Stardent

WINDOW SYSTEM MANUAL

Stardent Computer Inc.

Part Number: 340-0114-01

Change History

340-0022-02 Original
340-0114-01 January, 1990

CONTENTS

The *Window System Manual* contains:

- *Ardent X Server Private Extension*
- *Ardent X Multiple Buffering/Stereo Library Extension*
- The following *man* pages:

PRIVATE EXTENSIONS

CHAPTER ONE

Ardent X Server Private Extension

Mark Patrick

Ardent Computer

1. Introduction

This document describes the C library interface to a set of X Server protocol requests which are private to the Ardent X Server. These requests allow applications to inquire about the default colormaps on the Titan to perform graphics operations in the Titan overlay planes, and to switch between the stereo and mono screens under program control. Note these requests are specific to the Titan X server and clients which depend upon them will not be portable to other X platforms.

Users desiring to use the functions described in this document should include the file <X11/XTitan.h> in addition to the standard X include files. Client programs should be linked using a command line such as:

```
cc -o myprog myprog.o -lXtitan -lX11
```

2. Overlay Graphics on the Titan

This section describes how to use the X window system to perform graphics in the Titan's 4 overlay planes. We cover the following topics: basic structure of an X client, the different overlay modes supported by the Titan, and event handling for an X client. This document assumes you are already familiar with writing application programs at the Xlib level.

Currently, the X window system standard does not define how overlay graphics should be supported. If this is ever standardized the features described in this document may be changed to bring it into line with the standard.

The Titan has 4 overlay planes which are used by the X server for the cursor and which can now be used by an X client for drawing text, lines or any of the standard X primitives. Typically a client would use the overlay planes for performing annotation of an image without damaging the underlying image which has been drawn into a standard X window (either 8 or 24 bits).

2.1. Obtaining the Overlay Visual

The first step in using overlays is to obtain the overlay visual, this information is used in both creating colormaps and windows. It describes to X what kind of colormap or window you want. The following code fragment shows you how to get this information.

```
XVisualInfo    *visual_info, vinfo_template, *overlay_visual;

vinfo_template.visualid = XTitanOverlayVisualId(dpy, 0);

overlay_visual = XGetVisualInfo( dpy, VisualIDMask, &vinfo_template, &nvisuals);

if (nvisuals == 0) {
      printf("no overlay visual");
      exit(1);
}
```

After executing this code overlay_visual points to a data structure which can be used in creating overlay colormaps and windows. The extension function XTitanOverlayVisualId returns the visual ID for the specified display and screen. It is declared as follows:

```
VisualID XTitanOverlayVisualId(display, screen_number)
    Display *display;
    int screen_number;
```

display	Specifies the connection to the X server.
screen_number	the screen whose overlay visual id is required

2.2. Creating an Overlay Colormap

When creating an overlay window an overlay colormap is required. Such a colormap contains 4 entries. Since color 0 is always interpreted as transparent only the color values specified by pixel values 0, 1 and 2 are of interest. You can create a new overlay colormap using the standard X function *XCreateColormap*:

```
overlay_colormap = XCreateColormap(dpy, XDefaultRootWindow(dpy),
                                   overlay_visual, AllocNone);
```

Note that only one overlay colormap can be installed at any one time. Currently there is no way for a window manager to automatically install a colormap for a subwindow. So to install this colormap the client should call *XInstallColormap*:

```
XInstallColormap(dpy, overlay_colormap);
```

When the default overlay colormap is installed the X sever uses entries 0 and 1 to store the foreground and background colors for the currently displayed cursor.

2.3. Creating an Overlay Window

To draw graphics into the overlay planes a client must first create an overlay window. This is done using the standard *XCreateWindow* call or the extension *XBCreateWindow* or *XBCreateStereoWindow* calls. In each case the visual passed into the create window call is the value of *overlay_visual* obtained above.

The parent window in the create window call should be the window id of the window whose image you wish to overlay. Typically the position of the overlay window would be 0,0 with respective to the parent. By making the overlay window's width and height the full size of the screen you won't need to worry about ever resizing the overlay window if the parent changes size. Typically the overlay window's borderwidth will be zero. The overlay windows background pixmap should be set to None (to disable the window background).

```
attrMask = CWBackPixmap | CWColormap;

window_attributes.background_pixmap = None;
window_attributes.colormap = overlay_colormap;

overlay_win = XCreateWindow(dpy, win, 0, 0, 1280, 1024, 0, 8, InputOutput,
                                    overlay_visual->visual, attrMask, &window_attributes);

if (!overlay_win){
    fprintf(stderr, "could not create overlay window0);
    fflush(stderr);
    XCloseDisplay(dpy);
```

```
        exit(1);
    }

    XMapWindow(dpy, overlay_win);
```

Since the overlay's background pixmap has been set to none you cannot use **XClearArea** or **XClearWindow** to clear some portion of the window. The best way to clear a portion of an overlay window to some specified color is to set the desired pixel value as the foreground color in a gc and call **XFillRectangle**. For example:

```
XSetForeground(dpy, overlay_gc, clear_color);
XFillRectangle(dpy, overlay_win, overlay_gc, 0, 0, 1280, 1024);
```

2.4. Drawing when using overlays

Basically drawing is the same as when not using overlays. However there are a few things to bear in mind. First you should create a graphics context for drawing into the overlay window. This is done by using the standard *XCreateGC*:

```
overlay_gc = XCreateGC(dpy, overlay_win, 0, 0);
```

You can of course set any of the gc values you wish using this call, using XChangeGC or any of the gc convenience functions.

Since the overlay window overlays the entire regular window you must set the subwindow mode of the parent window to *IncludeInferiors*:

```
gcvalues.subwindow_mode = IncludeInferiors;
gc = XCreateGC(dpy, win, GCSubwindowMode, &gcvalues);
```

2.5. Handling Events

Since the overlay window overlays the entire regular window you should select for input events on the overlay window rather than the parent window. Since you have positioned the overlay window at 0, 0 in the parent window's coordinate system and since the overlay window has a zero width border all coordinates reported are the same for either window. Finally note that since the overlay window obscures the parent the background of the parent will not be repainted automatically by the X server. You can clear the main window via a call to:

```
XFillRectangle(dpy, win, gc, 0, 0, 1280, 1024);
```

2.6. Setting the Overlay Mode

The Titan supports two different overlay modes. Independent of the overlay mode an area of the screen where the overlay pixel has the value 0 will appear transparent, allowing the user to view what is displayed in the underlying standard X window. If the overlay mode is *XTitanRamdac3OverlayMode* a separate colormap (containing 4 entries) is used for the overlay planes, In this mode a client can use three colors plus transparency.

If the overlay mode is *XTitanBrooktreeOverlayMode* and the underlying window is 8 bit pseudo color the first 16 entries of the currently installed pseudo colormap are used for the overlay window, allowing 15 colors plus transparency. If the underlying window is 24 bit direct color then this mode behaves as for *XTitanRamdac3OverlayMode* providing 3 colors plus transparency. Use *XTitanSetOverlayMode* to set the overlay mode for a specified window.

void XTitanSetOverlayMode(dpy, window, overlay_mode)
    Display *display;
    Window window;
    int overlay_mode;

display	Specifies the connection to the X server.
window	Specifies the window whose overlay mode is being set.
overlay_mode	the desired overlay_mode.

To obtain the current overlay mode of a specified window use:

int XTitanGetOverlayMode(dpy, window)
    Display *display;
    Window window;

display	Specifies the connection to the X server.
window	Specifies the window whose overlay mode is being set.

3. Obtaining the Default Color Map's

Many applications runninng on the titan can share a common colormap. Sharing colormaps allows more application windows to be shown in their correct colors.

Use XTitanDefaultDirectColormap to obtain the resource id of this default direct color colormap. The definition of the function is:

Colormap XTitanDefaultDirectColormap(dpy, screen_number)
    Display *dpy;
    int screen_number;

dpy	Specifies the connection to the X server.
screen_number	Specifies the screen on which the colormap is needed

Use XTitanDefaultPseudoColormap to obtain the resource id of this default pseudo color colormap. The definition of the function is:

Colormap XTitanDefaultPseudoColormap(dpy, screen_number)
    Display *dpy;
    int screen_number;

dpy	Specifies the connection to the X server.
screen_number	Specifies the screen on which the colormap is needed

Use XTitanDefaultOverlayColormap to obtain the resource id of this default overlay color colormap. The definition of the function is:

Colormap XTitanDefaultOverlayColormap(dpy, screen_number)
    Display *dpy;
    int screen_number;

dpy	Specifies the connection to the X server.
screen_number	Specifies the screen on which the colormap is needed

4. Swapping between Screens

When the -stereo option is used with the Titan X server the user can switch between the mono and stereo screens using ALT-F2. The same effect can be achieved programatically using:

XTitanMapScreen(dpy, physical_screen, logical_screen)
    Display *dpy;
    int physical_screen, logical_screen;

dpy	Specifies the connection to the X server.
physical_screen	Specifies the physical screen
logical_screen	Specifies the desired logical screen

The *physical_screen* parameter defines the physical hardware on which the X screen should be displayed and currently this should be 0. The *logical_screen* specifies which X screen to display. If the logical_screen is 0 then the mono screen will be displayed. If the logical_screen is 1 then the stereo screen will be displayed. This function returns a BadMatch error if either the physical_screen or logical screen numbers are invalid.

To determine which screen is currently being displayed use:

int XTitanGetMappedScreen(dpy, physical_screen)
    Display **dpy*;
    int *physical_screen*;

dpy	Specifies the connection to the X server.
physical_screen	Specifies the physical screen

This request returns the currently displayed X screen number, or -1 if the physical_screen was invalid.

MULTIPLE BUFFERING EXTENSIONS

CHAPTER TWO

Ardent X Multiple Buffering/Stereo Library Extension

Mark Patrick

Ardent Computer

1. Introduction

This document defines a multiple buffering/stereo extension for the low-level C language interface to the X Window System protocol. This interface is specific to the Titan X server and will be replaced by a standardized multiple buffering extension currently being defined by the X consortium.

Clients desiring to use the functions described in this document should include the file <X11/XB.h> in addition to the standard X include files. Client programs should be linked using a command line such as:

```
cc -o myprog myprog.o -lXtitan -lXB -lX11
```

Using this extension clients of the Ardent X window system can: inquire what if any multiple buffering/stereo facilities are available; create one or more multiple buffered/stereo windows; select which buffer they wish to draw into; and select which buffer to display from. Although stereo is not supported (by the X window system) in the current release the programming interface has been defined and is described below. Stereo is planned for the next release.

Once a multiply buffered/stereo window has been created the application can proceed to use standard X11 graphics requests (or extended requests, for 3d images) to generate the individual frames of the animation, switching between the frames once they have been generated, thus producing a smoothly animated sequence.

All X11 InputOutput windows have at least one buffer. A normal X window has exactly one buffer (buffer 0), this is used both to display from and to draw into. For normal animation two buffers are used; one from which the display is currently being generated and one into which the next frame is being rendered. For stereo each buffer consists of a left and right half which are synchronized together. This extension places no restriction on how many buffers are available for display and for rendering, on the current Titan hardware the maximum number of buffers is 2.

2. Inquiring Stereo Facilities

To display a stereo image two synchronized images are required one for the left eye and one for the right. This can be implemented either by two separate screens or by subdividing a single screen into halves and viewing the screen through filters.

Use XBGetStereoInfo to determine what (if any) stereo facilities are supported by a server. The definition of the function is:

unsigned char* XBGetStereoInfo(display, n_screens_return)
    Display **display*;
    int * *n_screens_return*;

display	Specifies the connection to the X server.
n_screens_return	Returns the number of screens supporting stereo.

The XBGetStereoInfo function returns a list of screen numbers, listing each screen upon which the server will support stereo. Currently stereo is not supported by the X window system this will be added in the next release.

3. Inquiring Multiple Buffering Facilities

The number of available buffers depends opon available hardware, the screen on which the window is to be displayed, the depth of the window and its visual type.

Currently, on the Titan, there is a single screen, the number of available buffers depends upon the window's visual type and whether the system has a graphics expansion board. This is summarized in the following table.

Visual Type	Expansion Board	
	No	Yes
Pseudo Color	2	2
Direct Color	1	2

The XBufferInfo structure defined in <X/XB.h> is used to determine what multiple buffering facilities are provided by the server:

```
#define  BufferNoMask      0x0
#define  BufferScreenMask  0x1
#define  BufferDepthMask   0x2
#define  BufferClassMask   0x4

typedef struct {
   unsigned char screen;
   unsigned char depth;
   unsigned char class;
   unsigned char n_buffers;
} XBufferInfo;
```

Use XBGetBufferInfo to determine what double buffering facilities are available. The definition of the function is:

```
XBufferInfo *XBGetBufferInfo(display, binfo_mask, binfo_template, nitems_return)
      Display *display;
      unsigned long binfo_mask;
      XBufferInfo *binfo_template;
      int *nitems_return;
```

display	Specifies the connection to the X server.
binfo_mask	Specifies the buffer information request mask.
bbinfo_template	Specifies the buffer attributes that are to be used in matching the buffer information structures.
nitems_return	Returns the number of matching buffer information structures.

The function XBGetBufferInfo returns a list of all facilities which match the clients request. The client can pass a template specifying which screen, depth and/or visual type is of interest. Exactly which, if any, of these values is used to filter the list of buffering options is determined by the value of the binfo_mask parameter. If there are no facilities matching the specified requirements then XBGetBufferInfo returns NULL. Use XFree to free the data returned by this function.

The returned information specifies for each alternative which matches the specified template: the maximum number of buffers available for a specified screen, depth, and visual class.

4. Creating a multiply buffered window

Use XBCreateWindow to create a window which provides multiple buffering. The definition of the function is:

```
Window XBCreateWindow(display, parent, x, y, width, height,
        borderWidth, depth, visual,
        n_buffers, independent, valuemask, attributes)
    Display *display;
    Window parent;
    int x, y;
    unsigned int width, height, borderWidth;
    int depth;
    Visual *visual;
    int n_buffers;
    unsigned char independent;
    unsigned long valuemask;
    XSetWindowAttributes *attributes;
```

display	Specifies the connection to the X server.
parent	Specifies the parent window ID.
x *y*	Specify the x and y coordinates. These coordinates are the top left outside corner of the created window's borders and are relative to the inside of the parent window's borders.
width *height*	Specify the width and height. These are the created window's inside dimensions. These dimensions do not include the created window's borders, which are entirely outside the window. The dimensions must be nonzero. Otherwise, a BadValue error is returned.
border_width	Specifies, in pixels, the width of the created window's border.
depth	A depth of CopyFromParent means the depth is taken from the parent.
visual	Specifies the visual type. A visual of CopyFromParent means that the visual type is taken from the parent.
n_buffers	Specifies the number of buffers.
independent	Specifies if the window is independently double buffered from the parent.
valuemask	Specifies which window attributes are defined in the attribute argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero (0), the rest is ignored, and the attributes are not referenced.
attributes	Attributes of the window to be set at creation time should be set in this structure. The valuemask should have appropriate bits set to indicate which attributes have been set in the structure.

The XBCreateWindow function creates a multiply buffered unmapped subwindow for a specified parent window, returns the ID of the created window, and causes the server to generate a CreateNotify event. The created window is placed on top in the stacking order with respect to siblings.

The 'independent' parameter specifies whether the client wishes to switch buffers on this window independently from the parent. If the independent parameter has the value False then the number of buffers for this window must be the same as for the parent.

The errors that can be generated by XBCreateWindow are: BadAlloc, BadColor, BadCursor, BadMatch, BadPixmap, BadValue, and BadWindow.

Use XBCreateStereoWindow to create a multiply buffered stereo window. The definition of the function is:

```
Window XBCreateStereoWindow(display, parent, x, y, width, height,
        borderWidth, depth, visual,
        n_buffers, independent, valuemask, attributes)
      Display *display;
      Window parent;
      int x, y;
      unsigned int width, height, borderWidth;
      int depth;
      Visual *visual;
      int n_buffers;
      unsigned char independent;
      unsigned long valuemask;
      XSetWindowAttributes *attributes;
```

display	Specifies the connection to the X server.
parent	Specifies the parent window ID.
x *y*	Specify the x and y coordinates. These coordinates are the top left outside corner of the created window's borders and are relative to the inside of the parent window's borders.
width *height*	Specify the width and height. These are the created window's inside dimensions. These dimensions do not include the created window's borders, which are entirely outside the window. The dimensions must be nonzero. Otherwise, a BadValue error is returned.
border_width	Specifies, in pixels, the width of the created window's border.
depth	A depth of CopyFromParent means the depth is taken from the parent.
visual	Specifies the visual type. A visual of CopyFromParent means that the visual type is taken from the parent.
n_buffers	Specifies the number of buffers.
independent	Specifies if the window is independently double buffered from the parent.
valuemask	Specifies which window attributes are defined in the attribute argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero (0), the rest is ignored, and the attributes are not referenced.
attributes	Attributes of the window to be set at creation time should be set in this structure. The valuemask should have appropriate bits set to indicate which attributes have been set in the structure.

The XBCreateStereoWindow function creates a multiply buffered unmapped stereo subwindow for a specified parent window, returns the ID of the created window, and causes the server to generate a CreateNotify event. The created window is placed on top in the stacking order with respect to siblings.

The 'independent' parameter specifies whether the client wishes to switch buffers on this window independently from the parent. If the independent parameter has the value False then the number of buffers for this window must be the same as for the parent.

The errors that can be generated by XBCreateStereoWindow are: BadAlloc, BadColor, BadCursor, BadMatch, BadPixmap, BadValue, and BadWindow.

5. Selecting Display and Draw Buffers

Use XBSetDrawBuffer to specify which buffer to draw into. The definition of the function is:

XBSetDrawBuffer(display, w, buffer_id, eye)
    Display **display*;
    Window *w*;
    int *buffer_id*;
    unsigned char *eye* ;

display	Specifies the connection to the X server.
w	Specifies the window.
buffer_id	Specifies the buffer.
eye	Specifies which half of a stereo pair to draw into.

The XBSetDrawBuffer function selects which buffer output requests are directed to. Initially the draw buffer and the display buffers are both 0. On a stereo screen eye specifes whether we want to draw into the left eye (eye == XBLeftEye) or the right (eye == XBRightEye). If the screen is not stereo the eye argument is not checked (but some value should be supplied).

The errors that can be generated by XBSetDrawBuffer are: BadWindow, BadValue

Use XBSetDisplayBuffer to specify which buffer to display from. The definition of the function is:

XBSetDisplayBuffer(display, w, buffer_id)
    Display **display*;
    Window *w*;
    int *buffer_id*;

display	Specifies the connection to the X server.
w	Specifies the window.
buffer_id	Specifies the buffer.

The XBSetDisplayBuffer function selects which buffer is to be displayed from when the window is visible. Initially the draw buffer and the display buffers are both 0.

The errors that can be generated by XBSetDisplayBuffer are: BadWindow, and BadValue.

Use XBQueryBuffers to determine the query the additional state information associated with a multiply buffered window. The definition of the function is:

XBQueryBuffers(display, w, n_buffers_return, stereo_return, eye_return,
    display_buffer_return, draw_buffer_return)
    Display **display*;
    Window *w*;
    unsigned short **n_buffers_return*;
    unsigned char **stereo_return*;
    unsigned char **eye_return*;
    unsigned short **display_buffer_return*;
    unsigned short **draw_buffer_return*;

display	Specifies the connection to the X server.
w	Specifies the window.
n_buffers_return	Returns the number of available buffers for the specified window.
stereo_return	Returns True if the window is stereo.
eye_return	Returns which part of a stereo buffer is currently being drawn into. If the window is not stereo the value is undefined.
display_buffer_return	Returns the current display buffer for the specified window.
draw_buffer_return	Returns the current draw buffer for the specified window.

The XBQueryBuffers function returns the number of buffers provided by the window, whether the window is stereo and if so which part of the draw buffer is being drawn into, the current draw buffer and the current display buffer.

The error that can be generated by XBQueryBuffers is: BadWindow

CLIENT *MAN* PAGES

CHAPTER THREE

NAME

awm – Window Manager X Client Application

SYNOPSIS

awm [**–f** filename] [**–e** execfile] [**–b**] [**–i**]

DESCRIPTION

The *awm* command is a window manager client application of the window server. It is heavily based on an earlier work by M. Gancarz of Digital Equipment Corporation (see the end of this document for appropriate acknowledgments).

When the command is invoked, it traces a predefined search path to locate any *awm* startup files. If no startup files exist, *awm* initializes its built-in default file.

If startup files exist in any of the following locations, it adds the variables to the default variables. In the case of contention, the variables in the last file found override previous specifications. Files in the *awm* search path are:

$LIBDIR/*awm/system.awmrc*
$HOME/*.awmrc*

To use only the settings defined in a single startup file, include the variables, **resetbindings, resetmenus** and **resetgagdets** at the top of that specific startup file.

OPTIONS

–f filename
Names an alternate file as an *awm* startup file.

–e execfile
Names a file to exec (typically a shell script invoking other clients) after all startup files have been loaded. This is useful for minimizing the number of map/unmaps that occur when titlebars are added.

–b Causes *awm* to ignore the system startup file.

–i Causes *awm* to ignore $HOME/*.awmrc*.

