAccessControl(display)

XDisableAccessControl(display)
```

10.6 More On Events

Some of the event routines have conditional versions. They take a routine of this form which returns true if the event is wanted, else False.

```
  Bool (*predicate)(display,event,args)
  Display *display;
  XEvent *event;
  char *args;

XIfEvent(display,event,predicate,args) /* returns an event iff right sort*/
   Display *display;
   XEvent *event;
   Bool (*predicate)();
   char *args;

XCheckIfEvent(display,event,predicate,args)
   Display *display;
   XEvent *event;
   Bool (*predicate)();
   char *args;

XPeekIfEvent(display,event,predicate,args)  /* conditional Peek */
   Display *display;
   XEvent *event;
   Bool (*predicate)();
   char *args;
```

10.7 Grabbing

A client can monopolise the keyboard and pointer but if at all possible you should avoid doing this. If you have selected both `ButtonPress` and `ButtonRelease` in a window, then `X` will 'grab' the pointer after the `ButtonPress` event until the button is released.

```
XGrabServer(display)        /* the server will buffer up requests from  */
                            /* all other client programs                */
XUngrabServer(display)      /* allow execution of other client's requests  */

XSetInputFocus(display,focus,revert_to,time)/* keybd events go to window*/
    Window focus;        /* is in if it is a decendent of the focus  */
    int revert_to;       /*RevertToParent,RevertToPointerRoot,RevertToNone
                           if the window becomes non- viewable. */
    Time time;           /* timestamp  or CurrentTime */
    /* window, otherwise they are sent to the focus window itself*/
    /* the RootWindow is the default focus window                */

XGetInputFocus(display,focus,revert_to)
    Window *focus;        /* RETURNS ID of focus window  */
    int *revert_to;       /*RevertToParent,RevertToPointerRoot,RevertToNone*/
    /* GRAB ALL MOUSE EVENTS for those of the clients windows   */
Status XGrabPointer(display,w,owner_events,event_mask,
       pointer_mode,keyboard_mode,confine_to,cursor,time)
    Window w;       /* SelectInput suitably else send here if    */
                    /* normal window has not called SelectInput */
    Bool owner_events; /*If False then all pointer events selected by
                         the mask are reported with respect to grab_window.
                         If True then pointer events that would normally
                         have been reported are reported as usual,
                         otherwise as in False */
    unsigned int event_mask;/* always augmented with ButtonPressMask and
                               ButtonReleaseMask*/
    int pointer_mode,keyboard_mode;  /*If GrabModeAsync then event
                                       processing continues normally.
                                       If GrabModeSync the pointer/keyboard
                                       appears to freeze until XAllowEvents
                                       is called */
    Window confine_to;      /* if pointer isn't initially in this window
                               then it's warped into it */
    Cursor cursor;   /* temporary cursor to use    */
    Time time;       /* either a timestamp or CurrentTime */

XUngrabPointer(display,time)       /* UNDO THE EFFECT OF XGrabPointer */
    Time time;       /* either a timestamp or CurrentTime */

    /* GRAB THE MOUSE WHENEVER A PARTICULAR COMBINATION OF      */
    /* MOUSE BUTTONS AND KEYBOARD KEYS ARE PRESSED              */
Status XGrabButton(display,button,modifiers,w,owner_events,event_mask,
       pointer_mode,keyboard_mode,confine_to,cursor,time)
    unsigned int buttons;
    unsigned int modifiers;
    Window w;   /* window for events which are excluded by      */
       /* SelectInput from the window they would normally go to */
    Bool owner_events;
    unsigned int event_mask;
    int pointer_mode,keyboard_mode;
    Window confine_to;
    Cursor cursor;   /* temporary cursor to use                 */

XUngrabButton(display,button,modifiers,w) /* UNDO XGrabButton    */
    unsigned int buttons;
    unsigned int modifiers;
    Window w;   /* window for events which are excluded by      */
```

10.8 More On Fonts

Here are the individual font preparation routines.