STARTUP FILE VARIABLES

Variables are typically entered first, at the top of the startup file. Because of a merge with the resource manager, very few variables are set here now. The directives **resetbindings, resetmenus** and **resetgadgets** are still allowed, as are gadget declarations of the form:

gadget[*n*]=*expr*

Where *n* is a positive integer indicating the gadget to initialize and *expr* is one of the following:

string or "*string*" [^ *attributes*]
Set the name of the gadget to *string*. The name will be painted in the gadget box with the **gadget.font** resource or an overriding font attribute (see below). *string* may contain embedded non-alphanumeric characters in the form of \# where # is one or more decimal digits (i.e. \54) or *c* where *c* is a character in the standard C string literal set (i.e. n, r, t, f). This is useful if you've specified a gadget font with glyphs in it (such as cursor) and you want to paint a specific glyph from it in a gadget box. Many such glyphs are not represented by ascii characters.

(*string*) [^ *attributes*]
Load a pixmap from the file named by *string* and tile the gadget with it (see also: **path**).

Additional *attributes* may be specified after a '^' (caret) character in the form:

offset | *gravity* | *foreground* | *background* | *font*

Any omitted parameters will be set to default values.

offset is an integer specifying how far to place this gadget from its nearest neighbor (or an edge). Default offset is **gadget.pad**, or 2 if **gadget.pad** is not defined.

gravity is one of **NoGravity**, **LeftGravity**, **RightGravity** or **CenterGravity**. **NoGravity** specifies that the gadget is to be placed opposite of wherever the last gadget was placed. **LeftGravity** specifies that the gadget should stick to the left of the title bar, **RightGravity** to the right and **CenterGravity** to the center.

foreground and *background* specify the colors used to tile the gadget or draw the text.

font is the name of the font you want the gadget's name drawn in. This overrides the **gadget.font** setting for this gadget.

The default values for *attributes* are 0, **NoGravity**, black (**reverse**: white) and white (**reverse**: black), the setting of **gadget.font**.

It is important to note that in the absence of a *gravity* specification (i.e. we've defaulted to **NoGravity**), the window manager will automatically place a gadget on the side opposite of the last gadget placed. If it's the first gadget placed, it will go to the right. Thus in the absence of any *gravity* (or *offset*) specifications, the window manager will place gadgets in a right-left-right fashion until all gadgets have been placed.

For example:

```
gadget[0] = "die"
gadget[1] = (resize.b) ^ 2 | red | orange
gadget[2] = (iconbox.b) ^ | LeftGravity
gadget[3] = "\56" ^ | LeftGravity | green | black | cursor
```

These declarations will create 4 gadget boxes, situated in the following manner:

The first gadget box will be created wide enough to print the word "die" in it (in whatever gadget font has been defined) and will be placed on the right side (since it hasn't chosen a gravity) against the edge (since it hasn't chosen an offset). Background and foreground colors will be black and white (assignment depending on whether **reverse** is set).

The second gadget box will be tiled with the contents of the file "resize.b" (assuming that it's a valid bitmap file) and will go on the left side (since it also has no gravity and the last one went on the right). It will be offset from the edge by 2 pixels since there was an offset for it. Foreground will be red, background will be orange.

The third gadget will be tiled with the contents of "iconbox.b" and will be placed against the second gadget on the left hand side since we specified a gravity. Colors will be black and white (depending on **reverse**).

The fourth gadget will display glyph #56 from the cursor font in green and black (it's gumby of course).

IMPORTANT: Gadgets may be declared in any order, but you are not allowed to leave gaps, i.e..it's perfectly acceptable to declare gadgets in the order 0, 2, 3, 1, but **not** legal to declare gadgets in the order 0, 3, 2, 4 as gadget #1 has been omitted. This restriction may be removed in the future, but for now you'll get a diagnostic and *awm*

will exit.

All other variables controlling window manager behavior are described in the **X DEFAULTS** section of this man page.

BINDING SYNTAX

Mouse buttons may be bound to particular window manager functions with:

"function=[modifier key(s)]:[context]:mouse events:" menu name "

or

"function=[modifier key(s)]:[context]:mouse events:" text action "

Function and mouse events are the only required fields. The menu name is required with the *f.menu* function definition only. Similarly, text action is required only with the *f.action* function definition.

Function

f.action

Invokes a text action. 'text' should be in quotes with a preceding "action" character (one of '^', '!' or '|'). The syntax is identical to menu text actions which are discussed in greater detail under the **Menus** section of this document.

f.action=[*modifier key(s)*]:[*context*]:*mouse events:action" text "*

f.beep emits a beep from the keyboard. Loudness is determined by the volume variable.

f.circledown causes the top window that is obscuring another window to drop to the bottom of the stack of windows.

f.circleup exposes the lowest window that is obscured by other windows.

f.continue releases the window server display action after you stop action with the **f.pause** function.

f.destroy calls XKillClient on the selected window. Use with caution!! Binding it to naked mouse buttons is probably not a good idea!

f.exit exits the window manager. If you've started *awm* from *xinit* (actually sort of useful now that the **–e** flag has been added), this will also exit the window system.

f.focus directs all keyboard input to the selected window. To reset the focus to all windows, invoke *f.focus* from the root window.

f.iconify When implemented from a window, this function converts the window to its respective icon. When implemented from an icon, f.iconify converts the icon to its respective window.

f.lower lowers a window that is obstructing a window below it.

f.menu invokes a menu. Enclose 'menu name' in quotes if it contains blank characters or parentheses.

f.menu=[*modifier key(s)*]:[*context*]:*mouse events:" menu name "*

f.move moves a window or icon to a new location, which becomes the default location.

f.moveopaque moves a window or icon to a new screen location. When using this function, the entire window or icon is moved to the new screen location. The grid effect is not used with this function.

f.neaten neatens the desktop using the RTL neaten package. See the X **DEFAULTS** for the resources necessary to customize this somewhat complex feature. This function only works if *awm* has been compiled with the -DNEATEN flag (which compiles in the neaten package). Invoking this function without this is a noop (though a warning diagnostic is printed to stderr). See the INSTALLATION section of the README document for more details.

f.newiconify allows you to create a window or icon and then position the window or icon in a new default location on the screen.

f.pause temporarily stops all display action. To release the screen and immediately update all windows, use the **f.continue** function.

f.pushdown moves a window down. The distance of the push is determined by the push variables.

f.pushleft moves a window to the left. The distance of the push is determined by the push variables.

f.pushright moves a window to the right. The distance of the push is determined by the push variables.

f.pushup moves a window up. The distance of the push is determined by the push variables.

f.raise raises a window that is being obstructed by a window above it.

f.refresh results in exposure events being sent to the window server clients for all exposed or partially exposed windows. The windows will not refresh correctly if the exposure events are not handled properly.

f.resize resizes an existing window. Note that some clients, notably editors, react unpredictably if you resize the window while the client is running.

f.restart causes the window manager application to restart, retracing the *awm* search path and initializing the variables it finds.

f.[no]decorate adds or removes "decorations" on the selected window. What decorations are added (or deleted) depends on the settings of various booleans and client-specific resources (see: **SPECIAL RESOURCES**).

The booleans **titles, gadgets** and **borderContext.width** currently influence **awm**'s choice of default decorations.

Modifier Keys

It is preferable to use meta as a modifier key for **awm** (or any other window manager, for that matter), but one may also use ctrl, shift, lock, or null (no modifier key). Modifier keys must be entered in lower case, and can be abbreviated as: c, l, m, s for ctrl, lock, meta, and shift, respectively. It's also permissible to refer to the the modifier keys directly as "mod1, mod2, mod3, mod4 or mod5". A mouse button with no modifier key(s) is often referred to as a "naked" mouse button.

You may bind any number of modifier keys to a function, use the bar (|) character to combine them.

Context

The context refers to the screen location of the cursor when a command is initiated. When you include a context entry in a binding, the cursor must be in that context or the function will not be activated. The window manager recognizes the following seven contexts: icon, window, root, title, gadget[*n*] (where *n* is the gadget number), border and (null).

The icon context refers to any icon and may be safely bound without interfering with window events.

The window context refers to application windows and should be used carefully to avoid usurping button events that applications may want for their own purposes.

The root context refers to the root, or background window.

The title context refers to the titlebar area of a window, if one exists.

The gadget context (with mandatory index) specifies a given gadget box. Binding to a gadget that's undefined (not initialized to anything) is an error.

The border context refers to the artificial border area created when the resource **borderContext.width** is defined (see **borderContext.width** under **X DEFAULTS**). Using this context when no border area exists (i.e. **borderContext.width** is not defined) is a noop.

A (null) context is indicated when the context field is left blank, and allows a function to be invoked from any screen location. This is basically equivalent to specifying all the possible contexts.

Combine contexts using the bar (|) character.

Mouse Buttons

Any of the following mouse buttons are accepted (in lower case) and may be abbreviated as l, m, or r, respectively: left, middle, right.

With the specific button, you must identify the action of that button. Mouse actions can be:

down function occurs when the specified button is pressed down.

up function occurs when the specified button is released.

delta indicates that the mouse must be moved the number of pixels specified with the delta variable before the specified function is invoked. The mouse can be moved in any direction to satisfy the delta requirement.

MENU DEFINITION

After binding a set of function keys and a menu name to **f.menu**, you must define the menu to be invoked, using the following syntax:

menu = (*string*) " *menu name* " {
"*item name*" : "*action*"
.
.
.
}

The *string* in parenthesis is an optional argument which names a pixmap file (see also: **path**) to use as the menu title rather than just using the name of the menu. This is generally only useful if you're using pixmaps for the menu panes as well (see below). Though the *menu name* isn't displayed when you specify *string*, you still need to specify one for *awm* to use when looking up the binding to it.

Enter the *menu name* exactly the way it is entered with the **f.menu** function or the window manager will not recognize the link. If the *menu name* contains blank strings, tabs or parentheses, it must be quoted here and in the **f.menu** function entry. If you haven't chosen to display a pixmap title in *string*, the menu name will be displayed at the top of the menu in whatever font has been chosen for **menu.boldFont** (or its default).

You can enter as many menu items as your screen is long. You cannot scroll within menus.

Any menu entry that contains quotes, special characters, parentheses, tabs, or strings of blanks must be enclosed in double quotes. Follow the item name by a colon (:).

A special case is an item surrounded by parenthesis, which designates the *item name* as the name of a pixmap file to tile the menu pane with. Given a pixmap for the menu title as well (see above), it's possible to create menus that are totally pictorial in nature. There are, however, two caveats. Due to the fact that it's easier to do, the pixmaps are used as backgrounds for the menu panes rather than painting them on whenever a given pane in exposed. This has rather ugly consequences if one of the pixmaps (or a line of text if a pane is textual) is larger than the others. Since the server replicates pixmaps over the entire window, it results in a "wallpaper" effect on the smaller pixmaps. The solution is to make all the pixmaps the same size and/or not mix in any text items that will need a wider pane.

The second problem is that the check marks and pull-right indicators are always displayed in fixed positions on the right and left edges of menu panes. If your pixmaps try to use this real-estate, they may be partially covered by a check mark or pull-right pixmap. Design your menus with this in mind.

Menu Action

Window manager functions
: Any function previously described, e.g., **f.move** or **f.iconify**. Using **f.menu** results in a pull-right pane which you can use to "walk" between menus (see below). A "walk" can be done by moving the cursor onto the pull-right arrow displayed at the right edge of the pane, or by clicking another button in the pane while holding the original one down.

Walking menus
: Select the function **f.menu** and separate it from the *menu name* with a colon (:) i.e.

```
menu = "foo" {
Walking Menu:      f.menu: "NextMenu"
}
```

Text actions
: There are two kinds of special "actions" involving arbitrary strings of text. These are:

Shell commands
: Begin with an exclamation point (!) and set to run in background. You cannot include a new line character within a shell command.

Text strings
: Text strings are placed in the window server's cut buffer.

  Strings with a new line character must begin with an up arrow (^), which is stripped during the copy operation.

Strings without a new line must begin with the bar character (|), which is stripped during the copy operation.

Booleans

Any boolean variable previously described, e.g., **reverse** or **autoraise**. The current state of a boolean variable in a menu will be indicated with a check mark (a check mark means the boolean is set to true).

SPECIAL NOTE:

Menus bound to title bars, gadget boxes or borders cause (where logical) the selected menu action to occur automatically on the titled window as opposed to having to select a window for the action. However, actions requiring mouse tracking (i.e. move, resize) will usually **not** work well in this context. While this limitation will be eliminated in the near future, it is suggested that you use this feature to do things that do not require mouse tracking, such as raise, lower, iconify, etc.

Color Defaults

Colors default to the colors of the root window under any of the following conditions:

1) If you run out of color map entries, either before or during an invocation of *awm*.
2) If you specify a foreground or background color that does not exist in the RGB color database ($LIBDIR/*rgb.txt*).
3) If you omit a foreground or background color.
4) If you specify no colors in the resource database.

X DEFAULTS

A number of variables that were previously specified in the *.uwmrc* file have been moved out of the *.awmrc* file and are now retrieved from the resource database. When a value cannot be found, a default (compiled into *awm*) is substituted. The resource database is also now queried to determine whether or not to title a given window. See the end of this section for details.

In the descriptions below, variable names are listed in boldface, their type in parenthesis, and their default value in double quotes.

autoraise (boolean) "off"
Automatically raise a window to the top when it gains the input focus. See also: **raiseDelay**

autoselect (boolean) "off"
Specifies that the pointer be placed over the first item in a menu, rather than the title, when the menu is popped up.

background (string)
The default background color for all other color choices in *awm*. If **reverse** is not set, this defaults to white, otherwise it defaults to black. References to **background** in this document refer to this resource.

border.foreground (boolean) "**foreground**"
Specifies the border color to use for all windows (this color may be drawn solid or stippled, depending on the window focus and the setting of **border.hilite**).

border.hilite (boolean) "on"
Specifies whether or not window border colors are to be changed on focus changes. On focus in, the window border is changed to solid **border.foreground**. On focus out, it is changed to a "gray" stipple.

borderContext.background (string) "**background**"
Background color to use for border context pixmap. Value is meaningless if **borderContext.width** and **borderContext.pixmap** are undefined.

borderContext.boldPixmap (string) "none"
The name of a pixmap file to load and tile the border context area with when the focus is in. If this is defined, and **hilite** is set, focus changes will cause the border context background to alternate between **borderContext.pixmap** and **borderContext.boldPixmap**. If **borderContext.boldPixmap** is defined, but **borderContext.pixmap** is not, a blank pixmap will be used in place of **borderContext.pixmap**.

borderContext.cursor (int) "XC_cross"
Glyph (in decimal) to retrieve from cursor font for use in border context.

borderContext.foreground (string) "**foreground**"
Foreground color to use for border color pixmap. Value is meaningless if **borderContext.width** and **borderContext.pixmap** are undefined.

borderContext.pixmap (string) "**background**
Pixmap to display as border context area background. Value is meaningless is **borderContext.width** is undefined (or set to zero). Used exclusively as the background unless **borderContext.boldPixmap** and **hilite** are defined.

borderContext.width (int) "0"
Number of pixels wide to make the border context. Though functions may be bound to the border context (see: **Context**) without setting this, they will be impossible to invoke due to the fact that there will be nothing to click on. The

border context should not be confused with the actual window border. It is an artificial area around each window that resembles a border.

delta (int) "1"
Number of pixels that must be moved over before a "delta" action is taken (see: **BINDING SYNTAX**).

foreground (string)
The default foreground color for all other color choices in *awm*. If **reverse** is not set, this defaults to black, otherwise it defaults to white. References to **foreground** in this document refer to this resource.

frameFocus (boolean) "off"
[De]highlight when the pointer [leaves] enters the "frame" of the window (the frame includes the client window, title bar and border context areas, if present). Setting this option also causes the focus to follow the pointer so that keyboard input will go the the client regardless of where the pointer is in the "frame".

freeze (boolean) "off"
Lock out all other clients during certain window manager tasks, such as move and resize.

gadget.border (int) "1"
The width of all gadget borders in pixels.

gadget.font (string) "fixed"
Which font to use for (textual) gadget labels.

gadget.pad (int) "3"
The number of pixels to pad a gadget from its neighbor if it has no offset defined.

gadgets (boolean) "off"
Display gadgets in title bars, if any are declared.

grid (boolean) "off"
Display a finely ruled grid when positioning or resizing windows/icons.

hilite (boolean) "off"
Causes the following actions to occur when a window gains the input focus:

1. If **showName** is on:

   1a. If **title.boldFont** is defined, the window name is redrawn in this font.

   1b. If it's not, then the window name is redrawn in reverse video.

2. If **title.boldPixmap** is defined, the background of the title bar is set to it.

3. If **borderContext.boldPixmap** is defined, the background of the border context area is set to it.

On focus out, the window name is redrawn in **title.font** the title background to **title.pixmap** and the border context to **borderContext.pixmap**, respectively.

If **border.hilite** is undefined, this variable will set it automatically.

Note that most icon variables only affect icons owned by *awm*. Except for foreground and background colors, client created icons are left alone.

icon.background (string) "**background**"
Icon (pixmap) background color.

icon.border (string) "**icon.foreground**"
Color to use for icon borders.

icon.borderWidth (int) "2"
Width of icon border in pixels.

icon.font (string) "8x13"
Which font to use for icon text.

icon.foreground (string) "**foreground**"
Icon (pixmap) foreground color.

icon.hPad (int) "2"
Number of pixels to pad icon text horizontally.

icon.vPad (int) "2"
Number of pixels to pad icon text vertically.

icon.text.background (string) "**icon.background**"
Background color to use for icon text.

icon.text.foreground (string) "**icon.foreground**"
Foreground color to use for icon text.

icon.pixmap (string) "grey"
Pixmap to display as icon background. Since this pixmap will be used to tile all icons owned by *awm*, it's probably not a good idea to put application specific pictures in it. More typically, this will be a cross hatch pattern or some similar background weave. See also: **path, icon.foreground, icon.background.**

installColormap (boolean) "false"
Install a given window's colormap when the pointer enters it. When the pointer leaves, the default colormap is installed.

menu.background (string) "**background**"
Menu background color.

menu.boldFont (string) "8x13bold"
Which font to use for (textual) menu panes. Currently, the only pane using this font is the title pane (unless, of course, it's a pixmap).

menu.border (string) "**foreground**"
Menu border color.

menu.borderWidth (int) "2"
Width of menu border in pixels.

menu.delta (int) "20"
Number of pixels to move on a "pull-right" pane before the submenu attached to it is popped up.

menu.font (string) "8x13"
Which font to use in (textual) menu panes.

menu.foreground (string) "**foreground**"
Menu foreground color.

menu.itemBorder (int) "1"
Width of individual (menu) item borders.

menu.pad (int) "2"
Number of pixels to pad menu text/pixmaps vertically.

The following resources pertain only to the RTL Neaten package and are ignored if *awm* has not been compiled with that option (see the INSTALLATION file).

neaten.absMinWidth (int) "64"
Indicates the amount of space in pixels, that is used as the absolute minimum width of a window during the neaten operation.

neaten.absMinHeight (int) "64"
Indicates the amount of space in pixels, that is used as the absolute minimum height of a window during the neaten operation.

neaten.retainSize (boolean) "true"
Forces to windows to be at least their current size. Windows may overlap as a side effect.

neaten.fill (boolean) "true"
Allows windows to grow to their maximum size during the neaten operation. Normally a window will grow only to the maximum of its desired (based on the WM_NORMAL_HINTS property) and current size.

neaten.fixTopOfStack (boolean) "true"
Fixes the size and location of the window at the top of the window hierarchy. If necessary, this window will overlap even other windows which can not be tiled.

neaten.keepOpen (boolean) "true"
Constrains all windows to remain open during the neaten operation. No windows will be iconized. This operation may cause windows to overlap.

neaten.usePriorities (boolean) "true"
Assigns the windows priorities based on their stacking order (windows closer to the top in the stacking order are given higher priorities). Priorities are used when determining size and location of windows on the screen.

neaten.primaryIconPlacement (string) "**Top**"
Selects the side of the screen where icons are first placed. Legal values are: **Top, Left, Bottom, Right** and **Closest** (to its current position).

neaten.secondaryIconPlacement (string) "**Left**"
Determines where along the specified primary side the icon should be placed. Legal values are those for **neaten.primaryIconPlacement** plus **Center**. Not used if **neaten.primaryIconPlacement** is **Closest**

normali (boolean) "on"
Make sure that icons created with **f.newiconify** stay wholly within the root window (on screen), regardless of attempted placement. If off, put icons wherever the cursor is placed.

normalw (boolean) "on"
Make sure that windows mapped with **f.newiconify** are placed on-screen, regardless of cursor position. If off, put windows wherever the cursor is placed.

path (string) "null"
A number of items (titles, menus, etc) now allow you to specify a pixmap file, rather than just a text string to display. Since it would be tedious to type in full pathnames for these files if they all lived in the same places, the directory(s) named by **path** are searched if the pixmap file's pathname does not begin with a slash (/) or tilde (~) and is not found in the current directory.

path is a white-space separated list of one or more directories to search, much like that used by the Unix C-shell. The ~ notation used to designate your (or someone else's) home directory is supported, but wildcards are not.

popup.background (string) "**background**"
Background color to use for pop-up text.

popup.borderWidth (int) "2"
Width of pop-up window border in pixels.

popup.font (string) "9x15"
Which font to use for popup window text.

popup.foreground (string) "
Foreground color to use for pop-up text.

popup.pad (int) "4"
Number of pixels to pad pop-up text horizontally.

pushRelative (boolean) "on"
When a window is pushed, push 1/**push** of the window. If off, move window **push** pixels.

pushDown (boolean) "false"
When adding a title bar or border context to a window, put the border or title bar area at the current x, y position and "push" the window down to make room. For windows with an upper edge at or near the top of the screen, this gives the most asthetically pleasing results. For windows near the bottom, it does not. If set to false, the title bar/border will be added "on top" and the window will not be moved down. Note that the setting of this resource also affects how the window is maniplated during resizes, title removals, etc.

raiseDelay (int) "100"
Amount of time in milliseconds to wait (while window has focus) before raising. If pointer leaves window before time elapses, raise is not performed.

reverse (boolean) "on"
Reverse background/foreground colors for titles, menus, gadget windows, popup windows, etc. In the absence of any color specifications, this results in black-on-white.

rootResizeBox (boolean) "on"
Put the resize (popup) window in the upper left corner of the root window, rather than on the window being resized. This saves a potentially expensive refresh that would occur when the popup was unmapped. If your server supports save-unders, it's generally (but not always) better to turn **saveUnder** on instead.

saveUnder (boolean) "off"
Use save-unders for menus and pop-up windows. If the server does not support save-unders, this action does nothing.

showName (boolean) "on"
Display the window name in a title (assuming that the window is titled in the first place).

title.background (string) "**background**"
Background color to use for title pixmap.

title.boldFont (string) "none"
Which font to use for titlebar labels if focus is and **hilite** is enabled. If this isn't set, and **hilite** is, the title text will be displayed with **title.font** in reverse video.

title.boldPixmap (string) "none"
The name of a pixmap file to load and tile titlebars with when the focus is in. If this is defined, and **hilite** is set, focus changes will cause title backgrounds to alternate between **title.pixmap** and **title.boldPixmap**. If **title.boldPixmap** is defined, but **title.pixmap** is not, a blank pixmap will be used in place of **title.pixmap**.

title.cursor (int) "XC_left_ptr"
Glyph (in decimal) to retrieve from cursor font for use in title bar.

title.font (string) "vtsingle"
Which font to use for titlebar labels. Used exclusively unless **title.boldFont** and **hilite** are set.

title.foreground (string) "**foreground**"
Foreground color to use when drawing background (both normal and bold) pixmaps.

title.pad (int) "2"
Number of pixels to pad title bar text vertically.

title.pixmap (string) "none"
The name of a pixmap file to load and tile titlebars with. This background is use exclusively unless the **title.boldPixmap** is defined and **hilite** is set.

title.text.background (string) "**title.background**"
Background color to use when drawing title bar text.

title.text.foreground (string) "**title.foreground**"
Foreground color to use when drawing title bar.

titles (boolean) "off"
Put title bars on all windows (both existing windows and new ones as they're created. See also: **f.title**

volume (int) "2"
Specifies the bell volume (delta on volume set with *xset*).

wall (boolean) "off"
Restrict window movement to edges of screen (rootwindow). This feature is fairly handy and should probably be bound to a menu so that it can readily be turned on and off.

warpOnDeIconify (boolean) "off"
Warp pointer to upper right corner of window on de-iconify.

warpOnIconify (boolean) "off"
Warp pointer to center of icon on iconify.

warpOnRaise (boolean) "off"
Warp pointer to upper left corner of window on raise.

windowName.offset (int) "0"
Number of pixels from the right or left edge of a titlebar to print the window name (assuming that **showName** is set). If this value is negative, the name will be offset **nameOffset** (plus the name length) pixels from the right edge. If the value is positive, then the name will be offset **nameOffset** pixels from the left edge. If the value is zero, the name will be centered. Since the length of a window name can vary dynamically, this value will be adjusted, when necessary, to ensure that the name is visible in the title bar.

zap (boolean) "off"
Causes ghost lines to follow the window or icon from its previous location to its new location during a move, resize or iconify operation.

SPECIAL RESOURCES

name.**wm_option.autoRaise** (boolean)

name.**wm_option.borderContext** (boolean)

name.**wm_option.gadgets** (boolean)

name.**wm_option.title** (boolean)

These resources determine whether or not a given application really wants a title, gadgets, border context area or to be auto-raised. The application's CLASS and NAME (in the WM_CLASS property) are checked against the string supplied for *name* (for example: Xclock*wm_option.title: off).

Specifying one of these resources overrides any other boolean settings (I.E. **awm.titles** or **awm.gadgets**) and may be used to turn things on and off at the application and/or class level for applications, regardless of **awm**'s settings.

Note: Both class and name resources are checked, and in that order. Thus specific applications may override settings for their class, if desired.

These resources are "special" as they are checked for under the application's name, not *awm*'s; I.E. **xclock.wm_option.autoRaise** is not **awm.xclock.wm_option.autoRaise** as one might think.

EXAMPLES

The following sample startup file shows the default window manager options:

```
# Global variables
#
resetbindings
resetmenus
#
# Mouse button/key maps
#
# FUNCTION   KEYS  CONTEXT  BUTTON   MENU(if any)
# ========   ====  =======  ======   ============
f.menu =     meta  :    :left down   :"WINDOW OPS"
f.menu =     meta  :    :middle down :"EXTENDED WINDOW OPS"
f.move =     meta  :w|i :right down
f.circleup = meta  :root :right down
#
# Menu specifications
#
menu = "WINDOW OPS" {
"(De)Iconify": f.iconify
Move:          f.move
Resize:        f.resize
Lower:         f.lower
Raise:         f.raise
}

menu = "EXTENDED WINDOW OPS" {
Create Window:              !"xterm &"
Iconify at New Position:    f.lowericonify
Focus Keyboard on Window:   f.focus
Freeze All Windows:         f.pause
Unfreeze All Windows:       f.continue
Circulate Windows Up:       f.circleup
Circulate Windows Down:     f.circledown
}
```

RESTRICTIONS

The color specifications have no effect on a monochrome system. There's currently no way to specify a keysym in place of a button (up/down/delta) specification. This restriction will be removed in the near future.

FILES

$LIBDIR/rgb.txt
$LIBDIR/font
/usr/skel/.awmrc
$LIBDIR/awm/system.awmrc
$HOME/.awmrc

SEE ALSO

X(1), X(8C)

AUTHOR

M. Gancarz, DEC Ultrix Engineering Group, Merrimack, New Hampshire, using some algorithms originally by Bob Scheifler, MIT Laboratory for Computer Science

J. Hubbard, U.C. Berkeley, Berkeley, Ca. Ardent Computer, Sunnyvale, Ca. Various modifications and enhancements using code developed by M. Gancarz and Digital Equipment Corp.

NAME

bdfresize – Resize BDF Format Font

SYNOPSIS

bdfresize [-w *factor*] [-h *factor*] [-f *factor*] [*bdf-file*]

DESCRIPTION

Bdfresize is a command to magnify or reduce font which is described with the standard BDF format. If *bdf-file* is not specified, it reads from stdin. *Bdfresize* outputs the result to stdout in BDF format. A few COMMENT lines are inserted to the result font. FONT, ATTRIBUTES, STARTCHAR and ENCODING lines are copied from source font. If a syntax error occurs in source font, *bdfresize* stop its process.

OPTIONS

–w *factor*
Specifies resize factor for the font width.

–h *factor* Specifies resize factor for the font height.

–f *factor* Same as specifying both **-w** and **-h** with same *factor*.

factor is described either of following forms.
<digits>
<digits>/<digits>

SEE ALSO

Character Bitmap Distribution Format 2.1 (Adobe Systems, Inc.)

AUTHOR

kagotani@cs.titech.junet

NAME

bdftosnf - BDF to SNF font compiler for X11

SYNOPSIS

bdftosnf [-p#] [-s#] [-m] [-l] [-M] [-L] [-w] [-W] [-t] [-i] [bdf-file]

DESCRIPTION

bdftosnf reads a Bitmap Distribution Format (BDF) font from the specified file (or from standard input if no file is specified) and writes an X11 server normal font (SNF) to standard output.

OPTIONS

-p# Force the glyph padding to a specific number. The legal values are 1, 2, 4, and 8. **-s#** Force the scanline unit padding to a specific number. The legal values are 1, 2, and 4.

-m Force the bit order to most significant bit first.

-l Force the bit order to least significant bit first.

-M Force the byte order to most significant bit first.

-L Force the byte order to least significant bit first.

-w Print warnings if the character bitmaps have bits set to one outside of their defined widths.

-W Print warnings for characters with an encoding of -1; the default is to silently ignore such characters.

-t Expand glyphs in "terminal-emulator" fonts to fill the bounding box.

-i Don't compute correct ink metrics for "terminal-emulator" fonts.

SEE ALSO

X(1), Xserver(1)
"Bitmap Distribution Format 2.1"

NAME

bitmap, bmtoa, atobm – bitmap editor and converter utilities for X

SYNOPSIS

bitmap [-options ...] *filename WIDTHxHEIGHT*

bmtoa [-chars ...] [*filename*]

atobm [-chars *cc*] [-name *variable*] [-xhot *number*] [-yhot *number] [filename]*

DESCRIPTION

The *bitmap* program is a rudimentary tool for creating or editing rectangular images made up of 1's and 0's. Bitmaps are used in X for defining clipping regions, cursor shapes, icon shapes, and tile and stipple patterns.

The *bmtoa* and *atobm* filters convert *bitmap* files (FILE FORMAT) to and from ASCII strings. They are most commonly used to quickly print out bitmaps and to generate versions for including in text.

USAGE

Bitmap displays grid in which each square represents a single bit in the picture being edited. Squares can be set, cleared, or inverted directly with the buttons on the pointer and a menu of higher level operations such as draw line and fill circle is provided to the side of the grid. Actual size versions of the bitmap as it would appear normally and inverted appear below the menu.

If the bitmap is to be used for defining a cursor, one of the squares in the images may be designated as the *hotspot*. This determines where the cursor is actually pointing. For cursors with sharp tips (such as arrows or fingers), this is usually at the end of the tip; for symmetric cursors (such as crosses or bullseyes), this is usually at the center.

Bitmaps are stored as small C code fragments suitable for including in applications. They provide an array of bits as well as symbolic constants giving the width, height, and hotspot (if specified) that may be used in creating cursors, icons, and tiles.

The *WIDTHxHEIGHT* argument gives the size to use when creating a new bitmap (the default is 16x16). Existing bitmaps are always edited at their current size.

If the *bitmap* window is resized by the window manager, the size of the squares in the grid will shrink or enlarge to fit.

OPTIONS

Bitmap accepts the following options:

–help
This option will cause a brief description of the allowable options and parameters to be printed.

–display *display*
This option specifies the name of the X server to used.

–geometry *geometry*
This option specifies the placement and size of the bitmap window on the screen. See *X* for details.

–nodashed
This option indicates that the grid lines in the work area should not be drawn using dashed lines. Although dashed lines are prettier than solid lines, on some servers they are significantly slower.

-name *variablename*
This option specifies the variable name to be used when writing out the bitmap file. The default is to use the basename of the *filename* command line argument.

-bw *number*
This option specifies the border width in pixels of the main window.

-fn *font*
This option specifies the font to be used in the buttons.

-fg *color*
This option specifies the color to be used for the foreground.

-bg *color*
This option specifies the color to be used for the background.

-hl *color*
This option specifies the color to be used for highlighting.

-bd *color*
This option specifies the color to be used for the window border.

-ms *color*
This option specifies the color to be used for the pointer (mouse).

Bmtoa accepts the following option:

-chars *cc*
This option specifies the pair of characters to use in the string version of the bitmap. The first character is used for 0 bits and the second character is used for 1 bits. The default is to use dashes (-) for 0's and sharp signs (#) for 1's.

Atobm accepts the following options:

-chars *cc*
This option specifies the pair of characters to use when converting string bitmaps into arrays of numbers. The first character represents a 0 bit and the second character represents a 1 bit. The default is to use dashes (-) for 0's and sharp signs (#) for 1's.

-name *variable*
This option specifies the variable name to be used when writing out the bitmap file. The default is to use the basename of the *filename* command line argument or leave it blank if the standard input is read.

-xhot *number*
This option specifies the X coordinate of the hotspot. Only postive values are allowed. By default, no hotspot information is included.

-yhot *number*
This option specifies the Y coordinate of the hotspot. Only postive values are allowed. By default, no hotspot information is included.

CHANGING GRID SQUARES

Grid squares may be set, cleared, or inverted by pointing to them and clicking one of the buttons indicated below. Multiple squares can be changed at once by holding the button down and dragging the cursor across them. Set squares are filled and represent 1's in the bitmap; clear squares are empty and represent 0's.

Button 1
This button (usually leftmost on the pointer) is used to set one or more squares. The corresponding bit or bits in the bitmap are turned on (set to 1) and the square or squares are filled.

Button 2
This button (usually in the middle) is used to invert one or more squares. The corresponding bit or bits in the bitmap are flipped (1's become 0's and 0's become 1's).

Button 3
This button (usually on the right) is used to clear one or more squares. The corresponding bit or bits in the bitmap are turned off (set to 0) and the square or squares are emptied.

MENU COMMANDS

To make defining shapes easier, *bitmap* provides 13 commands for drawing whole sections of the grid at once, 2 commands for manipulating the hotspot, and 2 commands for updating the bitmap file and exiting. A command buttons for each of these operations is located to the right of the grid.

Several of the commands operate on rectangular portions of the grid. These areas are selected after the command button is pressed by moving the cursor to the upper left square of the desired area, pressing a pointer button, dragging the cursor to the lower right hand corner (with the button still pressed) , and then releasing the button. The command may be aborted by pressing any other button while dragging or by releasing outside the grid.

To invoke a command, move the pointer over that command and click any button.

Clear All
This command is used to clear all of the bits in the bitmap as if Button 3 had been dragged through every square in the grid. It cannot be undone.

Set All
This command is used to set all of the bits in the bitmap as if Button 1 had been dragged through every square in the grid. It cannot be undone.

Invert All
This command is used to invert all of the bits in the bitmap as if Button 2 had been dragged through every square in the grid.

Clear Area
This command is used to clear a region of the grid as if Button 3 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be cleared should be selected as outlined above.

Set Area
This command is used to set a region of the grid as if Button 1 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be set should be selected as outlined above.

Invert Area
This command is used to inverted a region of the grid as if Button 2 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be inverted should be selected as outlined above.

Copy Area
This command is used to copy a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be copied should be selected as outlined

above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be copied.

Move Area

This command is used to move a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be moved should be selected as outlined above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be moved. Any squares in the region's old position that aren't also in the new position are cleared.

Overlay Area

This command is used to copy all of the set squares in a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be copied should be selected as outlined above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be overlaid. Only the squares that are set in the region will be touched in the new location.

Line

This command will set the squares in a line between two points. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the two end points of the line.

Circle

This command will set the squares on a circle specified by a center and a point on the curve. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the center of the circle and then over a point on the curve. Small circles may not look very round because of the size of the grid and the limits of having to work with discrete pixels.

Filled Circle

This command will set all of the squares in a circle specified by a center and a point on the curve. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the center of the circle and then over a point on the curve. All squares side and including the circle are set.

Flood Fill

This command will set all clear squares in an enclosed shape. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on any empty square inside the shape to be filled. All empty squares that border horizontally or vertically with the indicated square are set out to the enclosing shape. If the shape is not closed, the entire grid will be filled.

Set Hot Spot

This command designates one square in the grid as the hot spot if this bitmap to be used for defining a cursor. When the command is invoked, the cursor will change indicating that the pointer should be clicked on the square to contain the hot spot.

Clear Hot Spot

This command removes any designated hot spot from the bitmap.

Write Output

This command writes a small fragment of C code representing the bitmap to the filename specified on the command line. If the file already exists, the original file will be renamed to *filename~* before the new file is created. If an error occurs in either the renaming or the writing of the bitmap file, a dialog box will appear asking whether or not *bitmap* should use */tmp/filename* instead.

Quit

This command causes *bitmap* to display a dialog box asking whether or not it should save the bitmap (if it has changed) and then exit. Answering *yes* is the same as invoking *Write Output*; *no* causes *bitmap* to simply exit; and *cancel* will abort the *Quit* command so that more changes may be made.

FILE FORMAT

The *Write Output* command stores bitmaps as simple C program fragments that can be compiled into programs, referred to by X Toolkit pixmap resources, manipulated by other programs (see *xsetroot*), or read in using utility routines in the various programming libraries. The width and height of the bitmap as well as the hotspot, if specified, are written as preprocessor symbols at the start of the file. The bitmap image is then written out as an array of characters:

```
#define name_width 11
#define name_height 5
#define name_x_hot 5
#define name_y_hot 2

static char name_bits[] = {
   0x91, 0x04, 0xca, 0x06, 0x84,
   0x04, 0x8a, 0x04, 0x91, 0x04
};
```

The **name** prefix to the preprocessor symbols and to the bits array is constructed from the *filename* argument given on the command line. Any directories are stripped off the front of the name and any suffix beginning with a period is stripped off the end. Any remaining non-alphabetic characters are replaced with underscores. The *name_x_hot* and *name_y_hot* symbols will only be present if a hotspot has been designated using the *Set Hot Spot* command.

Each character in the the array contains 8 bits from one row of the image (rows are padded out at the end to a multiple of 8 to make this is possible). Rows are written out from left to right and top to bottom. The first character of the array holds the leftmost 8 bits of top line, and the last characters holds the right most 8 bits (including padding) of the bottom line. Within each character, the leftmost bit in the bitmap is the least signficant bit in the character.

This process can be demonstrated visually by splitting a row into words containing 8 bits each, reversing the bits each word (since Arabic numbers have the significant digit on the right and images have the least significant bit on the left), and translating each word from binary to hexidecimal.

In the following example, the array of 1's and 0's on the left represents a bitmap containing 5 rows and 11 columns that spells *X11*. To its right is is the same array split into 8 bit words with each row padded with 0's so that it is a multiple of 8 in length (16):

```
10001001001     10001001 00100000
01010011011     01010011 01100000
00100001001     00100001 00100000
01010001001     01010001 00100000
10001001001     10001001 00100000
```

Reversing the bits in each word of the padded, split version of the bitmap yields the left hand figure below. Interpretting each word as hexidecimal number yields the array of numbers on the right:

```
10010001 00000100     0x91 0x04
11001010 00000110     0xca 0x06
10000100 00000100     0x84 0x04
10001010 00000100     0x8a 0x04
10010001 00000100     0x91 0x04
```

The character array can then be generated by reading each row from left to right, top to bottom:

```
static char name_bits[] = {
  0x91, 0x04, 0xca, 0x06, 0x84,
  0x04, 0x8a, 0x04, 0x91, 0x04
};
```

The *bmtoa* program may be used to convert *bitmap* files into arrays of characters for printing or including in text files. The *atobm* program can be used to convert strings back to *bitmap* format.

USING BITMAPS IN PROGRAMS

The format of *bitmap* files is designed to make bitmaps and cursors easy to use within X programs. The following code could be used to create a cursor from bitmaps defined in *this.cursor* and *this_mask.cursor*:

```
#include "this.cursor"
#include "this_mask.cursor"

XColor foreground, background;
/* fill in foreground and background color structures */
Pixmap source = XCreateBitmapFromData (display, drawable,
    this_bits, this_width, this_height);
Pixmap mask = XCreateBitmapFromData (display, drawable,
    this_mask_bits, this_mask_width, this_mask_height);
Cursor cursor = XCreatePixmapCursor (display, source, mask,
    foreground, background, this_x_hot, this_y_hot);
```

Additional routines are available for reading in *bitmap* files and returning the data in the file, in Bitmap (single-plane Pixmap for use with routines that require stipples), or full depth Pixmaps (often used for window backgrounds and borders). Applications writers should be careful to understand the difference between Bitmaps and Pixmaps so that their programs function correctly on color and monochrome displays.

For backward compatibility, *bitmap* will also accept X10 format *bitmap* files. However, when the file is written out again it will be in X11 format

X DEFAULTS

Bitmap uses the following resources:

Background
The window's background color. Bits which are 0 in the bitmap are displayed in this color. This option is useful only on color displays. The default value is *white.*

BorderColor
The border color. This option is useful only on color displays. The default value is *black.*

BorderWidth
The border width. The default value is 2.

BodyFont
The text font. The default value is *variable.*

Foreground
The foreground color. Bits which are 1 in the bitmap are displayed in this color. This option is useful only on color displays. The default value is *black.*

Highlight
The highlight color. *bitmap* uses this color to show the hot spot and to indicate rectangular areas that will be affected by the *Move Area, Copy Area, Set Area,* and *Invert Area* commands. If a highlight color is not given, then *bitmap* will highlight by inverting. This option is useful only on color displays.

Mouse
The pointer (mouse) cursor's color. This option is useful only on color displays. The default value is *black.*

Geometry
The size and location of the bitmap window.

Dimensions
The *WIDTHxHEIGHT* to use when creating a new bitmap.

SEE ALSO

X(1), *Xlib - C Language X Interface* (particularly the section on *Manipulating Bitmaps*), *XmuReadBitmapDataFromFile*

BUGS

The old command line arguments aren't consistent with other X programs.

If you move the pointer too fast while holding a pointer button down, some squares may be missed. This is caused by limitations in how frequently the X server can sample the pointer location.

There is no way to write to a file other than the one specified on the command line.

There is no way to change the size of the bitmap once the program has started.

There is no *undo* command.