```
Font XLoadFont(display,name)      /* load a font returning the font-id */
    char *name;                   /* null-terminated font name         */

Status XQueryFont(font, info)
    Font font;          /* for the font with this font-id    */
    FontInfo *info;     /* RETURN with FontInfo filled in    */
                        /* except for the font widths array  */

XFreeFontInfo(names,info,actual_count)
XUnloadFont(display,font)

short *XFontWidths(font)
    Font font;          /* fill font widths array for font   */
```

10.9 More On Color

As well as grabbing the keyboard and mouse you can also hog resources. This behaviour is most graphically displayed when clients create `colormaps` of their own. When the pointer goes into their window, their colormap gets swopped in and the windows of other clients suddenly have their colormap taken from under their feet, resulting in false colors. If you do have to make your own colormap and manipulate it, use

```
Colormap XCreateColormap(display,w,visual,alloc)
   Visual *visual; /* GrayScale, Pseudocolor etc */
   int alloc;      /* AllocAll or AllocNone.
                      If visual is StaticGray or TrueColor then pick
                      AllocNone, meaning none of colormap cells are writeable*/

Colormap XCopyColormapAndFree(display,cmap)
                   /* This moves all of the client's existing allocations to
                      a new colormap and frees the old ones. This is useful
                      when the old colormap gets filled up*/

XFreeColormap(display,cmap)

XSetWindowColormap(display,w,cmap)
```

`Read/Write Color Cells` - Using read/write colour cells is more involved than the read-only cells in the earlier chapter. Allocating colorcells and Storing colors in them involve separate commands. N colors and planes can be allocated simultaneously. The idea is that

```
Status XAllocColorCells(display,cmap,contig,plane_masks,nplanes,pixels,ncolors);
   Bool contig;   /* True if planes must be contiguous, else False */
```

Figure 3: Colors and bit planes

```
   unsigned long  plane_masks[nplanes];     /*RETURN*/
   unsigned int   nplanes;         /*supplies number of planes ( >=0 )*/
   unsigned long  pixels[ncolors];          /*RETURN*/
   unsigned int   ncolors;        /*supplies number of colors ( >0 ) */
```

allocates a number of pixel values to an client program. If nplanes is zero then the pixel array returned will contain the pixel values. Otherwise you will need to `OR` each element in the masks array with each element in the pixels array to get all the pixel values. No mask will have any bits in common with other masks or pixel values. If you ask for 3 pixel values and 3 bits in the plane mask, your bit plane mask would be 7 containing the bits 1, 2 and 4 and you would be allocated the pixel values 8, 16 and 24. Thus from the pixel value 8 the values from 8 to 15 can be derived by ORing in one or more of these bits. Similarly 16 gives (16 – 23) and 24 gives (24 – 31). Thus we have in fact been allocated all the pixel values from 8 to 31 inclusive.

By ORing together masks and pixels, $ncolors * 2^n planes$ distinct pixels can be produced, all of which are writeable.

For `GrayScale` or `PseudoColor` each mask will have exactly 1 bit; for `DirectColor` each mask will have 3 bits. The cells you allocate have no defined colors until set with `XLookupColor` then `XStoreColor(s)` or `XStoreNamedColor`.

If no pixels are asked for then as many bit planes as possible are allocated. Such a request can succeed only once. See the example program at the end of this section for further clarification of read/write colour cells.

If you want to vary the number of bits that correspond to each primary color, use

```
Status XAllocColorPlanes(display,cmap,contig,pixels,ncolors,
                         nreds,ngreens,nblues,rmask,gmask,bmask)
```

```
   Bool contig;   /* True if planes must be contiguous, else False */
   unsigned long  pixels[nplanes];     /*RETURN*/
   int   ncolors; /* specifies number of colors needed ( >0 ) */
   int nreds, ngreens, nblues; /* specifies the number of shades you want */
   unsigned long  *rmask, *gmask, *bmask; /*RETURN bit masks for planes*/
```

No mask will have any bits in common with other masks. By ORing together masks and pixels, $ncolors * 2^{(}nreds + ngreens + nblues)$ distinct pixels can be produced, but in the colormap there are only $ncolors * 2^{(}nreds)$ red entries, $ncolors * 2^{(}ngreen)$ green entries and $ncolors * 2^{(}nblue)$ blues entries. To set colors, use