COPYRIGHT

AUTHOR

bitmap by Ron Newman, MIT Project Athena; documentation, *bmtoa,* and *atobm* by Jim Fulton, MIT X Consortium.

NAME

cpicker – colormap editor for X11

SYNTAX

/usr/bin/X11/cpicker [-id *id*] [-root] [-wname *name*] [-display *display*]

OPTIONS

-id *id* This option allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window.

-wname *name*
This option allows the user to specify that the window named *name* is the target window on the command line rather than using the mouse to select the target window.

-root This option specifies that X's root window is the target window.

-display *display*
This option allows you to specify the server to connect to; see *X(1)*.

DESCRIPTION

Cpicker makes temporary changes to the installed colormap, allowing the user to observe the effects. It's useful for trying to pick that perfect color or find the appropriate color combination for an application.

When cpicker first starts, it either uses the colormap of the window specified, or asks you to click on the window whose colormap you wish to edit. Then, it displays a grid of the color cells in the installed colormap. Click on the cell you wish to edit. Then, in the upper right there will be a box containing the current color along with a label showing the current RGB values in X11 hex format.

To adjust the current color you can use one of the nine sliders, each controlling one of the RGB, HSV, or CMY values for the current color. Or you can click on a cell displayed in the palette to use its color. The button underneath the hex label switches between the three palettes: range, narrow, and wide.

The "select" button allows you to choose another cell to edit, the "cancel" button restores the current color to its original value, the "restore" button restores all the cells of the colormap to their original value, and the "quit" button exits out of cpicker.

SEE ALSO

pixedit(1)

AUTHOR

Mike Yang

BUGS

When clicking in a window to select its colormap, be sure to click in the window contents. Clicking in window manager real estate may or may not result in the correct colormap.

NAME

Gnuplot – a command driven interactive plotting program.

SYNOPSIS

gnuplot [options]

DESCRIPTION

Gnuplot is a command driven, interactive plotting program which can draw graphs either an X11 window or into a Postscript file which can then be printed on a Postscript-using printer. Output drivers for several other kinds of hardcopy plotters and personal computer graphic displays are also available.

It is extremely simple to display graphs of mathematical functions using gnuplot. For example, if you want to view the parabola defined by the equation:

```
y = x^2
```

then, after entering gnuplot and receiving the prompt "gnuplot>", simply type:

```
plot x**2
```

(gnuplot, like Fortran, uses the syntax x**y to denote exponentation).

The X values plotted range by default from -10 to +10. The graph scaling in both the X and Y directions is done automatically according to the values that need to be plotted.

Different commands exist to change the plotting style, manually set ranges and scaling, change output devices, etc. Also, there is a fairly good library of mathematical functions. All of these functions work in the complex plane, as gnuplot does all of its calculations using complex numbers. (Only real numbers actually get plotted, though. Complex domain plotting waits for a future enhancement).

There is a built in help system which you can access by typing "help" to the "gnuplot>" prompt. Typing "help ?" lists all the topics that the help system knows about. Examining the help for each of the topics, in the order that "help ?" lists them, shows you all of the material in the printed gnuplot manual.

gnuplot was written by Thomas Williams and Colin Kelley at Villanova University in Pennsylvania. The X window driver and some other features were added at MIT. There is at the moment no connection between the original gnuplot authors and the GNU project of the Free Software Foundation Inc. gnuplot's name might have been chosen because the authors intend to contribute the program to the GNU project someday, or it might be a coincidence. All attempts to date to get in touch with them and get this question answered have failed.

HARD COPY

Gnuplot will can formats plots for a postscript printer and send them to a file, the commands to do this are:

```
gnuplot> set terminal postscript
gnuplot> set output "<filename>"
gnuplot> replot
gnuplot> set terminal xwindow
```

The quotation marks are important. You may now simply print the file on a postscript printer.

OPTIONS

host:display
Normally, *Gnuplot* gets the host and display number to use from the environment variable "DISPLAY". One can, however specify them explicitly. The *host* specifies which machine to create the window on, and the *number* argument specifies the display number. For example, "orpheus:1" creates a shell window on display one on the machine orpheus.

–d *print standard .Xdefaults*
The 'default' .Xdefaults will be printed to standard out. You may then change the values to suit your particular needs.

–f fontname *Set the Font*
This sets the font used for the text.

–r *reverse video*
The foreground and background colors will be switched. The default colors are black on white.

X DEFAULTS

Gnuplot uses a number of standard default values.

ReverseVideo
If 'on', reverse the definition of foreground and background color.

Geometery
This uses a standard geometry string to set the position of the window that contains the plot. The height and width fields are ignored.

Font
The font to be used for all text in the plot window.

BUGS

There probabally are some.

AUTHOR

Thomas Williams, Colin Kelley

X Window Driver added by Paul Ruben (MIT - Project Athena).
X Defaults and command line options added by Chris Peterson (MIT - Project Athena).

NAME

ico – animate an icosahedron or other polyhedron

SYNOPSIS

ico [-display display] [-geometry geometry] [-r] [-d pattern] [-i] [-dbl] [-faces] [-noedges] [-sleep n] [-obj object] [-objhelp] [-colors color-list]

DESCRIPTION

Ico displays a wire-frame rotating polyhedron, with hidden lines removed, or a solid-fill polyhedron with hidden faces removed. There are a number of different polyhedra available; adding a new polyhedron to the program is quite simple.

OPTIONS

-r Display on the root window instead of creating a new window.

-d pattern
Specify a bit pattern for drawing dashed lines for wire frames.

-i Use inverted colors for wire frames.

-dbl Use double buffering on the display. This works for either wire frame or solid fill drawings. For solid fill drawings, using this switch results in substantially smoother movement. Note that this requires twice as many bit planes as without double buffering. Since some colors are typically allocated by other programs, most eight-bit-plane displays will probably be limited to eight colors when using double buffering.

-faces
Draw filled faces instead of wire frames.

-noedges
Don't draw the wire frames. Typically used only when -faces is used.

-sleep *n*
Sleep n seconds between each move of the object.

-obj *object*
Specify what object to draw. If no object is specified, an icosahedron is drawn.

-objhelp
Print out a list of the available objects, along with information about each object.

-colors *color color ...*
Specify what colors should be used to draw the filled faces of the object. If less colors than faces are given, the colors are reused.

ADDING POLYHEDRA

If you have the source to ico, it is very easy to add more polyhedra. Each polyhedron is defined in an include file by the name of objXXX.h, where XXX is something related to the name of the polyhedron. The format of the include file is defined in the file polyinfo.h. Look at the file objcube.h to see what the exact format of an objXXX.h file should be, then create your objXXX.h file in that format.

After making the new objXXX.h file (or copying in a new one from elsewhere), simply do a 'make depend'. This will recreate the file allobjs.h, which lists all of the objXXX.h files. Doing a 'make' after this will rebuild ico with the new object information.

SEE ALSO

X(1)

BUGS

A separate color cell is allocated for each name in the -colors list, even when the same name may be specified twice.

COPYRIGHT

NAME

kterm – terminal emulator for X

SYNOPSIS

kterm [*-toolkitoption* ...] [-option ...]

DESCRIPTION

The *kterm* program is a Kanji terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. It comes from *xterm* in the core programs of the distribution of the X Window System. The most of the functions are the same as original *xterm*'s, however, it has capabilities of displaying Kanji strings and handling of the status line, if compiled with -DKANJI or -DSTATUSLINE compile time options.

OPTIONS

The *kterm* terminal emulator accepts all of the standard *xterm* command line options along with the additional options listed below (if the option begins with a '+' instead of a '–', the option is restored to its default value):

–fk *kanji-font*
This option specifies a Kanji font to be used when displaying Kanji text. This font must be the same height and width as the ascii font. The default is "a14."

–fkb *kanji-font*
This option specifies a Kanji bold font to be used when displaying bold text. This font must be the same height and width as the kanji font. If no Kanji bold font is specified, it will be used as the normal font and the bold font will be produced by overstriking this font. The default is not specified.

–fk *kana-font*
This option specifies a Kana font, which may be used as GR in 8bit environment. This font is used if "ESC (I" is appeared in JIS Kanji mode, SS2 (0x8e) is appeared in EUC Kanji mode, and not used so frequntly in normal Japanese text. The default is "kana14."

–fkb *kana-font*
This option specifies a Kana bold font.

–km kanji-mode
This option specifies the Kanji code from the pty output. If kanji-mode is "jis", then it assumes the output is coded by JIS code, i.e., each Kanji string is proceeded by ESC-$-B or ESC-$-@ and each ascii string is proceeded by ESC-(-B or ESC-(-J. This mode does not require 8 bit passing tty modele because 7 bit encoding with appropriate escape sequences is used. If kanji-mode is "euc", then it assumes the output is coded by EUC. If kanji-mode is "sjis", then it assumes the output is coded by Shift-JIS code (which is the same as MicroSoft Kanji code). The default mode is "jis."

–sn By default, the status line is in reverse-video (relative to the rest of the window). This option causes the status line to be in normal video (the status line is still enclosed in a box).

–st This option causes the status line to be displayed on startup.

X DEFAULTS

The program understands all of the core xterm resource names and classes as well as:

kanjiFont (class **KanjiFont**)
Specifies the name of the kanji font. The default is "a14."

kanjiboldFont (class KanjiFont)
Specifies the name of the bold font. The default is not specified.

kanaFont (class KanaFont)
Specifies the name of the kana font. The default is "kana14."

kanaboldFont (class KanaFont)
Specifies the name of the bold font. The default is not specified.

kanjiMode (class KanjiMode)
Specifies the Kanji code of pty output. The default is "jis."

statusLine (class StatusLine)
Causes the status line to be displayed on startup.

statusNormal (class StatusNormal)
Specifies whether or not the last column bug in cursor should be worked around. The default is "false."

EMULATIONS

The VT102 emulation is fairly complete, but does not support the blinking character attribute nor the double-wide and double-size character sets. *Termcap*(5) entries that work with *xterm* include "kterm", "xterm", "vt102", "vt100" and "ansi", and *xterm* automatically searches the termcap file in this order for these entries and then sets the "TERM" and the "TERMCAP" environment variables.

POINTER USAGE

Kterm converts the specified text by the cut operation into JIS code regardless of the Kanji mode and then saves it to the Xserver. This convention allows us to cut and paste between different kterm's with differecnt Kanji mode.

SEE ALSO

xterm(1), resize(1), X(1), pty(4), tty(4)
"Xterm Control Sequences"

BUGS

Xterm will hang forever if you try to paste too much text at one time. It is both producer and consumer for the pty and can deadlock.

Variable-width fonts are not handled reasonably.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

The focus is considered lost if some other client (e.g., the window manager) grabs the pointer; it is difficult to do better without an addition to the protocol.

There needs to be a dialog box to allow entry of log file name and the COPY file name.

Many of the options are not resettable after *xterm* starts.

All programs should be written to use X directly; then we could eliminate this program.

Current *kterm* does not support the designate sequence of KANJI as "ESC $ (B" but "ESC $ B" which is still valid sequence in ISO 2022 (even if it seems to be historical reason :-)

COPYRIGHT

AUTHORS

Far too many people, including:

Katsuya Sano (Tokyo Inst. of Tech.), Michael Irie (Sony Corp.), Akira Kato (Tokyo Inst. of Tech.), Michiharu Ariza (Software Research Associates), Makoto Ishisone (Software Research Associates)

NAME

mkfontdir – create fonts.dir file from directory of font files.

SYNOPSIS

mkfontdir [directory-names]

DESCRIPTION

Mkfontdir For each directory argument, mkfontdir reads all of the font files in the directory searching for properties named "FONT", or (failing that) the name of the file stripped of its suffix. These are used as font names, which are written out to the file "fonts.dir" in the directory along with the name of the font file.

The kinds of font files read by mkfontdir depends on configuration parameters, but typically include SNF (suffix ".snf"), compressed SNF (suffix ".snf.Z"), BDF (suffix ".bdf"), and compressed BDF (suffix ".bdf.Z"). If a font exists in multiple formats, the most efficient format will be used.

FONT NAME ALIAES

The file "fonts.alias" which can be put in any directory of the font-path is used to map new names to existing fonts, and should be edited by hand. The format is straight forward enough, two white-space separated columns, the first containing aliases and the second containing font-name patterns.

When a font alias is used, the name it references is search for in the normal manner, looking through each font directory in turn. This means that the aliases need not mention fonts in the same directory as the alias file.

To embed white-space in either name, simply enclose them in double-quote marks, to embed double-quote marks (or any other character), preceed them with back-slash:

"magic-alias with spaces"	"\"font\name\" with quotes"
regular-alias	fixed

If the string "FILE_NAMES_ALIASES" stands alone on a line, each file-name in the directory (stripped of it's .snf suffix) will be used as an alias for that font.

USAGE

Xserver(1) looks for both "fonts.dir" and "fonts.alias" in each directory in the font path each time it is set (see xset(1)).

SEE ALSO

X(1), Xserver(1), xset(1)

NAME

muncher – draw interesting patterns in an X window

SYNOPSIS

muncher [-option ...]

OPTIONS

–r display in the root window

–s *seed* seed the random number seed

–v run in verbose mode

–q run in quite mode

–geometry *geometry*
define the initial window geometry; see *X(1)*.

–display *display*
specify the display to use; see *X(1)*.

DESCRIPTION

Muncher draws some interesting patterns in a window.

SEE ALSO

X(1)

BUGS

There are no known bugs. There are lots of lacking features.

COPYRIGHT

NAME

pixedit – pixel color editor for X11

SYNTAX

/usr/bin/X11/pixedit

DESCRIPTION

Pixedit makes temporary changes to the installed colormap, allowing the user to observe the effects. It's useful for trying to pick that perfect color or find the appropriate color combination for an application.

When pixedit first starts, the cursor changes to a crosshair. Click on the screen pixel whose color and whose colormap you wish to edit. Then, in the upper right there will be a box containing the current color along with a label showing the current RGB values in X11 hex format.

To adjust the current color you can use one of the nine sliders, each controlling one of the RGB, HSV, or CMY values for the current color. Or you can click on a cell displayed in the palette to use its color. The button underneath the hex label switches between the three palettes: range, narrow, and wide.

The "select" button allows you to choose another cell to edit, the "cancel" button restores the current color to its original value, the "restore" button restores all the cells of the colormap to their original value, and the "quit" button exits out of pixedit.

SEE ALSO

cpicker(1)

AUTHOR

Mike Yang

NAME

plaid – paint some plaid-like patterns in an X window

SYNOPSIS

plaid [-option ...]

OPTIONS

–b enable backing store for the window

–fg *color* This option specifies the color to use for the foreground of the window. The default is "white."

–bg *color*
This option specifies the color to use for the background of the window. The default is "black."

–bd *color*
This option specifies the color to use for the border of the window. The default is "white."

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–geometry *geometry*
define the initial window geometry; see *X(1)*.

–display *display*
specify the display to use; see *X(1)*.

DESCRIPTION

Plaid displays a continually changing plaid-like pattern in a window.

SEE ALSO

X(1)

BUGS

There are no known bugs. There are lots of lacking features.

COPYRIGHT

NAME

puzzle – 15-puzzle game for X

SYNOPSIS

puzzle [-option ...]

OPTIONS

–display *display*
This option specifies the display to use; see *X(1)*.

–geometry *geometry*
This option specifies the size and position of the puzzle window; see *X(1)*.

–size *WIDTHxHEIGHT*
This option specifies the size of the puzzle in squares.

–speed *num*
This option specifies the speed in tiles per second for moving tiles around.

–picture *filename*
This option specifies an image file containing the picture to use on the tiles. Try "mandrill.cm." This only works on 8-bit pseudo-color screens.

–colormap
This option indicates that the program should create its own colormap for the picture option.

DESCRIPTION

Puzzle with no arguments plays a 4x4 15-puzzle. The control bar has two boxes in it. Clicking in the left box scrambles the puzzle. Clicking in the right box solves the puzzle. Clicking the middle button anywhere else in the control bar causes puzzle to exit. Clicking in the tiled region moves the empty spot to that location if the region you click in is in the same row or column as the empty slot.

SEE ALSO

X(1)

BUGS

The picture option should work on a wider variety of screens.

COPYRIGHT

AUTHOR

Don Bennett, HP Labs

NAME

showsnf - print contents of an SNF file

SYNOPSIS

showsnf [-v] [-g] [-m] [-M] [-l] [-L] [-p#] [-u#]

DESCRIPTION

The *showsnf* utility displays the contents of font files in the Server Natural Format produced by *bsdtosnf*. It is usually only to verify that a font file hasn't been corrupted or to convert the individual glyphs into arrays of characters for proofreading or for conversion to some other format.

OPTIONS

-v This option indicates that character bearings and sizes should be printed.

-g This option indicates that character glyph bitmaps should be printed.

-m This option indicates that the bit order of the font is MSBFirst (most significant bit first).

-l This option indicates that the bit order of the font is LSBFirst (least significant bit first).

-M This option indicates that the byte order of the font is MSBFirst (most significant bit first).

-L This option indicates that the byte order of the font is LSBFirst (least significant bit first).

-p# This option specifies the glyph padding of the font (# is a number).

-u# This option specifies the scanline unit of the font (# is a number).

SEE ALSO

X(1), Xserver(1), bdftosnf(1)

BUGS

There is no way to just print out a single glyph.

COPYRIGHT

NAME

uwm - a window manager for X

SYNTAX

uwm [-display *display*] [-f *filename*]

DESCRIPTION

The *uwm* program is a window manager for X.

When *uwm* is invoked, it searches a predefined search path to locate any *uwm* startup files. If no startup files exist, *uwm* initializes its built-in default file.

If startup files exist in any of the following locations, it adds the variables to the default variables. In the case of contention, the variables in the last file found override previous specifications. Files in the *uwm* search path are:

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

To use only the settings defined in a single startup file, include the variables **resetbindings, resetmenus, resetvariables** at the top of that specific startup file.

OPTIONS

-f *filename*
Names an alternate file as a *uwm* startup file.

STARTUP FILE VARIABLES

Variables are typically entered first, at the top of the startup file. By convention, **resetbindings, resetmenus,** and **resetvariables** head the list.

autoselect/noautoselect
places the menu cursor in the first menu item. If unspecified, the menu cursor is placed in the menu header when the menu is displayed.

background=*color*
specifies the default background color for popup sizing windows, menus, and icons. The default is to use the WhitePixel for the current screen.

bordercolor=*color*
specifies the default border color for popup sizing windows, menus, and icons. The default is to use the BlackPixel for the current screen.

borderwidth=*pixels*
specifies the default width in pixels for borders around popup sizing windows, menus, and icons. The default is 2.

delta=*pixels* indicates the number of pixels the cursor is moved before the action is interpreted by the window manager as a command. (Also refer to the **delta** mouse action.)

foreground=*color*
specifies the default foreground color for popup sizing windows, menus, and icons. The default is to use the BlackPixel for the current screen.

freeze/nofreeze
locks all other client applications out of the server during certain window manager tasks, such as move and resize.

grid/nogrid displays a finely-ruled grid to help you position an icon or window during resize or move operations.

hiconpad=*pixels*
indicates the number of pixels to pad an icon horizontally. The default is five pixels.

hmenupad=*pixels*
indicates the amount of space in pixels that each menu item is padded to the left and to the right of the text.

borderwidth=*pixels*
indicates the width in pixels of the border surrounding icons.

iconfont=*fontname*
names the font that is displayed within icons. Font names for a given server can be obtained using *xlsfonts(1)*.

maxcolors=*number*
limits the number of colors the window manager can use in a given invocation. If set to zero, or not specified, *uwm* assumes no limit to the number of colors it can take from the color map. **maxcolors** counts colors as they are included in the file.

mborderwidth=*pixels*
indicates the width in pixels of the border surrounding menus.

normali/nonormali
places icons created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormali** the icon is placed exactly where the cursor leaves it.

normalw/nonormalw
places window created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormalw** the window is placed exactly where the cursor leaves it.

push=*number* moves a window *number* pixels or 1/*number* times the size of the window, depending on whether **pushabsolute** or **pushrelative** is specified. Use this variable in conjunction with **f.pushup, f.pushdown, f.pushright,** or **f.pushleft.**

pushabsolute/pushrelative
pushabsolute indicates that the number entered with push is equivalent to pixels. When an f.push (left, right, up, or down) function is called, the window is moved exactly that number of pixels.

pushrelative indicates that the number entered with the push variable represents a relative number. When an f.push function is called, the window is invisibly divided into the number of parts you entered with the push variable, and the window is moved one part.

resetbindings, resetmenus, and **resetvariables**
resets all previous function bindings, menus, and variable entries, specified in any startup file in the *uwm* search path, including those in the default environment. By convention, these variables are entered first in the startup file.

resizefont=*fontname*
identifies the font of the indicator that displays dimensions in the corner of the window as you resize windows. See *xlsfonts(1)* for obtaining font names.

resizerelative/noresizerelative
indicates whether or not resize operations should be done relative to moving edge or edges. By default, the dynamic rectangle uses the actual pointer location to define the new size.

reverse/noreverse
defines the display as black characters on a white background for the window manager windows and icons.

viconpad=*pixels* indicates the number of pixels to pad an icon vertically. Default is five pixels.

vmenupad=*pixels*
indicates the amount of space in pixels that the menu is padded above and below the text.

volume=*number*
increases or decreases the base level volume set by the *xset(1)* command. Enter an integer from 0 to 7, 7 being the loudest.

zap/nozap causes ghost lines to follow the window or icon from its previous default location to its new location during a move or resize operation.

BINDING SYNTAX

function=[*control key(s)*]:[*context*]:*mouse events*:" *menu name* "

Function and mouse events are required input. Menu name is required with the *f.menu* function definition only.

Function

f.beep emits a beep from the keyboard. Loudness is determined by the volume variable.

f.circledown causes the top window that is obscuring another window to drop to the bottom of the stack of windows.

f.circleup exposes the lowest window that is obscured by other windows.

f.continue releases the window server display action after you stop action with the **f.pause** function.

f.focus directs all keyboard input to the selected window. To reset the focus to all windows, invoke *f.focus* from the root window.

f.iconify when implemented from a window, this function converts the window to its respective icon. When implemented from an icon, f.iconify converts the icon to its respective window.

f.kill kills the client that created a window.

f.lower lowers a window that is obstructing a window below it.

f.menu invokes a menu. Enclose 'menu name' in quotes if it contains blank characters or parentheses.

f.menu=[*control key(s)*]:[*context*]:*mouse events*:" *menu name* "

f.move moves a window or icon to a new location, which becomes the default location.

f.moveopaque moves a window or icon to a new screen location. When using this function, the entire window or icon is moved to the new screen location. The grid effect is not used with this function.

f.newiconify allows you to create a window or icon and then position the window or icon in a new default location on the screen.

f.pause temporarily stops all display action. To release the screen and immediately update all windows, use the **f.continue** function.

f.pushdown moves a window down. The distance of the push is determined by the push variables.

f.pushleft moves a window to the left. The distance of the push is determined by the push variables.

f.pushright moves a window to the right. The distance of the push is determined by the push variables.

f.pushup moves a window up. The distance of the push is determined by the push variables.

f.raise raises a window that is being obstructed by a window above it.

f.refresh results in exposure events being sent to the window server clients for all unobscured or partially obscured windows. The windows will not refresh correctly if the exposure events are not handled properly.

f.resize resizes an existing window. Note that some clients, notably editors, react unpredictably if you resize the window while the client is running.

f.restart causes the window manager application to restart, retracing the *uwm* search path and initializing the variables it finds.

Control Keys

By default, the window manager uses meta as its control key. It can also use ctrl, shift, lock, or null (no control key). Control keys must be entered in lower case, and can be abbreviated as: c, l, m, s for ctrl, lock, meta, and shift, respectively.

You can bind one, two, or no control keys to a function. Use the bar (|) character to combine control keys.

Note that client applications other than the window manager use the shift as a control key. If you bind the shift key to a window manager function, you can not use other client applications that require this key.

Context

The context refers to the screen location of the cursor when a command is initiated. When you include a context entry in a binding, the cursor must be in that context or the function will not be activated. The window manager recognizes the following four contexts: icon, window, root, (null).

The root context refers to the root, or background window, A (null) context is indicated when the context field is left blank, and allows a function to be invoked from any screen location. Combine contexts using the bar (|) character.

Mouse Buttons

Any of the following mouse buttons are accepted in lower case and can be abbreviated as l, m, or r, respectively: left, middle, right.

With the specific button, you must identify the action of that button. Mouse actions can be:

down function occurs when the specified button is pressed down.

up function occurs when the specified button is released.

delta indicates that the mouse must be moved the number of pixels specified with the delta variable before the specified function is invoked. The mouse can be moved in any direction to satisfy the delta requirement.

MENU DEFINITION

After binding a set of function keys and a menu name to **f.menu**, you must define the menu to be invoked, using the following syntax:

menu = " *menu name* **" {**
"item name" : "action"
.
.
.
}

Enter the menu name exactly the way it is entered with the **f.menu** function or the window manager will not recognize the link. If the menu name contains blank strings, tabs or parentheses, it must be quoted here and in the f.menu function entry. You can enter as many menu items as your screen is long. You cannot scroll within menus.

Any menu entry that contains quotes, special characters, parentheses, tabs, or strings of blanks must be enclosed in double quotes. Follow the item name by a colon (:).

Menu Action

Window manager functions
Any function previously described. E.g., **f.move** or **f.iconify**.

Shell commands
Begin with an exclamation point (!) and set to run in background. You cannot include a new line character within a shell command.

Text strings
Text strings are placed in the window server's cut buffer.

Strings starting with an up arrow (^) will have a new line character appended to the string after the up arrow (^) has been stripped from it.

Strings starting with a bar character (|) will be copied as is after the bar character (|) has been stripped.

Color Menus

Use the following syntax to add color to menus:

menu = "*menu name***" (***color1:color2:color3:color4***) {**
"item name" : (color5 :color6) : " action "
.
.
.
}

color1 Foreground color of the header.

color2 Background color of the header.

color3 Foreground color of the highlighter, the horizontal band of color that moves with the cursor within the menu.

color4 Background color of the highlighter.

color5 Foreground color for the individual menu item.

color6 Background color for the individual menu item.

Color Defaults

Colors default to the colors of the root window under any of the following conditions:

1) If you run out of color map entries, either before or during an invocation of *uwm*.

2) If you specify a foreground or background color that does not exist in the RGB color database of the server (see */usr/lib/X11/rgb.txt* for a sample) both the foreground and background colors default to the root window colors.

3) If you omit a foreground or background color, both the foreground and background colors default to the root window colors.

4) If the total number of colors specified in the startup file exceeds the number specified in the *maxcolors* variable.

5) If you specify no colors in the startup file.

EXAMPLES

The following sample startup file shows the default window manager options:

```
# Global variables
#
resetbindings;resetvariables;resetmenus
autoselect
delta=25
freeze
grid
hiconpad=5
hmenupad=6
iconfont=oldeng
menufont=timrom12b
resizefont=9x15
viconpad=5
vmenupad=3
volume=7
#
# Mouse button/key maps
#
# FUNCTION   KEYS CONTEXT BUTTON   MENU(if any)
# ========  ==== ======= ======  ============
f.menu =     meta :   :left down  :"WINDOW OPS"
f.menu =     meta :   :middle down :"EXTENDED WINDOW OPS"
f.move =     meta :w | i :right down
f.circleup = meta :root :right down
#
# Menu specifications
#
menu = "WINDOW OPS" {
"(De)Iconify":  f.iconify
Move:           f.move
```

```
Resize:         f.resize
Lower:          f.lower
Raise:          f.raise
}

menu = "EXTENDED WINDOW OPS" {
Create Window:                          !"xterm &"
Iconify at New Position: f.lowericonify
Focus Keyboard on Window:    f.focus
Freeze All Windows:          f.pause
Unfreeze All Windows:        f.continue
Circulate Windows Up:        f.circleup
Circulate Windows Down:              f.circledown
}
```

RESTRICTIONS

The color specifications have no effect on a monochrome system.

FILES

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

SEE ALSO

X(1), Xserver(1), xset(1), xlsfonts(1)

COPYRIGHT

AUTHOR

M. Gancarz, DEC Ultrix Engineering Group, Merrimack, New Hampshire, using some algorithms originally by Bob Scheifler, MIT Laboratory for Computer Science.

NAME

X11 – a portable, network transparent window system

SYNOPSIS

X+ is the Ardent implementation of the X window system for the Titan computer. X is a network transparent window system developed at MIT which runs under a wide variety of operating systems. The standard distribution from MIT works on Ultrix-32 Version 1.2 (and higher), 4.3BSD Unix, SunOS 3.2 (and higher), HP-UX 6.01, and DOMAIN/IX 9.7. In addition, many vendors support the X Window System under other operating systems.

THE OFFICIAL NAMES

The official names of the software described herein are:

X
X Window System
X Version 11
X Window System, Version 11
X11

Note that the phrases X.11, X-11, X Windows or any permutation thereof, are explicity excluded from this list and should not be used to describe the X Window System (window system should be thought of as one word).

X Window System is a trademark of the Massachusetts Institute of Technology.

DESCRIPTION

X window system servers run on computers with bitmap displays. The server distributes user input to, and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of functions, see the *Xlib - C Language X Interface* manual, the *X Window System Protocol* specification, and various toolkit documents.

When you first log in on a display running X, you are usually using the *xterm(1)* terminal emulator program. You need not learn anything extra to use a display running X as a terminal beyond moving the mouse cursor into the login window to log in normally.

The core X protocol provides mechanism, not policy. Windows are manipulated (including moving, resizing and iconifying) not by the server itself, but by a separate program called a "window manager" of your choosing. This program is simply another client and requires no special privileges. If you don't like the ones that are supplied (see *awm(1)* and *uwm(1)*), you can write your own.

The number of programs that use X is growing rapidly. Of particular interest are: a terminal emulator (*xterm(1)*), window managers (*awm(1)* and *uwm(1)*), a mailer reader (*xmh(1)*), a bitmap editor (*bitmap(1)*), an access control program (*xhost(1)*), user preference setting programs (*xset(1)*, *xsetroot(1)*, and *xmodmap(1)*), a load monitor (*xload(1)*), clock (*xclock(1)*), a font displayer (*xfd(1)*), and a protocol translator for running X10 programs (*x10tox11(1)*).

DISPLAY SPECIFICATION

When you first log in, the environment variable DISPLAY will be set to a string specifying the name of the machine on which the server is running, a number indicating which of possibly several servers to use, and possibly a number indicating the default screen of the server (usually this is omitted and defaults to 0). By convention, servers on a particular machine are numbered starting with zero. The format of the DISPLAY string depends on the type of communications channel used to contact the server.

The following connection protocols are supported:

TCP/IP

DISPLAY should be set to "*host:dpy.screen*" where *host* is the symbolic name of the machine (e.g. expo), *dpy* is the number of the display (usually 0), and *screen* is the number of the screen. The *screen* and preceding period are optional, with the default value being zero (0). Full Internet domain names (e.g. expo.lcs.mit.edu) are allowed for the host name.

Unix domain

DISPLAY should be set to "unix:*dpy.screen*", where *dpy* is the display number and *screen* is the screen number; *screen* and the preceding period are optional, with the default value being zero (0).

Most programs accept a command line argument of the form "-display *display*" that can be used to override the DISPLAY environment variable.

GEOMETRY SPECIFICATION

One of the advantages of using window systems over hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running, most applications accept a command line argument that is treated as the prefered size and location for this particular application's window.

This argument, usually specified as "-geometry WxH+X+Y," indicates that the window should have a width of W and height of H (usually measured in pixels or characters, depending on the application), and the upper left corner X pixels to the right and Y pixels below the upper left corner of the screen (origin (0,0)). "WxH" can be omitted to obtain the default application size, or "+X+Y" can be omitted to obtain the default application position (which is usually then left up to the window manager or user to choose). The X and Y values may be negative to position the window off the screen. In addition, if minus signs are used instead of plus signs (e.g. WxH-X-Y), then (X,Y) represents the location of the lower right hand corner of the window relative to the lower right hand corner of the screen.

By combining plus and minus signs, the window may be place relative to any of the four corners of the screen. For example:

555x333+11+22

This will request a window 555 pixels wide and 333 pixels tall, with the upper left corner located at (11,22).

300x200-0+0

This will request a window measuring 300 by 200 pixels in the upper right hand corner of the screen.

48x48--5--10

This will request a window measuring 48 by 48 pixels whose lower right hand corner is 5 pixel off the right edge and the screen and 10 pixels off the bottom edge.

COMMAND LINE ARGUMENTS

Most X programs attempt to use a common set of names for their command line arguments. The X Toolkit automatically handles the following arguments:

-bg *color*, **-background** *color*
Either option specifies the color to use for the window background.

-bd *color*, **-bordercolor** *color*
Either option specifies the color to use for the window border.

-bw *number*, **-borderwidth** *number*
Either option specifies the width in pixels of the window border.

-display *display*
This option specifies the name of the X server to use.

-fg *color*, **-foreground** *color*
Either option specifies the color to use for text or graphics.

-fn *font*, **-font** *font*
Either option specifies the font to use for displaying text.

-geometry *geometry*
This option specifies the initial size and location of the window.

-iconic
This option indicates that application should start out in an iconic state. Note that how this state is represented is controlled by the window manager that the user is running.

-name
This option specifies the name under which resources for the application should be found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.

-rv, **-reverse**
Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.

+rv
This option indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.

-synchronous
This option indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since *Xlib* normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.

-title *string*
This option specifies the title to be used for this window. This information is sometimes used by a window manager to provide some sort of header identifying the window.

-xrm *resourcestring*
This option specifies a resource name and value to override any defaults. It is also very useful for setting resources that don't have explicitly command

line arguments.

RESOURCES

To make the tailoring of applications to personal preferences easier, X supports several mechanisms for storing default values for program resources (e.g. background color, window title, etc.) Resources are specified as strings of the form *"name*subname*subsubname...: value"* (see the *Xlib* manual section *Using the Resource Manager* for more details) that are loaded into a client when it starts up. The *Xlib* routine *XGetDefault(3X)* and the resource utilities within the X Toolkit obtain resources from the following sources:

RESOURCE_MANAGER root window property
Any global resources that should be available to clients on all machines should be stored in the RESOURCE_MANAGER property on the root window using the *xrdb(1)* program.

application-specific directory
Any application- or machine-specific resources can be stored in the class resource files located in the directory /usr/X11/app-defaults in the standard distribution).

XENVIRONMENT
Any user- and machine-specific resources may be specified by setting the XENVIRONMENT environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, the X Toolkit looks for a file named .Xdefaults-*hostname*, where *hostname* is the name of the host where the application is executing.

–xrm *resourcestring*
Applications that use the X Toolkit can have resources specified from the command line. The *resourcestring* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of **–xrm** arguments may be given on the command line.

Program resources are organized into groups called "classes," so that collections of individual "instance" resources can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an upper case letter. Multiple word resources are concatentated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit will have at least the following resources:

background (class **Background**)
This resource specifies the color to use for the window background.

borderWidth (class **BorderWidth**)
This resource specifies the width in pixels of the window border.

borderColor (class **BorderColor**)
This resource specifies the color to use for the window border.

Most X Toolkit applications also have the resource **foreground** (class **Foreground**), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set Background and Foreground classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources.

When a named resource is unavailable (for example, a color named chartrusse or a font named teeneyweeney), normally no error message will be printed; whether or not useful results ensue is dependent on the particular application. If you wish to see error messages (for example, if an application is failing for an unknown reason), you may specify the value "on" for the resource named "StringConversionWarnings." If you want such warnings for all applications, specify "*StringConversionWarnings:on" to the resource manager. If you want warnings only for a single application named "zowie", specify "zowie*StringConversionWarnings:on" to the resource manager.

DIAGNOSTICS

The default error handler uses the Resource Manager to build diagnostic messages when error conditions arise. The default error database is stored in the file XErrorDB in the directory /usr/X11. If this file is not installed, error messages will tend to be somewhat cryptic.

SEE ALSO

xterm(1), bitmap(1), awm(1), uwm(1), x10tox11(1), xcalc(1), xclock(1), xedit(1), xfd(1), xhost(1), xinit(1), xload(1), xlogo(1), xlsfonts(1), xmh(1), xmodmap(1), xpr(1), xprop(1), xrdb(1), xrefresh(1), xset(1), xsetroot(1), xwd(1), xwininfo(1), xwud(1), X(1).
Xlib – C Language X Interface, X Toolkit Intrinsics - C Language X Interface

COPYRIGHT

The following copyright and permission notice outlines the rights and restrictions covering most parts of the standard distribution of the X Window System from MIT. Other parts have additional or different copyrights and permissions; see the individual source files.

AUTHORS

It is no longer feasible to list all people who have contributed something to X, but see doc/contributors in the standard sources.

NAME

X – X Window System server

SYNOPSIS

X displaynumber [-option ...]

DESCRIPTION

X is the Ardent window system server. Normally it is started from the *xstart(1)* command. The *displaynumber* argument is used by clients in their DISPLAY environment variables to indicate which server to contact. Currently the only valid value for *displaynumber* is 0.

The Ardent X+ server has support for the following protocols:

TCP/IP
The server listens on port htons(6000+N), where N is the display number.

Unix Domain
The name for the socket is X11-unix:0.

When the Ardent X+ server starts up, it takes over the display. Normally when the server is started an xterm is used to display messages directed to the console. While the X server is running you cannot log into the console.

OPTIONS

The following options can be given on the command line to the X+ server:

–a *number*
sets pointer acceleration (i.e. the ratio of how much is reported to how much the user actually moved the pointer).

–base use base graphics board only.

–bs disables backing store support on all screens.

–c turns off key-click.

c *volume* sets key-click volume (allowable range: 0-8).

–f *volume*
sets feep (bell) volume (allowable range: 0-7).

–logo turns on the X Window System logo display in the screen-saver. There is currently no way to change this from a client.

nologo turns off the X Window System logo display in the screen-saver. There is currently no way to change this from a client.

–p *minutes*
sets screen-saver pattern cycle time in minutes.

–pseudo
set default visual class to PseudoColor (and default depth to 8).

–r turns off auto-repeat.

r turns on auto-repeat.

–s *minutes*
sets screen-saver timeout time in minutes.

–ssp specify screen-saver program.

–stereo enable stereo screen.

–su disables save under support on all screens.

-t *numbers*
sets pointer acceleration threshold in pixels (i.e. after how many pixels pointer acceleration should take effect).

-to *seconds*
sets default screensaver timeout in seconds.

v sets video-on screen-saver preference.

-v sets video-off screen-saver preference

-wm WhenMapped default backing-store

-co *filename*
sets name of RGB color database

-help prints a usage message

-fp *fontPath*
sets the search path for fonts

-fc *cursorFont*
sets default cursor font

-fn *font* sets the default font

-bp *pixel-value*
set default black-pixel color

-wp *pixel-value*
set default white-pixel color

SECURITY

X uses an access control list for deciding whether or not to accept a connection from a given cleint. This list initially consists of the machine on which the server is running, and any hosts listed in the file */etc/X*.hosts* (where * is the display number). This file should contain one line per host name, with no white space.

The user can manipulate a dynamic form of this list in the server using the *xhost(1)* program from the same machine as the server.

Unlike some window systems, *X* does not have any notion of window operation permissions or place any restrictions on what a client can do; if a program can connect to a display, it has full run of the screen.

SIGNALS

The *X* server attaches special meaning to the following signals:

SIGHUP
This signal causes the server to close all existing connections, free all resources, and restore all defaults. It is sent by the display manager whenever the main user's main application (usually an *xterm* or window manager) exits to force the server to clean up and prepare for the next user.

SIGTERM
This signal causes the server to exit cleanly.

FONTS

Fonts are usually stored as individual files in directories. The list of directories in which the server looks when trying to open a font is controlled by the *font path*. Although most sites will choose to have the server start up with the appropriate font path (using the *-fp* option mentioned above), it can be overridden using the *xset* program. The default font path for the Ardent X server contains the single directory: */usr/X11/fonts*.

Font databases are created by running the *mkfontdir* program in the directory containing the compiled versions of the fonts (the *.snf* files). Whenever fonts are added to a directory, *mkfontdir* should be rerun so that the server can find the new fonts. If *mkfontdir* **is not run, the server will not be able to find any fonts in the directory.**

PROGRAMMABLE SCREEN SAVER

The Ardent X server provides support for programmable screen savers. The **-ssp** option allows you to set the screen saver program used by the server. The currently available options are *ardentlogo* which moves the Ardent logo around the screen and *stringart* which draws some nice vector patterns. You can also define your own.

A screen saver program is invoked with two arguments: the window id in the form 0xhhhh and the screen saver interval in milliseconds. The window passed to the screen saver program will cover the entire screen and will either be a 24 bit direct color window or an 8 bit pseudo color window. The program should be able to handle both cases.

STEREO

The Ardent X Server provides support for stereo graphics. If the -stereo option is used then pressing the ALT and F2 keys on the titan keyboard will display the stereo screen. Applications wishing to display on the stereo screen should create windows on screen 1. This screen does not have a square aspect ratio (since half the y resolution is used for the left eye and half for the right). Pressing ALT F2 key toggles between the stereo and normal screens. Stereo viewing requires special hardware see your local Ardent representative for details.

DIAGNOSTICS

Too numerous to list them all. If run from *start(1)*, errors are logged in the file */usr/adm/X*msgs*,

FILES

/etc/X*.hosts	Initial access control list
/usr/X11/fonts	Font directory
/usr/X11/rgb/rgb.txt	Color database
/usr/adm/X*msgs	Error log file

SEE ALSO

X11(1), xstart(1), xinit(1), xterm(1), awm(1), xhost(1), xset(1), xsetroot(1).

BUGS

The option syntax is inconsistent with itself and *xset(1)*.

The acceleration option should take a numerator and a denominator like the protocol.

If *X* dies before its clients, new clients won't be able to connect until all existing connections have their TCP TIME_WAIT timers expire.

The color database is missing a large number of colors. However, there doesn't seem to be a better one available that can generate RGB values tailorable to particular displays.

COPYRIGHT

AUTHORS

The sample server was originally written by Susan Angebranndt, Raymond Drewry, Philip Karlton, and Todd Newman, with support from a cast of thouands.

The Ardent X+ server was implemented by: Mark Patrick and John Reiser.