```
XStoreColor(display,cmap,screen_def)
   XColor *screen_def;
XStoreColors(display,cmap,defs,ncolors)
   XColor defs[ncolors];

XStoreNamedColor(display,cmap,color,pixel,flags);
   char *color;
   int flags; /* DoRed, DoGreen,DoBlue */
```

To free color cells or planes use

```
XFreeColors(display,cmap,pixels,npixels,planes);
unsigned long pixels[npixels]; /* pixels you want to free */
unsigned long planes;          /* planes you want to free */
```

This is an example of using read/write colour cells. It is rather long winded in order to illustrate the way in which the pixel values are allocated. It eventually ends up with 24 different colours which it draws in a technicolor stripe across the screen and prints some detail of the pixels is has been given as it goes along.

```
#include  <X11/cursorfont.h>
#include <stdio.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#define TRUE 1
#define FALSE 0
#define PLANELIM  65536
#define NUMPLANES     3            /* number of bit planes asked for */
#define NUMCOLS    3               /* number of colours asked for    */
#define PLNUM     (1 << NUMPLANES)  /* pixel value factor from planes */
#define PIXNUM    NUMCOLS * PLNUM   /* total pixel values obtained    */
 main (argc, argv)
 char   *argv[];
 int     argc;
 {
     XColor   color_cell[PIXNUM];    /* must have room for pixnum */
     XSetWindowAttributes attributes;
     int     Red = 60000;
     int     Green = 40000;
     int     Blue = 20000;
     int     i,j;
     int     mask;
     Visual *visual;
     int depth;
     int which_color,which_cell,which_plane;
     Display *display;
     Visual *v;
```

```
Window  win;
GC gc;
Colormap cmap;
XEvent event;
int     stat;
long     plane_masks[NUMPLANES];
long     pixar[NUMCOLS];
char str0[50],str1[50],str2[50],str3[50], str4[50];
int ret;

setbuf (stdout, NULL);
setbuf (stderr, NULL);

display = XOpenDisplay(NULL);
visual = DefaultVisual(display,0);
depth  = DefaultDepth(display,0);
attributes.background_pixel      = XWhitePixel(display,0);
attributes.border_pixel          = XBlackPixel(display,0);
attributes.override_redirect     = 0;

win = XCreateWindow(display,XRootWindow(display,0),0,0,507,426,5,
   depth,  InputOutput,visual ,CWBackPixel+CWBorderPixel+CWOverrideRedirect,
   &attributes);

XSelectInput(display,win,ExposureMask + ButtonPressMask+ KeyPressMask) ;


gc=XCreateGC(display,win,NULL,NULL);

XMapWindow(display,win);
cmap= XDefaultColormap(display,0);
stat = XAllocColorCells(display,cmap,FALSE,plane_masks, NUMPLANES,
       pixar, NUMCOLS); /* claim some colorcells */

do{
  XNextEvent(display,&event);
  switch(event.type){

  case Expose:
         sprintf(str0,"depth is = %d", depth);
         sprintf(str1,"there are %d colorcells", DisplayCells(display,0));
         switch(XDoesBackingStore(display)){
           case WhenMapped : sprintf(str2,"when mapped");
                           break;
           case NotUseful :  sprintf(str2,"not useful");
                           break;
           case Always :     sprintf(str2,"always");
                           break;
           default :         sprintf (str2,"error in DoesBackingStore");
         }
         XDrawImageString(display,win,gc,20,15,str0,strlen(str0));
         XDrawImageString(display,win,gc,20,30,str1,strlen(str1));
         XDrawImageString(display,win,gc,20,45,str2,strlen(str2));

         sprintf(str2,"planes = %d, %d, %d", plane_masks[0],
                     plane_masks[1],plane_masks[2]);
         sprintf(str3,"pixels = %d, %d, %d.",
                     pixar[0], pixar[1], pixar[2]);
```