NAME

x10tox11 – X version 10 to version 11 protocol converter

SYNOPSIS

x10tox11 [-display host:display]

DESCRIPTION

x10tox11 masquerades as an X Window System Version 10 server. It enables an X Version 10 client to run unchanged under X Version 11 by converting Version 10 requests into appropriate Version 11 requests, and by converting all Version 11 events received from the server into Version 10 events. From the perspective of Version 10 clients, all Version 11 clients look like Version 10 clients; and from the perspective of Version 11 clients, all Version 10 clients just look like Version 11 clients. Hence, a Version 11 window manager can manipulate Version 10 clients.

This program does NOT use the X10 *libnest* ddX library. It does actual protocol translation, rather than simply using X11 graphics calls to implement X10 low level operations. As a result, it is both faster and more robust than the X10 Xnest server.

TYPICAL USAGE

The protocol converter must be run after the X11 server is running and should be run in the background:

x10tox11 &

The program will continue to run until you intentionally kill it or the X11 server is shut down.

OPTIONS

-display host:display

Standard option for specifying the X11 display to which you wish to be connected. By default, it uses unix:0.0. Note that *x10tox11* will always pretend to be an X10 server with the same display number as the X11 server to which it connects. For example, if the DISPLAY environment variable or the *-display* option specifies *fizzle:1.0*, then *x10tox11* will connect to the X11 server on host *fizzle* for display 1 and then will pretend to the the X10 server for display 1. Consequently, your X10 clients will expect to have the environment variable DISPLAY set to *fizzle:1* (but they should still work even if your X10 clients use *fizzle:1.0*).

MinimumTileSize=n

Set minimum acceptable tile size to *n*. There is a difference in semantics between X10's XQueryShape and X11's XQueryBestSize such that X11 will allow any tile size but will return the optimum whereas X10 enforced a minimum tile size. Usually this minimum tile size was 16 and this is the default for *x10tox11*. If you find that this makes your X10 clients break, then you can override it with this option.

help

This prints out a usage message and exits.

NoOverrideRedirect

This instructs *x10tox11* to make every effort not to use OverrideRedirect when creating and mapping windows. Normally, *x10tox11* creates all windows with the OverrideRedirect attribute set to true. Placing this option on the command line will cause *x10tox11* not to use OverrideRedirect except for windows that look like they might be menus. This will allow window managers that provide title bars to do so. Unfortunately, it is impossible to determine ahead of time what an X10 client intends to do with windows. In addition, X10 clients are known to spontaneously unmap their windows

which upsets X11 window managers unless the OverrideRedirect attribute is true. Further, some X11 window managers may refuse to resize or move windows that are marked with OverrideRedirect. This may can be fixed to some extent when an Inter Client Communications Convention Manual (ICCCM) is adopted by the X11 community.

SEE ALSO

X(1), Xserver(1)

BUGS

There are limitations with respect to emulating Version 10 through a Version 11 server. See the file /usr/lib/X/x10tox11.help for more details.

Some window managers may refuse to move, resize or perform any operations on X10 client windows because, by default,

If the source is compiled with certain flags, there are significant debugging facilities available. Using the *help* option will tell you whether debugging facilities are available. *x10tox11* marks them with OverrideRedirect. See **OPTIONS** above.

COPYRIGHT

AUTHOR

Todd Brunhoff, Visual Systems Laboratory, Tektronix.

NAME

xbiff – mailbox flag for X

SYNOPSIS

xbiff [*-toolkitoption* ...] [-option ...]

DESCRIPTION

The *xbiff* program displays a little image of a mailbox. When there is no mail, the flag on the mailbox is down. When mail arrives, the flag goes up and the mailbox beeps. By default, pressing any mouse button in the image forces *xbiff* to remember the current size of the mail file as being the "empty" size and to lower the flag.

This program is nothing more than a wrapper around the Athena *Mailbox* widget.

OPTIONS

Xbiff accepts all of the standard X Toolkit command line options along with the additional options listed below:

–help This option indicates that a brief summary of the allowed options should be printed on the standard error.

–update *seconds*
This option specifies the frequency in seconds at which *xbiff* should update its display. If the mailbox is obscured and then exposed, it will be updated immediately. The default is 60 seconds.

–file *filename*
This option specifies the name of the file which should be monitored. By default, it watches /usr/spool/mail/*username*, where *username* is your login name.

–volume *percentage*
This option specifies how loud the bell should be rung when new mail comes in.

The following standard X Toolkit command line arguments are commonly used with *xbiff:*

–display *display*
This option specifies the X server to contact.

–geometry *geometry*
This option specifies the prefered size and position of the mailbox window. The mailbox is 48 pixels wide and 48 pixels high and will be centered in the window.

–bg *color*
This option specifies the color to use for the background of the window. The default is "white."

–bd *color*
This option specifies the color to use for the border of the window. The default is "black."

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–fg *color* This option specifies the color to use for the foreground of the window. The default is "black."

–rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Mailbox* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

checkCommand (class **CheckCommand**)
Specifies a shell command to be executed to check for new mail rather than examining the size of **file**. The specified string value is used as the argument to a *system*(3) call and may therefore contain i/o redirection. A successful (zero) exit status should indicate that new mail is waiting.

file (class **File**)
Specifies the name of the file to monitor. The default is to watch /usr/spool/mail/*username*, where *username* is your login name.

onceOnly (class **Boolean**)
Specifies that the bell is only rung the first time new mail is found and is not rung again until at least one interval has passed with no mail waiting. The window will continue to indicate the presence of new mail until it has been retrieved.

width (class **Width**)
Specifies the width of the mailbox.

height (class **Height**)
Specifies the height of the mailbox.

update (class **Interval**)
Specifies the frequency in seconds at which the mail should be checked.

volume (class **Volume**)
Specifies how load the bell should be rung. The default is 33 percent.

foreground (class **Foreground**)
Specifies the color for the foreground. The default is "black" since the core default for background is "white."

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background should be reversed.

ACTIONS

The *Mailbox* widget provides the following actions for use in event translations:

check() This action causes the widget to check for new mail and display the flag appropriately.

unset() This action causes the widget to lower the flag until new mail comes in.

set() This action causes the widget to raise the flag until the user resets it.

The default translation is

```
<ButtonPress>: unset()
```

ENVIRONMENT

DISPLAY
to get the default host and display number.

XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X(1), xrdb(1), stat(2)

BUGS

The mailbox bitmaps are ugly.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium
Additional hacks by Ralph Swick, DEC/MIT Project Athena

NAME

xcalc – scientific calculator for X

SYNOPSIS

xcalc [-display *display*] [-bw *pixels*] [-stip] [-rv] [-rpn] [-analog] [-geometry *geometry*]

DESCRIPTION

Xcalc is a scientific calculator desktop accessory that can emulate a TI-30, an HP-10C, and a slide rule.

OPTIONS

–display *displayname*
This option specifies the X server to contact.

–geometry *geometry*
This option specifies the size and placement of the top level window. By default, the minimum size will be used. Note that your window manager may require you to place it explicitly anyway.

–fg *color* This option specifies the foreground color to use.

–bg *color*
This option specifies the background color to use.

–bw *pixels*
This option specifies the border width in pixels.

–stip This option indicates that the background of the calculator should be drawn using a stipple of the foreground and background colors. On monochrome displays this makes for a nicer display.

–rv This option indicates that reverse video should be used.

–rpn This option indicates that Reverse Polish Notation should be used. In this mode the calculator will look and behave like an HP-10C. Without this flag, it will emulate a TI-30.

–analog This option indicates that a slide rule should be used.

OPERATION

Pointer Usage: Most operations are done with the Button1 (usually leftmost button on the pointer). The only exception is that pressing the AC key on the TI calculator with Button3 (usually on the right) will exit the calculator.

Key Usage (Normal mode): The number keys, the +/- key, and the +, -, *, /, and = keys all do exactly what you would expect them to. It should be noted that the operators obey the standard rules of precedence. Thus, entering "3+4*5=" results in "23", not "35". The parentheses can be used to override this. For example, "(1+2+3)*(4+5+6)=" results in "6*15=90". The non-obvious keys are detailed below.

1/x replaces the number in the display with its reciprocal.

x^2 squares the number in the display.

SQRT takes the square root of the number in the display.

CE/C when pressed once, clears the number in the display without clearing the state of the machine. Allows you to re-enter a number if you screw it up. Pressing it twice clears the state, also.

AC clears everything, the display, the state, the memory, everything. Pressing it with the right button 'turns off' the calculator, in that it exits the program. Somewhat more equivalent to throwing the calculator in the trash, if we were to pursue the analogy.

INV inverts the meaning of the function keys. See the individual function keys for details.

sin computes the sine of the number in the display, as interpreted by the current DRG mode (see DRG, below). If inverted, it computes the arcsine.

cos computes the cosine, or arccosine when inverted.

tan computes the tangent, or arctangent when inverted.

DRG changes the DRG mode, as indicated by 'DEG', 'RAD', or 'GRAD' at the bottom of number window of the calculator. When in 'DEG' mode, numbers in the display are taken as being degrees. In 'RAD' mode, numbers are in radians, and in 'GRAD' mode, numbers are in gradians. When inverted, the DRG key has the nifty feature of converting degrees to radians to gradians and vice-versa. Example: put the calculator into 'DEG' mode, and type "45 INV DRG". The display should now show something along the lines of ".785398", which is 45 degrees converted to radians.

e the constant 'e'. (2.7182818...)

EE used for entering exponential numbers. For example, to enter "-2.3E-4" you'd type "2 . 3 +/- EE 4 +/-"

log calculates the log (base 10) of the number in the display. When inverted, it raises "10.0" to the number in the display. For example, typing "3 INV log" should result in "1000".

ln calcuates the log (base e) of the number in the display. When inverted, it raises "e" to the number in the display. For example, typing "e ln" should result in "1"

y^x raises the number on the left to the power of the number on the right. For example "2 y^x 3 =" results in "8", which is 2^3. For a further example, "(1+2+3) y^x (1+2) =" equals "6 y^x 3" which equals "216".

PI the constant 'pi'. (3.1415927....)

x! computes the factorial of the number in the display. The number in the display must be an integer in the range 0-500, though, depending on your math library, it might overflow long before that.

STO copies the number in the display to the memory location.

RCL copies the number from the memory location to the display.

SUM adds the number in the display to the number in the memory location.

EXC swaps the number in the display with the number in the memory location.

Key Usage (RPN mode): The number keys, CHS (change sign), +, -, *, /, and ENTR keys all do exactly what you would expect them to do. Many of the remaining keys are the same as in normal mode. The differences are detailed below.

<- is a backspace key that can be used while typing a number. It will erase digits from the display.

ON clears everything, the display, the state, the memory, everything. Pressing it with the right button 'turns off' the calculator, in that it exits the program. Somewhat more equivalent to throwing the calculator in the trash, if we were to pursue the analogy.

INV inverts the meaning of the function keys. This would be the "f" key on an HP calculator, but xcalc does not have the resolution to display multiple legends on each key. See the individual function keys for details.

10^x raises "10.0" to the number in the top of the stack. When inverted, it calculates the log (base 10) of the number in the display.

e^x raises "e" to the number in the top of the stack. When inverted, it calcuates the log (base e) of the number in the display.

STO copies the number in the top of the stack to a memory location. There are 10 memory locations. The desired memory is specified by following this key with pressing a digit key.

RCL pushes the number from the specified memory location onto the stack.

SUM adds the number on top of the stack to the number in the specified memory location.

x:y exchanges the numbers in the top two stack positions.

R v rolls the stack downward. When inverted, it rolls the stack upward.

blank these keys were used for programming functions on the HP11-C. Their functionality has not been duplicated here.

KEYBOARD EQUIVALENTS

If you have the pointer in the xcalc window, you can use the keyboard to speed entry, as almost all of the calculator keys have a keyboard equivalent. The number keys, the operator keys, and the parentheses all have the obvious equivalent. The less-obvious equivalents are as follows:

```
n: +/-        !: x!
p: PI         e: EE
l: ln        ^: y^x
i: INV        s: sin
c: cos        t: tan
d: DRG      BS, DEL: CE/C ("<-" in RPN mode)
CR: ENTR      q: quit
```

COLOR USAGE

Xcalc uses a lot of colors, given the opportunity. In the default case, it will just use two colors (Foreground and Background) for everything. This works out nicely. However, if you're a color fanatic you can specify the colors used for the number keys, the operator (+-*/=) keys, the function keys, the display, and the icon.

X DEFAULTS

The program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

BorderWidth
Specifies the width of the border. The default is 2.

ReverseVideo
Indicates that reverse video should be used.

Stipple Indicates that the background should be stippled. The default is "on" for monochrome displays, and "off" for color displays.

Mode Specifies the default mode. Allowable values are *rpn, analog*.

Foreground
Specifies the default color used for borders and text.

Background
Specifies the default color used for the background.

NKeyFore, NKeyBack
Specifies the colors used for the number keys.

OKeyFore, OKeyBack
Specifies the colors used for the operator keys.

FKeyFore, FKeyBack
Specifies the colors used for the function keys.

DispFore, DispBack
Specifies the colors used for the display.

IconFore, IconBack
Specifies the colors used for the icon.

EXAMPLES

If you're running on a monochrome display, you shouldn't need any .Xdefaults entries for xcalc. On a color display, you might want to try the following in normal mode:

```
xcalc.Foreground:       Black
xcalc.Background:       LightSteelBlue
xcalc.NKeyFore:         Black
xcalc.NKeyBack:         White
xcalc.OKeyFore:         Aquamarine
xcalc.OKeyBack:         DarkSlateGray
xcalc.FKeyFore:         White
xcalc.FKeyBack:         #900
xcalc.DispFore:         Yellow
xcalc.DispBack:         #777
xcalc.IconFore:         Red
xcalc.IconBack:         White
```

SEE ALSO

X(1), xrdb(1)

BUGS

The calculator doesn't resize.

The slide rule and HP mode may or may not work correctly.

This application should really be implemented with the X Toolkit. It would make a very good example of a compound widget.

COPYRIGHT

AUTHORS

John Bradley, University of Pennsylvania
Mark Rosenstein, MIT Project Athena

NAME

xclock – analog / digital clock for X

SYNOPSIS

xclock [*-toolkitoption* ...] [-option ...]

DESCRIPTION

The *xclock* program displays the time in analog or digital form. The time is continuously updated at a frequency which may be specified by the user. This program is nothing more than a wrapper around the Athena Clock widget.

OPTIONS

Xclock accepts all of the standard X Toolkit command line options along with the additional options listed below:

–help This option indicates that a brief summary of the allowed options should be printed on the standard error.

–analog This option indicates that a conventional 12 hour clock face with tick marks and hands should be used. This is the default.

–digital This option indicates that a 24 hour digital clock should be used.

–chime This option indicates that the clock should chime once on the half hour and twice on the hour.

–hd *color*
This option specifies the color of the hands on an analog clock. The default is *black.*

–hl *color* This option specifies the color of the edges of the hands on an analog clock, and is only useful on color displays. The default is *black.*

–update *seconds*
This option specifies the frequency in seconds at which *xclock* should update its display. If the clock is obscured and then exposed, it will be updated immediately. A value of less than 30 seconds will enable a second hand on an analog clock. The default is 60 seconds.

–padding *number*
This option specifies the width in pixels of the padding between the window border and clock text or picture. The default is 10 on a digital clock and 8 on an analog clock.

The following standard X Toolkit command line arguments are commonly used with *xclock:*

–bg *color*
This option specifies the color to use for the background of the window. The default is *white.*

–bd *color*
This option specifies the color to use for the border of the window. The default is *black.*

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–fg *color* This option specifies the color to use for displaying text. The default is *black.*

–fn *font* This option specifies the font to be used for displaying normal text. The default is *6x10.*

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the prefered size and position of the clock window.

-display *host:display*
This option specifies the X server to contact.

-xrm *resourcestring*
This option specifies a resource string to be used.

X DEFAULTS

This program uses the *Clock* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

height (class **Height**)
Specifies the height of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

update (class **Interval**)
Specifies the frequency in seconds at which the time should be redisplayed.

foreground (class **Foreground**)
Specifies the color for the tic marks. The default is *black* since the core default for background is *white*.

hands (class **Foreground**)
Specifies the color of the insides of the clock's hands.

highlight (class **Foreground**)
Specifies the color used to highlight the clock's hands.

analog (class **Boolean**)
Specifies whether or not an analog clock should be used instead of a digital one. The default is True.

chime (class **Boolean**)
Specifies whether or not a bell should be rung on the hour and half hour.

padding (class **Margin**)
Specifies the amount of internal padding in pixels to be used. The default is 8.

font (class **Font**)
Specifies the font to be used for the digital clock. Note that variable width fonts currently will not always display correctly.

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background colors should be reversed.

SEE ALSO

X(1), xrdb(1), time(3C), Athena Clock widget

BUGS

Xclock believes the system clock.

When in digital mode, the string should be centered automatically.

When specifying a time offset, the grammar requires an hours field but if only minutes are given they will be quietly ignored. A negative offset of less than 1 hour is treated as a positive offset.

Digital clock windows default to the analog clock size.

Border color has to be explicitly specified when reverse video is used.

COPYRIGHT

AUTHORS

Tony Della Fera (MIT-Athena, DEC)
Dave Mankins (MIT-Athena, BBN)
Ed Moy (UC Berkeley)

NAME

xdpr – dump an X window directly to the printer

SYNOPSIS

xdpr [-option ...]

DESCRIPTION

Xdpr runs the commands *xwd*(1), *xpr*(1),*and lpr*(1) to dump an X window, process it for a laser printer, and print it out. This is the easiest way to get a printout of a window. *Xdpr* by default will print the largest possible representation of the window on the output page.

-P*printer*
This option specifies the name of the printer to be used.

–display *display*
This option specifies the X server to contact; see *X(1)*.

Any other arguments will be passed as arguments to the *xpr(1)* command.

SEE ALSO

X(1), xwd(1), xpr(1), xwud(1)

ENVIRONMENT

DISPLAY - for which display to use be default.

COPYRIGHT

AUTHOR

Michael R. Gretzinger, MIT Project Athena
Jim Gettys, MIT Project Athena
Extended for Ardent Printer Format by Mark Patrick

NAME

xdpyinfo – display information utility for X

SYNOPSIS

xdpyinfo [-display *displayname*]

DESCRIPTION

Xdpyinfo is a utility for displaying information about an X server. It is used to examine the capabilities of a server, the predefined values for various parameters used in communicating between clients and the server, and the different types of screens and visuals that are available.

EXAMPLE

The following shows a sample produced by *xdpyinfo* when connected to display that supports an 8 plane Pseudocolor screen as well as a 1 plane (monochrome) screen.

```
name of display:  empire:0.0
version number:  11.0
vendor string:  MIT X Consortium
vendor release number:  3
maximum request size:  16384 longwords (65536 bytes)
motion buffer size:  0
bitmap unit, bit order, padding:  32, MSBFirst, 32
image byte order:  MSBFirst
keycode range:  minimum 8, maximum 129
default screen number:  0
number of screens:  2

screen #0:
 dimensions:  1152x900 pixels (325x254 millimeters)
 resolution:  90x90 dots per inch
 root window id:  0x8006d
 depth of root window:  1 plane
 number of colormaps:  minimum 1, maximum 1
 default colormap:  0x80065
 default number of colormap cells:  2
 preallocated pixels:  black 1, white 0
 options:  backing-store YES, save-unders YES
 current input event mask:  0x1b8003c
  ButtonPressMask  ButtonReleaseMask  EnterWindowMask
  LeaveWindowMask  SubstructureNotifyMask  SubstructureRedirectMask
  FocusChangeMask  ColormapChangeMask  OwnerGrabButtonMask
 number of visuals:  1
 default visual id:  0x80064
 visual:
  visual id:  0x80064
  class:  StaticGray
  depth:  1 plane
  size of colormap:  2 entries
  red, green, blue masks:  0x0, 0x0, 0x0
  significant bits in color specification:  1 bits

screen #1:
 dimensions:  1152x900 pixels (325x254 millimeters)
 resolution:  90x90 dots per inch
 root window id:  0x80070
```

```
depth of root window:  8 planes
number of colormaps:  minimum 1, maximum 1
default colormap:  0x80067
default number of colormap cells:  256
preallocated pixels:  black 1, white 0
options:  backing-store YES, save-unders YES
current input event mask:  0x0
number of visuals:  1
default visual id:  0x80066
visual:
 visual id:  0x80066
 class:  PseudoColor
 depth:  8 planes
 size of colormap:  256 entries
 red, green, blue masks:  0x0, 0x0, 0x0
 significant bits in color specification:  8 bits
```

ENVIRONMENT

DISPLAY

To get the default host, display number, and screen.

SEE ALSO

X(1), xwininfo(1), xprop(1), xrdb(1)

BUGS

Due to a bug in the Xlib interface, there is currently no portable way to determine the depths of pixmap images that are supported by the server.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xedit – simple text editor for X

SYNTAX

xedit [*-toolkitoption* ...] [filename]

OPTIONS

Xedit accepts all of the standard X Toolkit command line options, plus:

filename Specifies the file that is to be loaded during start-up. If a file is not specified, *xedit* lets you load a file or create a new file after it has started up.

DESCRIPTION

Xedit provides a window consisting of the following three areas:

Commands Menu	Lists editing commands (for example, **Undo** or **Search**).
Message Window	Displays *xedit* messages. In addition, this window can be used as a scratch pad.
Edit Window	Displays the text of the file that you are editing or creating.

COMMANDS

Quit	Quits the current editing session. If any changes have not been saved, *xedit* displays a warning message and allows you to save the file.
Save	Stores a copy of the original, unedited file in *file*.BAK. Then, overwrites the original file with the edited contents.
Edit	Allows the text displayed in the Edit window to be edited.
Load	Loads the specified file and displays it in the Edit window.
Undo	Undoes the last edit only.
More	Undoes each edit previous to the last edit, which must first be undone with the **Undo** command.
Jump	Advances the cursor from the beginning of the file to the text line that corresponds to the selected line number.
<<	Searches from the cursor back to the beginning of the file for the string entered in the Search input box. If you do not enter a string in the Search input box, *xedit* automatically copies the last string that you selected from any X application into the Search input box and searches for that string.
Search >>	Searches from the cursor forward to the end of the file for the string entered in the search input box. If you do not enter a string in the Search input box, *xedit* automatically copies the last string that you selected from any X application into the Search input box and searches for that string.
Replace	Replaces the last searched-for string with the string specified in the Replace input box. If no string has been previously searched for, searches from the insert cursor to the end of the file for the next occurrence of the search

string and highlights it.

All — Repositions the cursor at the beginning of the file and replaces all occurrences of the search string with the string specified in the Replace input box.

X DEFAULTS

For *xedit*, the available class identifiers are:

ButtonBox
Command
Scrollbar
Text

For *xedit*, the available name identifiers are:

All
Edit
EditWindow
Jump
Load
MessageWindow
More
Quit
Replace
Save
Undo
xedit

For *xedit*, the available resources are:

EnableBackups — Specifies that, when edits made to an existing file are saved, *xedit* is to copy the original version of that file to *file*.BAK before it saves the changes. If the value of this option is specified as off, a backup file is not created.

background — Specifies the background color to be displayed in command buttons. The default is white.

border — Specifies the border color of the *xedit* window.

borderWidth — Specifies the border width, in pixels, of the *xedit* window.

font — Specifies the font displayed in the *xedit* window.

foreground — Specifies the foreground color of the *xedit* window. The default is black.

geometry — Specifies the geometry (window size and screen location) to be used as the default for the *xedit* window. For information about the format of the geometry specification, see ARGUMENTS.

internalHeight — Specifies the internal horizontal padding (spacing between text and button border) for command buttons.

internalWidth — Specifies the internal vertical padding (spacing between text and button border) for command buttons.

KEY BINDINGS

Each specification included in the *.XtActions* file modifies a key setting for the editor that *xedit* uses. When defining key specifications, you must use the following resource specification:

text.EventBindings: .XtActions

Each key specification assigns an editor command to a named key and/or mouse combination and has the format:

key: *function*

key Specifies the key or mouse button that is used to invoke the named function.

function Specifies the function to be invoked when the named key is pressed.

For more information about specifications in the **.XtActions** file, see

FILES

~/.XtActions
/usr/lib/X11/.XtActions

SEE ALSO

X(1), xrdb(1)

RESTRICTIONS

Large numbers of certain edit functions (for example, Undo or More) tend to degrade performance over time. If there is a noticeable decrease in response time, save and reload the file.

COPYRIGHT

NAME

xev – print contents of X events

SYNOPSIS

xev [–display *displayname*] [–geometry *geom*]

DESCRIPTION

Xev creates a window and then asks the X server to send it notices called *events* whenever anything happens to the window (such as being moved, resized, typed in, clicked in, etc.). It is useful for seeing what causes events to occur and to display the information that they contain.

OPTIONS

–display *display*
This option specifies the X server to contact.

–geometry *geom*
This option specifies the size and/or location of the window.

SEE ALSO

X(1), xwininfo(1), xdpyinfo(1), Xlib Programmers Manual, X Protocol Specification

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xfd – font displayer for X

SYNOPSIS

xfd [-options ...] -fn *fontname*

OPTIONS

-display *display*
Specifies the display to use.

-geometry *geometry*
Specifies an initial window geometry.

-bw *number*
Allows you to specify the width of the window border in pixels.

-rv The foreground and background colors will be switched. The default colors are black on white.

-fw Overrides a previous choice of reverse video. The foreground and background colors will not be switched.

-fg *color* On color displays, determines the foreground color (the color of the text).

-bg *color*
On color displays, determines the background color.

-bd *color*
On color displays, determines the color of the border.

-bf *fontname*
Specifies the font to be used for the messages at the bottom of the window.

-tl *title* Specifies that the title of the displayed window should be *title.*

-in *iconname*
Specifies that the name of the icon should be *iconname.*

-icon *filename*
Specifies that the bitmap in file *filename* should be used for the icon.

-verbose
Specifies that extra information about the font should be displayed.

-gray Specifies that a gray background should be used.

-start *charnum*
Specifies that character number *charnum* should be the first character displayed.

DESCRIPTION

Xfd creates a window in which the characters in the named font are displayed. The characters are shown in increasing order from left to right, top to bottom. The first character displayed at the top left will be character number 0 unless the -start option has been supplied in which case the character with the number given in the -start option will be used.

The characters are displayed in a grid of boxes, each large enough to hold any single character in the font. If the -gray option has been supplied, the characters will be displayed using XDrawImageString using the foreground and background colors on a gray background. This permits determining exactly how XDrawImageString will draw any given character. If -gray has not been supplied, the characters will simply be drawn using the foreground color on the background color.

All the characters in the font may not fit in the window at once. To see additional characters, click the right mouse button on the window. This will cause the next window full of characters to be displayed. Clicking the left mouse button on the window will cause the previous window full of characters to be displayed. *Xfd* will beep if an attempt is made to go back past the 0th character.

Note that if the font is a 8 bit font, the characters 256-511 (100-1ff in hexidecimal), 512-767 (200-2ff in hexidecimal), ... will display exactly the same as the characters 0-255 (00-ff in hexidecimal). *Xfd* by default creates a window big enough to display 16 rows of 16 columns (totally 256 characters).

Clicking the middle button on a character will cause that character's number to be displayed in both decimal and hexidecimal at the bottom of the window. If verbose mode is selected, additional information about that particular character will be displayed as well. The displayed information includes the width of the character, its left bearing, right bearing, ascent, and its descent. If verbose mode is selected, typing '<' or '>' into the window will display the minimum or maximum values respectively taken on by each of these fields over the entire font.

The font name is interpreted by the X server. To obtain a list of all the fonts available, use *xlsfonts(1)*.

The window stays around until the xfd process is killed or one of 'q', 'Q', ' ', or ctrl-c is typed into the *Xfd* window.

X DEFAULTS

Xfd uses the following X resources:

BorderWidth	Set the border width of the window.
BorderColor	Set the border color of the window.
ReverseVideo	If "on", reverse the definition of foreground and background color.
Foreground	Set the foreground color.
Background	Set the background color.
BodyFont	Set the font to be used in the body of the window. (I.e., for messages, etc.) This is not the font that *Xfd* displays, just the font it uses to display information about the font being displayed.
IconName	Set the name of the icon.
IconBitmap	Set the file we should look in to get the bitmap for the icon.
Title	Set the title to be used.

SEE ALSO

X(1), xlsfonts(1), xrdb(1)

BUGS

It should display the name of the font somewhere.

Character information displayed in verbose mode is sometimes clipped to the window boundary, hiding it from view.

It should be rewritten to use the X toolkit.

It should skip over pages full of non-existent characters.

COPYRIGHT

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xfed – font editor for X Version 11 bdf fontfiles

SYNOPSIS

xfed [–options ...] *filename*

xfed lets you interactively edit existing bdf-fontfiles. When run, *xfed* opens a window showing a magnified character of the font. There are four buttons to interact with *xfed*, two stepping buttons to advance to the previous/next character in the font, a write-file button and an exit button. If you leave *xfed* without saving a modified font, a dialog window will appear, asking if you want save changes before quitting. The write option moves the original fontfile

OPTIONS

xfed accepts the following options:

–display *host:dpy*
the server to be used. See *X(1)* for details.

–geometry *geometry*
the placement and size of the bitmap window on the screen. See *X(1)* for details.

–psize *number*
the pixel magnification factor.

–nogrid no grid in the edit window (not very useful).

–bw *number*
border width in pixels.

–fg *colorname*
foreground color.

–bg *colorname*
background color.

–bd *colorname*
border color. Note, that this color is also used for the grid.

–font *fontname*
font used for text in the font editor.

The window created by *xfed* has several subwindows. At the top, a window displays the name of the file being edited. Below this window, two similar windows display information about the font and the character being displayed. Below this window is a direct access window through which any character in the font can be accessed.

The direct access window has a vertical bar which represents the relative position of the displayed character in the font. The vertical bar can be "dragged" with any mouse button to access a character. Alternatively, you may type the desired character with the pointer in this window to access a character.

Below the direct access window is the edit window which displays the character with pixels magnified by the factor **psize**. The command buttons to step through the font, save the font, or exit the editor are to the right of the edit window. Two windows below the command buttons display the character in actual size using the foreground and background colors.

COMMANDS

Two arrows point into the edit window on the top and right sides. To change the bounding box for the current character they may be "dragged" with any mouse button.

Button 1 (usually the left button) turns on pixels.

Button 2 (usually th middle button) inverts pixels.

Button 3 (usually the right button) turns off pixels.

Four editing commands which are typed directly from the keyboard with the pointer within the bounding box of the characte are:r

i Insert a horizontal line at the pointer position

d Delete a horizontal line at the pointer position

I Insert a vertical line at the pointer position

D Delete a vertical line at the pointer position

The direct access window (above the edit window):

Any key pressed in this window will display that character in the edit window and move the vertical bar to the corresponding positon in the font.

Pushing a mouse button in this window will move the vertical bar to that position, and thus display the corresponding character in the edit window.

FILE FORMAT

see X-Doc 'Character Bitmap Distribution Format 2.1'

X DEFAULTS

xfed uses the routine *XGetDefault(3X)* to read defaults.

Geometry
The size and location of the bitmap window. The default depends on the font.

PixelSize
The initial pixel magnification factor. The default depends on the font.

NoGrid
No grid will be used in the edit window if true. The default is false.

BorderWidth
The border width. The default value is 2.

Foreground
The foreground color. Bits which are 1 in the bitmap are displayed in this color. This option is useful only on color displays. The default value is "black".

Background
The window's background color. Bits which are 0 in the bitmap are displayed in this color. This option is useful only on color displays. The

default value is "white".

BorderColor
The border color. This option is useful only on color displays. The default value is "black".

BodyFont
The text font. The default value is "fixed". If *xfed* is given variable width font it will not display the current character while the direct access bar is "dragged".

ENVIRONMENT

DISPLAY - the default host and display number.
XENVIRONMENT - the name of the defaults file to use.

SEE ALSO

X(1), BITMAP(1), X-Doc 'Character Bitmap Distribution Format 2.1'

BUGS

Limited edit features.

No "undo" command.

Unable to change font characteristics and character properties.

If you move the pointer too fast while holding a pointer button down, some squares may be 'missed'. This is caused by limitations in how frequently the X server can sample the mouse location.

There is no way to write to a file other than that specified on the command line.

COPYRIGHT

AUTHOR

Olaf Brandt
Network Cumputing Devices, Inc.
Palo Alto, CA

Copyright (c) 1988 by Siemens
Claus Gittinger
Software Consultant
Siemens Munich
Dep. D-St-Sp-4
Charles-de-Gaullestr. 2a
8000 Munich/Neuperlach
West Germany
Email: ..!decvax!unido!sinix!claus

NAME

xhost – server access control program for X

SYNOPSIS

xhost [[+-]hostname ...]

DESCRIPTION

The *xhost* program is used to add and delete hosts to the list of machines that are allowed to make connections to the X server. This provides a rudimentary form of privacy control and security. It is only sufficient for a workstation (single user) environment, although it does limit the worst abuses. Environments which require more sophisticated measures should use the hooks in the protocol for passing authentication data to the server.

The server initially allows network connections only from programs running on the same machine or from machines listed in the file */etc/X*.hosts* (where * is the display number of the server). The *xhost* program is usually run either from a startup file or interactively to give access to other users.

Hostnames that are followed by two colons (::) are used in checking DECnet connections; all other hostnames are used for TCP/IP connections.

OPTIONS

Xhost accepts the following command line options described below. For security, the options that effect access control may only be run from the same machine as the server.

[+]*hostname*
: The given *hostname* (the plus sign is optional) is added to the list of machines that are allowed to connect to the X server.

–*hostname*
: The given *hostname* is removed from the list of machines that are allowed to connect to the server. Existing connections are not broken, but new connection attempts will be denied. Note that the current machine is allowed to be removed; however, further connections (including attempts to add it back) will not be permitted. Resetting the server (thereby breaking all connections) is the only way to allow local connections again.

\+
: Access is granted to everyone, even if they aren't on the list of allowed hosts (i.e. access control is turned off).

–
: Access is restricted to only those machines on the list of allowed hosts (i.e. access control is turned on).

nothing
: If no command line arguments are given, the list of hosts that are allowed to connect is printed on the standard output along with a message indicating whether or not access control is currently enabled. This is the only option that may be used from machines other than the one on which the server is running.

FILES

/etc/X*.hosts

SEE ALSO

X(1), Xserver(1)

ENVIRONMENT

DISPLAY
: to get the default host and display to use.

BUGS

You can't specify a display on the command line because **-display** is a valid command line argument (indicating that you want to remove the machine named *"display"* from the access list).

COPYRIGHT

AUTHORS

Bob Scheifler, MIT Laboratory for Computer Science,
Jim Gettys, MIT Project Athena (DEC).

NAME

xinit – X Window System initializer

SYNOPSIS

xinit [-keepserver][[client] options] [-- [server] [display] options]

DESCRIPTION

The *xinit* program is used to start the X Window System server and a first client program (usually a terminal emulator) on systems that cannot start X directly from */etc/init* or in environments that use multiple window systems.

If xinit is used to run a program such as awm (the Ardent Window Manager) it is necessary to keep the server alive after the initial client has exited, this can be achieved through the -keepserver option. If the -keepserver argument is not specified then when this first client exits, *xinit* will kill the X server and then terminate.

If no specific client program is given on the command line, *xinit* will look for a file in the user's home directory called *.xinitrc* to run as a shell script to start up client programs. If no such file exists, *xinit* will use the following as a default:

```
xterm -geometry +1+1 -n login -display :0
```

If no specific server program is given on the command line, *xinit* will look for a file in the user's home directory called *.xserverrc* to run as a shell script to start up the server. If no such file exists, *xinit* will use the following as a default:

```
X :0
```

Note that this assumes that there is a program named X in the current search path. However, servers are usually named *Xdisplaytype* where *displaytype* is the type of graphics display which is driven by this server. The site administrator should, therefore, make a link to the appropriate type of server on the machine, or create a shell script that runs *xinit* with the appropriate server.

An important point is that programs which are run by *.xinitrc* and by *.xserverrc* should be run in the background if they do not exit right away, so that they don't prevent other programs from starting up. However, the last long-lived program started (usually a window manager or terminal emulator) should be left in the foreground so that the script won't exit (which indicates that the user is done and that *xinit* should exit).

An alternate client and/or server may be specified on the command line. The desired client program and its arguments should be given as the first command line arguments to *xinit*. To specify a particular server command line, append a double dash (--) to the *xinit* command line (after any client and arguments) followed by the desired server comand.

Both the client program name and the server program name must begin with a slash (/) or a period (.). Otherwise, they are treated as an arguments to be appended to their respective startup lines. This makes it possible to add arguments (for example, foreground and background colors) without having to retype the whole command line.

If an explicit server name is not given and the first argument following the double dash (--) is a colon followed by a digit, *xinit* will use that number as the display number instead of zero. All remaining arguments are appended to the server command line.

EXAMPLES

Below are several examples of how command line arguments in *xinit* are used.

xinit This will start up a server named *X* and run the user's *.xinitrc*, if it exists, or else start an *xterm*.

xinit -- /usr/bin/X11/Xqdss :1
This is how one could start a specific type of server on an alternate display.

xinit -geometry =80x65+10+10 -fn 8x13 -j -fg white -bg navy
This will start up a server named *X*, and will append the given arguments to the default *xterm* command. It will ignore *.xinitrc*.

xinit -e widgets -- ./Xsun -l -c
This will use the command *./Xsun -l -c* to start the server and will append the arguments *-e widgets* to the default *xterm* command.

xinit /usr/ucb/rsh fasthost cpupig -display ws:1 -- :1 -a 2 -t 5
This will start a server named *X* on display 1 with the arguments *-a 2 -t 5*. It will then start a remote shell on the machine **fasthost** in which it will run the command *cpupig*, telling it to display back on the local workstation.

Below is a sample *.xinitrc* that starts a clock, several terminals, and leaves the window manager running as the "last" application. Assuming that the window manager has been configured properly, the user then chooses the "Exit" menu item to shut down X.

```
xrdb -load $HOME/.Xres
xsetroot -solid gray &
xclock -g 50x50-0+0 -bw 0 &
xload -g 50x50-50+0 -bw 0 &
xterm -g 80x24+0+0 &
xterm -g 80x24+0-0 &
uwm
```

Sites that want to create a common startup environment could simply create a default *.xinitrc* that references a site-wide startup file:

```
#!/bin/sh
. /usr/local/lib/site.xinitrc
```

Another approach is to write a script that starts *xinit* with a specific shell script. Such scripts are usually named *x11*, *xstart*, or *startx* and are a convenient way to provide a simple interface for novice users:

```
#!/bin/sh
xinit /usr/local/bin/startx -- /usr/bin/X11/Xhp :1
```

ENVIRONMENT VARIABLES

DISPLAY
This variable gets set to the name of the display to which clients should connect.

XINITRC
This variable specifies an init file containing shell commands to start up the initial windows. By default, *.xinitrc* in the home directory will be used.

SEE ALSO

X(1), Xserver(1), xterm(1), xrdb(1)

COPYRIGHT

AUTHOR

Bob Scheifler, MIT Laboratory for Computer Science

The -keepserver option was added by Jordan Hubbard, Ardent Computer

NAME

xload – load average display for X

SYNOPSIS

xload [*-toolkitoption* ...] [-scale *integer*] [-update *seconds*]

DESCRIPTION

The *xload* program displays a periodically updating histogram of the system load average. This program is nothing more than a wrapper around the Athena Load widget.

OPTIONS

Xload accepts all of the standard X Toolkit command line options along with the following additional options:

–scale *integer*
This option specifies the minimum number of tick marks in the histogram, where one division represents one load average point. If the load goes above this number, *xload* will create more divisions, but it will never use fewer than this number. The default is 1.

–update *seconds*
This option specifies the frequency in seconds at which *xload* updates its display. If the load average window is uncovered (by moving windows with a window manager or by the *xrefresh* program), the graph will be also be updated. The minimum amount of time allowed between updates is 5 seconds (which is also the default).

–hl *color* This option specifies the color of the label and scale lines.

The following standard X Toolkit arguments are commonly used with *xload*:

–bd *color*
This option specifies the border color. The default is *black.*

–bg *color*
This option specifies the background color. The default is *white.*

–bw *pixels*
This option specifies the width in pixels of the border around the window. The default value is 2.

–fg *color* This option specifies the graph color. The default color is *black.*

–fn *fontname*
This option specifies the font to be used in displaying the name of the host whose load is being monitored. The default is *6x10.*

–rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–geometry *geometry*
This option specifies the prefered size and postion of the window.

–display *display*
This option specifies the X server to contact.

–xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Load* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the load average graph.

height (class **Height**)
Specifies the height of the load average graph.

update (class **Interval**)
Specifies the frequency in seconds at which the load should be redisplayed.

scale (class **Scale**)
Specifies the initial number of ticks on the graph. The default is 1.

minScale (class **Scale**)
Specifies the minimum number of ticks that will be displayed. The default is 1.

foreground (class **Foreground**)
Specifies the color for the graph. The default is *black* since the core default for background is *white*.

highlight (class **Foreground**)
Specifies the color for the text and scale lines. The default is the same as for the **foreground** resource.

label (class **Label**)
Specifies the label to use on the graph. The default is the hostname.

font (class **Font**)
Specifies the font to be used for the label. The default is *fixed*.

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background should be reversed.

SEE ALSO

X(1), xrdb(1), mem(4), Athena Load widget

DIAGNOSTICS

Unable to open display or create window. Unable to open /dev/kmem. Unable to query window for dimensions. Various X errors.

BUGS

This program requires the ability to open and read the special system file */dev/kmem*. Sites that do not allow general access to this file should make *xload* belong to the same group as */dev/kmem* and turn on the *set group id* permission flag.

Reading /dev/kmem is inherently non-portable. Therefore, the widget upon which this application is based must be ported to each new operating system.

Border color has to be explicitly specified when reverse video is used.

COPYRIGHT

AUTHORS

K. Shane Hartman (MIT-LCS) and Stuart A. Malone (MIT-LCS);
with features added by Jim Gettys (MIT-Athena), Bob Scheifler (MIT-LCS), and Tony Della Fera (MIT-Athena)

NAME

xlogo – X Window System logo

SYNOPSIS

xlogo [*-toolkitoption* ...]

DESCRIPTION

The *xlogo* program displays the X Window System logo. This program is nothing more than a wrapper around the Athena Logo widget.

OPTIONS

Xlogo accepts all of the standard X Toolkit command line options, of which the following are used most frequently:

-bg *color*
This option specifies the color to use for the background of the window. The default is *white*. A correct color for the background is something like *maroon*.

-bd *color*
This option specifies the color to use for the border of the window. The default is *black*.

-bw *number*
This option specifies the width in pixels of the border surrounding the window.