```
        XDrawImageString(display,win,gc,20,60,str2,strlen(str2));
        XDrawImageString(display,win,gc,20,75,str3,strlen(str3));
        v=DefaultVisual(display,0);
        switch(v->class){
    case StaticGray:sprintf(str0,"static gray"); break;
    case GrayScale:sprintf(str0," gray scale"); break;
          case StaticColor:sprintf(str0,"static color"); break;
          case PseudoColor:sprintf(str0,"pseudocolor"); break;
          case TrueColor:sprintf(str0,"true color"); break;
          case DirectColor:sprintf(str0,"direct color"); break;
        }
        XDrawImageString(display,win,gc,250,20,str0,strlen(str0));
        XDrawImageString(display,win,gc,400,30,"pix vals used",13);


        for(which_color = 0, which_cell = 0; which_color < NUMCOLS;
            which_color++){
           color_cell[ which_cell++].pixel = pixar[which_color];
           for(which_plane = 0; which_plane < PLNUM; which_plane++){
                   /* permute the bits to generate all the */
        mask = 0;      /* possible pixel values */
              for(i = 1, j = 0; i < PLNUM; i <<= 1, j++)
             if(i & which_plane)
      mask |= plane_masks[j];
        color_cell[ which_cell++].pixel = pixar[which_color] |  mask;
              sprintf(str1," %d",(pixar[which_color]  | mask));
              XDrawImageString(display,win,gc,430,30+12*which_cell,
                 str1,strlen(str1));
           }
        }
        for(i = 0; i < PIXNUM; i++){
           /* generate some arbitrary colours */
           color_cell[i].flags= DoRed | DoGreen | DoBlue;
           color_cell[i].red = Red;
           color_cell[i].green = Green;
           color_cell[i].blue = Blue;
           Red = ((Red + 3000) % 65536);
           Green = ((Green + 3000) % 65536);
           Blue = ((Blue + 3000) % 65536);
        }
        XStoreColors(display,cmap, color_cell,PIXNUM);

        for(i = 0; i < PIXNUM; i++){
           /* draw a multi-coloured stripe    */
           XSetForeground(display,gc, color_cell[i].pixel);
           XFillRectangle(display,win,gc,10 * i, 100, 10, 200);
        }
        XFlush(display);

        printf("now press button: \n");
        break;

case ButtonPress:
        printf("closing display\n");
        XCloseDisplay(display);
        exit(0);
}
```

```
   }while(1);
}
```

`XAllocColorCells` allows you to do quick overlays. By drawing into one plane, you can change the displayed output without affecting the other planes and restore the original quickly.

11 Glossary

atom:- An “atom” is a unique ID corresponding to a string name. Atoms are used to identify properties, types, and selections.

backing store:- When a server maintains the contents of a window, the off-screen saved pixels are known as a “backing store”.

bit gravity:- When a window is resized, the contents of the window are not necessarily discarded. It is possible to request the server (though no guarantees are made) to relocate the previous contents to some region of the window. This attraction of window contents for some location of a window is known as “bit gravity”.

byte order:- For image (pixmap/bitmap) data, byte order is defined by the server, and clients with different native byte ordering must swap bytes as necessary. For all other parts of the protocol, the byte order is defined by the client, and the server swaps bytes as necessary.

buffers:- Memory used to transfer data between clients

class:- Windows can be of class `InputOnly` or `InputOutput`.

client:- An application program connects to the window system server by some interprocess communication (IPC) path, such as a TCP connection or a shared memory buffer. This program is referred to as a “client” of the window system server. More precisely, the client is the IPC path itself. A program with multiple paths open to the server is viewed as multiple clients by the protocol. Resource lifetimes are controlled by connection lifetimes, not by program lifetimes.

color map:- A “color map” consists of a set of cells defining color values. The color map associated with a window is used to display the contents of the window; each pixel value indexes the color map to produce RGB values that drive the guns of a monitor. Depending on hardware limitations, one or more color maps may be installed at one time such that windows associated with those maps display with true colors.

depth:- The “depth” of a window or pixmap is the number of bits per pixel it has. The depth of a graphics context is the depth of the drawables it can be used in conjunction with for graphics output.

direct color:- A class of color map in which a pixel value is decomposed into three separate subfields for indexing. One subfield indexes an array to produce red intensity values; the second subfield indexes a second array to produce blue intensity values; and the third subfield indexes a third array to produce green intensity values. The RGB (red, green, and blue) values in the colormap entry can be changed dynamically.

drawable:- Both windows and pixmaps may be used as sources and destinations in graphics operations. These are collectively known as “drawables”. However, an `InputOnly` window cannot be used as a source or destination in a graphics operation.

extension:- Named “extensions” to the core protocol can be defined to extend the system. Extension to output requests, resources, and event types are all possible.

glyph:- A glyph is a char/pattern in a font.

grab:- If a client grabs a pointer, button + motion events are sent to that client.

graphics context:- Various information for graphics output such as foreground pixel, background pixel, line width, clipping region, etc is stored in a "graphics context". A graphics context can only be used with drawables that have the same root and the same depth as the graphics context.

gravity:- The contents of windows, or subwindows themselves, have a "gravity". This determines how they will be moved when a window is resized. See "bit gravity" and "window gravity".

gray scale:- Gray scale can be viewed as a degenerate case of pseudo color, in which case the red, green, and blue values in any given color map entry are equal, thus producing shades of gray. The gray values can be changed dynamically.

identifier:- Each resource has an "identifier", a unique value associated with it that clients use to name the resource. An identifier can be used over any connection to name the resource.

Image:- Images can be handled via `XImage` structures which hide the diversity of image types from the routines.

InputOnly window:- A window that cannot be used for graphics requests. InputOnly windows are "invisible" and can be used to control such things as cursors, input event generation, and grabbing. InputOnly windows cannot have InputOutput windows as inferiors.

InputOutput window:- The "normal" kind of window that is used for both input and output. It usually has a background. InputOutput windows can have both InputOutput and InputOnly windows as inferiors.

keysym:- An encoding of a symbol on a keycap on a keyboard.

modifier keys:- Shift, Control, Meta, Super, Hyper, ALT, Compose, Apple, CapsLock, ShiftLock, and similar keys are called "modifier" keys.

monochrome:- A special case of static gray, in which there are only two color map entries.

obscures:- Window A "obscures" window B if both are viewable `InputOutput` windows, if A is higher in the global stacking order, and if the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Note the (fine) distinction with "occludes". Also note that window borders are included in the calculation.

occludes:- Window A "occludes" window B if both are mapped, if A is higher in the global stacking order, and if the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Note the (fine) distinction with "obscures". Also note that window borders are included in the calculation. Note that `InputOnly` windows never obscure other windows but can occludes other windows.

pixel value:- A "pixel" is an N-bit value (at a single point), where N is the number of bit planes (that is, the depth) used in a particular window or pixmap. A pixel in a window indexes a color map to derive an actual color to be displayed.

property:- Windows may have associated "properties", consisting of a name, a type, a data format, and some data. The protocol places no interpretation on properties. They are intended as a general-purpose naming mechanism for clients. For example, clients might share information such as resize hints, program names, and icon formats with a window manager via properties.

property list:- The "property list" of a window is the list of properties that have been defined for the window.

pseudo color:- A class of color map in which a pixel value indexes the color map entry to produce independent red, green, and blue values. That is, the color map is viewed as an array of triples (RGB values). The RGB values can be changed dynamically.

Quarks:- The resource manager that deals with Xdefaults has many strings and string comparisons to deal with. By representing each string as an integer then operations are faster. These integers are called `Quarks`.

Region:- These are sets of points (not necessarily rectangles) on which arithmetic can be done.

reply:- Information requested by a client program by means of the X protocol is sent back to the client with a "reply." Both events and replies are multiplexed on the same connection. Most requests do not generate replies. Some requests generate multiple replies.

request:- A command to the server is called a "request". It is a single block of data sent over a connection.

resource:- Windows, pixmaps, cursors, fonts, graphics contexts, and color maps are known as "resources". They all have unique identifiers associated with them for naming purposes. The lifetime of a resource is bounded by the lifetime of the connection over which the resource was created.

RGB values:- Red, green, and blue intensity values are used to define a color. These values are always represented as 16 bit unsigned numbers, with zero the minimum intensity and 65535 the maximum intensity. The X server scales these values to match the display hardware.

root:- The "root" of a pixmap or a graphics context is the same as the root of whatever drawable was used when the pixmap or gcontext was created. The "root" of a window is the root window under which the window was created.

screen:- A server may provide several independent "screens", which typically have physically independent monitors. This would be the expected configuration when there is only a single keyboard and pointer shared among the screens. A Screen structure contains the information about that screen and is linked to the Display structure.

selection:- A "selection" can be thought of as an indirect property with dynamic type. That is, rather than having the property stored in the X server, it is maintained by some client (the "owner"). It is global in nature, 'belonging' to the user (but maintained by clients), rather than being private to a particular window subhierarchy or a particular set of clients. When a client asks for the contents of a selection, it specifies a selection "target type". This target type can be used to control the transmitted representation of the contents. For example, if the selection is "the last thing the user clicked on", and that is currently an image, then the target type might specify whether the contents of the image should be sent in XYFormat or ZFormat. The target type can also be used to control the class of contents transmitted, for example, asking for the "looks" (fonts, line spacing, indentation, and so forth) of a paragraph selection, rather than the text of the paragraph. The target type can also be used for other purposes. The semantics are not constrained by the protocol.

server:- The "server", also referred to as the "X server", provides the basic windowing mechanism. It handles IPC connections from clients, demultiplexes graphics requests onto the screens, and multiplexes input back to the appropriate clients.

static color:- Static color can be viewed as a degenerate case of pseudo color, in which the RGB values are predefined and read-only.

static gray:- Static gray can be viewed as a degenerate case of gray scale, in which the gray values are predefined and read-only. The values are typically (near-)linear increasing ramps.

stipple:- A "stipple pattern" is a bitmap that is used to tile a region to serve as an additional clip mask for a fill operation with the foreground color.

tile:- A pixmap can be replicated in two dimensions to "tile" a region. The pixmap itself is also known as a "tile".

timestamp:- A time value, expressed in milliseconds, typically since the last server reset. Timestamp values wrap around (after about 49.7 days). The server, given its current time is represented by timestamp T, always interprets timestamps from clients by treating half of the timestamp space as being earlier in time than T, and half of the timestamp space as being later in time than T. One timestamp value, represented by the constant `CurrentTime` is never generated by the server. This value is reserved for use in requests to represent the current server time.

true color:- True color can be viewed as a degenerate case of direct color, in which the subfields in the pixel value directly encode the corresponding RGB values. That is, the color map has predefined read-only RGB values. The values are typically (near-)linear increasing ramps.

type:- A type is an arbitrary atom used to identify the interpretation of property data. Types are completely uninterpreted by the server. They are solely for the benefit of clients. X predefines type atoms for many frequently used types, and clients also can define new types.

viewable:- A window is "viewable" if it and all of its ancestors are mapped. This does not imply that any portion of the window is actually visible. Graphics requests can be performed on a window when it is not viewable, but output will not be retained unless the server is maintaining backing store.

visible:- A region of a window is "visible" if someone looking at the screen can actually "see" it: the window is viewable and the region is not occluded by any other window.

visual:- A visual type represents a way of interpreting the pixel values of a colormap.

widget:- In the XToolkit a widget is an XWindow + associated semantics. They are created by combining other widgets or from scratch. Menus, Button Boxes, Scroll-barred windows and Text handlers can all be constructed using toolkit routines.

window gravity:- When windows are resized, subwindows may be repositioned automatically relative to some position in the window. This attraction of a subwindow to some part of its parent is known as "window gravity".