-fg *color* This option specifies the color to use for displaying the logo. The default is *black*. A correct color for the background is something like *silver*, which you can approximate with a shade of gray, like #aa9.

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the prefered size and position of the logo window.

-display *display*
This option specifies the X server to contact.

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Logo* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the logo. The default width is 100 pixels.

height (class **Height**)
Specifies the height of the logo. The default height is 100 pixels.

foreground (class **Foreground**)
Specifies the color for the logo. The default is *black* since the core default for background is *white*.

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background should be reversed.

ENVIRONMENT

DISPLAY
to get the default host and display number.

XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X(1), xrdb(1), Athena Logo widget

COPYRIGHT

AUTHORS

Ollie Jones of Apollo Computer wrote the logo graphics routine, based on a graphic design by Danny Chong and Ross Chapman of Apollo Computer.

NAME

xlsfonts – server font list displayer for X

SYNOPSIS

xlsfonts [-options ...] [-fn pattern]

DESCRIPTION

Xlsfonts lists the fonts that match the given *pattern.* The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

OPTIONS

–display *host:dpy*
This option specifies the X server to contact.

–l This option indicates that a long listing should be generated for each font.

–m This option indicates that long listings should also print the minimum and maximum bounds of each font.

–C This option indicates that listings should use multiple columns. This is the same as **-n 0.**

–1 This option indicates that listings should use a single column. This is the same as **-n 1.**

–w *width*
This option specifies the width in characters that should be used in figuring out how many columns to print. The default is 79.

–n *columns*
This option specifies the number of columns to use in displaying the output. By default, it will attempt to fit as many columns of font names into the number of character specified by **-w** *width.*

SEE ALSO

X(1), Xserver(1), xset(1), xfd(1)

ENVIRONMENT

DISPLAY
to get the default host and display to use.

BUGS

Doing "xlsfonts -l" can tie up your server for a very long time. This is really a bug with single-threaded non-preemptible servers, not with this program.

COPYRIGHT

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xlswins – server window list displayer for X

SYNOPSIS

xlswins [-options ...] [windowid ...]

DESCRIPTION

Xlswins lists the window tree. By default, the root window is used as the starting point, although a specific window may be specified using the *-id* option. If no specific windows are given on the command line, the root window will be used.

OPTIONS

–display *displayname*
This option specifies the X server to contact.

–l This option indicates that a long listing should be generated for each window. This includes a number indicating the depth, the geometry relative to the parent as well as the location relative to the root window.

–format *radix*
This option specifies the radix to use when printing out window ids. Allowable values are: *hex*, *octal*, and *decimal*. The default is hex.

–indent *number*
This option specifies the number of spaces that should be indented for each level in the window tree. The default is 2.

SEE ALSO

X(1), Xserver(1), xwininfo(1), xprop(1)

ENVIRONMENT

DISPLAY
to get the default host and display to use.

BUGS

This should be integrated with xwininfo somehow.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xmag – magnify parts of the screen

SYNOPSIS

xmag [-option ...]

DESCRIPTION

The *xmag* program allows you to magnify portions of the screen. If no explicit region is specified, a square centered around the pointer is displayed indicating the area to be enlarged. Once a region has been selected, a window is popped up showing a blown up version of the region in which each pixel in the source image is represented by a small square of the same color. Pressing Button1 on the pointer in the enlargement window pops up a small window displaying the position, number, and RGB value of the pixel under the pointer until the button is released. Pressing the space bar or any other pointer button removes the enlarged image so that another region may be selected. Pressing "q", "Q", or "^C" in the enlargement window exits the program.

OPTIONS

–display *display*
This option specifies the X server to use for both reading the screen and displaying the enlarged version of the image.

–geometry *geom*
This option specifies the size and/or location of the enlargement window. By default, the size is computed from the size of the source region and the desired magnification. Therefore, only one of **–source** *size* and **–mag** *magfactor* options may be specified if a window size is given with this option.

–source *geom*
This option specifies the size and/or location of the source region on the screen. By default, a 64x64 square centered about the pointer is provided for the user to select an area of the screen. The size of the source is used with the desired magnification to compute the default enlargement window size. Therefore, only one of **–geometry** *size* and **–mag** *magfactor* options may be specified if a source size is given with this option.

–mag *magfactor*
This option specifies an integral factor by which the source region should be enlarged. The default magnification is 5. This is used with the size of the source to compute the default enlargement window size. Therefore, only one of **-geometry** *size* and **–source** *geom* options may be specified if a magnification factor is given with this option.

–bw *pixels*
This option specifies the width in pixels of the border surrounding the enlargement window.

–bd *color*
This option specifies the color to use for the border surrounding the enlargement window.

–bg *colororpixelvalue*
This option specifies the name of the color to be used as the background of the enlargement window. If the name begins with a percent size (%), it is interpretted to be an absolute pixel value. This is useful when displaying large areas since pixels that are the same color as the background do not need to be painted in the enlargement. The default is to use the BlackPixel of the screen.

-fn *fontname*
This option specifies the name of a font to use when displaying pixel values (used when Button1 is pressed in the enlargement window).

-z This option indicates that the server should be grabbed during the dynamics and the call to XGetImage. This is useful for ensuring that clients don't change their state as a result of entering or leaving them with the pointer.

X DEFAULTS

The *xmag* program uses the following X resources:

geometry (class **Geometry**)
Specifies the size and/or location of the enlargement window.

source (class **Source**)
Specifies the size and/or location of the source region on the screen.

magnification (class **Magnification**)
Specifies the enlargement factor.

borderWidth (class **BorderWidth**)
Specifies the border width in pixels.

borderColor (class **BorderColor**)
Specifies the color of the border.

background (class **Background**)
Specifies the color or pixel value to be used for the background of the enlargement window.

font (class **Font**)
Specifies the name of the font to use when displaying pixel values when the user presses Button1 in the enlargement window.

SEE ALSO

X(1), xwd(1)

BUGS

This program will behave strangely on displays that support windows of different depths.

Because the window size equals the source size times the magnification, you only need to specify two of the three parameters. This can be confusing.

Being able to drag the pointer around and see a dynamic display would be very nice.

Another possible interface would be for the user to drag out the desired area to be enlarged.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xman – X window system manual page display program.

SYNOPSIS

xman [-options ...]

DESCRIPTION

Xman opens a small window on the display that contains the a menu of commands. My hope is that this is small enough to allow you to call it in your .login and leave it running throughout your entire login session. At this point in the program there several options; *Help* will pop up a window with on line help, *Quit* will exit, and *Manual Page* will pop up a window with a manual page browser in it, you may pop up more than one manual page browser window from a single execution of xman.

For further information on using xman please read the online help information. The rest of this manual page will dicsuss customization of xman to suit the needs of a particular user. For groups that desire to create a new directory listing (manual section), read the section on the *mandesc* file that allows system wide customization of manual sections.

Xman creates temporary files for all unformatted man pages and all apropos searches. These files are stored in /tmp.

OPTIONS

Xman is build upon the XToolkit (Xtk) and as such understands all default command line options of the Xtk.

MANDESC

Xman will search each directory in the MANPATH for files to add to the directory listing. xman also looks for a file called *mandesc* in each directory specified in the manpath. The mandesc file tells xman to create a separate section for one or more of the directories in the manpath. For example, if your manpath were '/usr/man:/usr/sipb/man' then xman would search /usr/man and finding no mandesc file would put all of its files in the default section names (e.g. manl gets a section name of local). But it does find a mandesc file while searching /usr/sipb/man, this file contains the line 'lSIPB Programs' (no quote marks) this tells xman to put all files in the 'manl' section in a new directoy called 'SIPB Programs'. Xman will search this file until there are no more lines of information. This flexabilty is *ideal for courses* that have their own manual pages. Like 'man' xman searches only the following directories: man0, man1, man2, man3... man8 ,manl (small L), mann. Although it usually ignores the information in man0 unless there is a *mandesc* that specifically tells it not to.

WIDGET AND RESOURCE NAMES

In order to change the default values for the widget resources you need to have the names, thus, below I have specified the names of some of the most common widgets.

Top Menu - topBox.
Help Window - help
Manual Page Display Window - manualBrowser.
Manual Page Command Popup Menu - xmanCommands.
Manual Page Section Popup Menu - xmanSections.
Manual Page Search Popup - xmanSearch.

In addition Xman has the following application resources:

manualFontNormal
manualFontBold
manualFontItalic
The fonts to use for the three types of text in the

manual pages.

directoryFontNormal
The font to use for the directory text.

bothShown
Either 'true' or 'false', specifiecs wether or not you want both the directory and the manual page shown at start up.

directoryHeight
The number of pixels hight the directory is when both it and the manual page are shown simultaniously.

topCursor
helpCursor
manpageCursor
searchEntryCursor
The cursors to use in the top box, help window, manual page window, and search entry text widget respectivily.

helpFile
Use this rather than the system default helpfile.

topBox
Either 'true' or 'false', determines wether the top box (containing the help, quit and manual page buttons) or a manual page is put on the screen at start-up.

FILES

/usr/man/* or those specified in the MANPATH.
mandesc

SEE ALSO

X(1), X(8C), man(1), apropos(1)

ENVIRONMENT

DISPLAY
to get the default host and display to use.

XENVIRONMENT
to get the name of the defaults file to load in.

MANPATH
to get the search path for manual pages. Directories are separated by colons (e.g. /usr/man:/mit/kit/man:/foo/bar/man).

BUGS

The last button pressed in any of the popup menus stays highlighted, this is caused by toolkit problems with grabs and window leave events.

AUTHORS

Chris Peterson, MIT Project Athena from the V10 version written by Barry Shein of Boston University.

NAME

xmh – X window interface to the mh Mail Handler

SYNOPSIS

xmh [-path *mailpath*] [-initial *foldername*] [-flag] [*-toolkitoption* ...]

DESCRIPTION

The *xmh* program provides a window-oriented front end to the mh Mail Handler. It is designed to take advantage of a large graphical display and pointer; it will not function on an ordinary terminal at all.

Xmh consists of user-interface code only. To actually do things with your mail, it makes calls to the *mh* package.

Please don't be misled by the size of this document. *Xmh* really is easy to use!

INSTALLATION

The current version of *xmh* requires that the user is already set up to use *mh*, version 6. To do so, see if there is a file called .mh_profile in your home directory. If you do, check to see if it contains a line that starts with "Current-Folder". If it does, then you've been using version 4 or earlier of *mh*; to convert to version 6, you must remove that line. (Failure to do so causes spurious output to stderr, which can hang *xmh* depending on your setup.)

If you do not already have a .mh_profile, you can create one (and everything else you need) by typing "inc" to the shell.

For more information, refer to the *mh* documentation.

RUNNING XMH

Run *xmh* as you would any other X application (e.g., xterm). It will accept a command-line display (of the form "-display host:dpy"); the default display is specified in the environment variable DISPLAY.

The rest of this document will probably be rather hard to follow without actually running *xmh* and seeing the things being described.

BASIC SCREEN LAYOUT

Xmh starts out with a single screen. There will be 6 or 7 areas on the screen:

- A list of your folders. (New users of mh will see only "inbox" here.)
- A list of the global and folder-oriented commands.
- A list of the messages in one of your folders (initially, this will show the messages in "inbox").
- A list of the message-oriented commands.
- A view of one of your messages. (Initially this is blank.)
- A list of commands for the message being viewed.

And, there will possibly be:

- A list of message-sequences defined for this folder. This appears just below the list of messages in this folder. (Message-sequences are discussed below; if you don't know what they are, then you won't have any.)

XMH AND THE TOOLKIT

Xmh uses the X Toolkit. Many of the features described below (scrollbars, buttonboxes, etc.) are actually part of the Toolkit, and are described here only for completeness. For more information, see the Toolkit documentation.

SCROLLBARS

Some parts of the screen will have a vertical area on the left containing a grey bar. This area is a *scrollbar*. They are used whenever the data in a window takes up more space than can be displayed. The grey bar indicates what portion of your data is visible. Thus, if the entire length of the area is grey, then you are looking at all your data. If only the first half is grey, then you are looking at the top half of your data.

You can use the pointer in the scrollbar to change what part of the data is visible. If you click with the middle button, then the top of the grey area will move to where the pointer is, and the corresponding portion of data will be displayed. If you hold down the middle button, you can drag around the grey area. This makes it easy to get to the top of the data: just press with the middle, drag off the top of the scrollbar, and release.

If you click with button 1, then the data to the right of the pointer will scroll to the top of the window. If you click with pointer button 3, then the data at the top of the window will scroll down to where the pointer is.

BUTTONBOXES

Any area consisting of many words or short phrases, each enclosed in a box, is called a *buttonbox*. Each box is actually a button that you can press by moving the pointer onto it and pressing pointer button 1. If a given buttonbox has more buttons in it than can fit, it will be displayed with a scrollbar, so you can always scroll to the button you want.

ADJUSTING THE RELATIVE SIZES OF AREAS

If you're not satisfied with the size of the various areas on the screen, they can easily be changed. Near the right edge of the border between each region is a black box, called a *grip*. Simply point to that grip with the pointer, press a pointer button, drag up or down, and release. Exactly what happens depends on which pointer button you press.

If you drag with the middle button, then only that border will move. This mode is simplest to understand, but is probably the least useful.

If you drag with pointer button 1, then you are adjusting the size of the window above. *Xmh* will attempt to compensate by adjusting some window below it.

If you drag with pointer button 3, then you are adjusting the size of the window below. *Xmh* will attempt to compensate by adjusting some window above it.

All windows have a mininum and maximum size; you will never be allowed to move a border past the point where it would make a window have an invalid size.

SELECTED FOLDER

The selected folder is whichever foldername is hilighted in the top buttonbox. Note that this is not necessarily the same folder that is being viewed. To change the

selected folder, just press on the desired folder button.

GENERAL COMMANDS AND FOLDER COMMANDS

The second buttonbox contains commands of a global nature:

Quit XMH
Exits *xmh*, after first checking that you won't lose any changes.

Compose Message
Composes a new message. A new window will be brought up; for a description of it, see COMPOSITION WINDOWS, below.

Open Folder
Display the data in the selected folder. Thus, the selected folder also becomes the viewed folder.

Open Folder in New Window
Creates a new screen, and displays the selected folder in that screen. Note, however, that you may not display the same folder in more than one screen at a time.

Create Folder
Create a new folder. You will be prompted for a name for the new folder; to enter the name, point the pointer at the blank box provided and type. Hit the Confirm button when finished, or hit Abort to cancel this operation.

Delete Folder
Destroy the selected folder. You will be asked to confirm this action (see CONFIRMATION WINDOWS).

HIGHLIGHTED MESSAGES, SELECTED MESSAGES AND THE CURRENT MESSAGE

It is possible to highlight a set of messages in the list of messages for the viewed folder. To highlight a message, just click on it with pointer button 1. To highlight a range of messages, click on the first one with pointer button 1 and on the last one with pointer button 3.

The selected messages are the same as the highlighted messages, if any. If no messages are highlighted, then the selected messages are considered the same as the current message.

The current message is indicated by a '+' next to the message number. It usually corresponds to the message currently being viewed.

MESSAGE COMMANDS

The third buttonbox (fourth if you have message-sequences displayed) contains commands to deal with messages:

Incorporate New Mail
Add any new mail received to your inbox folder, and set the current message to be the first new message. (This button is selectable only if "inbox" is the folder being viewed.)

View Next Message
View the first selected message. If no messages are highlighted, view the current message. If current message is already being viewed, view the first

unmarked message after the current message.

View Previous Message
View the last selected message. If no messages are highlighted, view the current message. If current message is already being viewed, view the first unmarked message before the current message.

Mark Deleted
Mark the selected messages for deletion. If no messages are highlighted, then this will automatically display the next unmarked message.

Mark Move
Mark the selected messages to be moved into the current folder. (If the current folder is the same as the viewed folder, this command will just beep.) If no messages are highlighted, then this will automatically display the next unmarked message.

Mark Copy
Mark the selected messages to be copied into the current folder. (If the current folder is the same as the viewed folder, this command will just beep.)

Unmark
Remove any of the above three marks from the selected messages.

View in New Window
Create a new window containing only a view of the first selected message.

Reply Create a composition window in reply to the first selected message.

Forward
Create a composition window whose body is initialized to be the contents of the selected messages.

Use as Composition
Create a composition window whose body is initialized to be this message. Note that any changes you make in the composition will also be saved in this message. This function is meant to be used with the "drafts" folder (see COMPOSITION WINDOWS).

Commit Changes
Execute any deletions, moves, and copies that have been marked in this folder.

Print Print the selected messages. *Xmh* normally prints by invoking the *enscript*(1) command, but you may change the command it uses. (See CUSTOMIZING, below).

Pack folder
Renumber the messages in this folder so they start with 1 and increment by 1.

Sort folder
Sort the messages in this folder in chronological order. As a side effect, this also packs the folder.

Force Rescan
Rebuild the list of messages. This can be used whenever you suspect *xmh*'s idea of what messages you have is wrong. (In particular, this is useful if you ever change things using straight mh commands without using *xmh*.)

Pick Messages
Define a new message-sequence. (See MESSAGE-SEQUENCES.)

The following buttons will appear but will be sensitive only if the current folder has any message-sequences defined (See MESSAGE-SEQUENCES).

Open Sequence
Change the viewed sequence to be the same as the selected sequence.

Add to Sequence
Add the selected messages to the selected sequence.

Remove from Sequence
Remove the selected messages from the selected sequence.

Delete Sequence
Remove the selected sequence entirely. Note the messages themselves are not affected; they simply are no longer grouped together as a message-sequence.

VIEW WINDOWS

The commands in these windows are the same as the message commands by the same name, except instead of affecting the selected messages, they affect the viewed message. In addition there is the "Edit View" button, which allows you to edit the message being viewed. While editing, the "Edit View" button will change to a "Save View" button, which should be pressed to save your edits.

COMPOSITION WINDOWS

Aside from the normal text editing functions, there are three command buttons associated with composition windows:

Abort Comp
Abort this composition window. If changes have been made, you will be asked to confirm losing them.

Send Send this composition. If any errors appear in the message header, you will receive a mail message containing this composition and a description of the error.

Save Save this composition in your drafts folder. (If you do not have a folder named "drafts", one will be created.) Then you can safely close the composition. At some future date, you can continue working on the composition by opening your drafts folder, selecting the message, and using the "Use as Composition" command.

TEXT EDITING COMMANDS

All of the text editing commands are actually defined by the Text widget in the X Toolkit. The commands may be bound to different keys than the defaults described below through the standard X Toolkit key re-binding mechanisms. See the X Toolkit and Athena Widgets documentation for more details.

Whenever you are asked to enter any text, you will be using a standard text editing interface. Various control and meta keystroke combinations are bound to a somewhat Emacs-like set of commands. In addition, the pointer buttons may be used to select a portion of text or to move the insertion point in the text. Pressing pointer button 1 causes the insertion point to move to the pointer. Double-clicking button 1 selects a word, triple-clicking selects a paragraph, and quadruple-clicking selects everything. Any selection may be extended in either direction by using pointer

button 3.

In the following, a *line* refers to one displayed row of characters in the window. A *paragraph* refers to the text between carriage returns. Text within a paragraph is broken into lines based on the current width of the window.

The following keystroke combinations are defined:

Control-A
Move to the beginning of the current line.

Control-B, Control-H, Backspace
Move backward one character.

Control-D
Delete the next character.

Control-E
Move to the end of the current line.

Control-F
Move forward one character.

Control-J, LineFeed
Create a new paragraph with the same indentation as the previous one.

Control-K
Kill the rest of this line.

Control-L
Repaint this window.

Control-M, Return
New paragraph.

Control-N
Move down to the next line.

Control-O
Break this paragraph into two.

Control-P
Move up to the previous line.

Control-V
Move down to the next screenfull of text.

Control-W
Kill the selected text.

Control-Y
Insert the last killed text.

Control-Z
Scroll the text one line up.

Meta-< Move to the beginning of the document.

Meta-> Move to the end of the document.

Meta-[Move backward one paragraph.

Meta-] Move forward one paragraph.

Meta-B Move backward one word.

Meta-D Kill the next word.

Meta-F Move forward one word.

Meta-H, Meta-Delete
Kill the previous word.

Meta-I Insert a file. If any text is selected, use the selected text as the filename. Otherwise, a box will appear in which you can type the desired filename.

Meta-V Move up to the previous screenfull of text.

Meta-Y Stuff the last selected text here. Note that this can be text selected in some other text subwindow. Also, if you select some text in an xterm window, it may be inserted in an *xmh* window with this command. Pressing pointer button 2 is equivalent to this.

Meta-Z Scroll the text one line down.

Delete Delete the previous character.

CONFIRMATION WINDOWS

Whenever you press a button that may cause you to lose some work or is otherwise dangerous, a window will appear asking you to confirm the action. This window will contain an "Abort" button and a "Confirm" button. Pressing the "Abort" button cancels the operation, and pressing the "Confirm" will proceed with the operation. (A very handy shortcut exists: if you press the offending button again, it will be interpreted as a "Confirm". If you press any other command button, it will be interpreted as an "Abort".)

MESSAGE-SEQUENCES

A mh message sequence is just a set of messages associated with some name. They are local to a particular folder; two different folders can have sequences with the same name. In all folders, the sequence "all" is predefined; it consists of the set of all messages in that folder. (The sequence "cur" is also usually defined for every folder; it consists of only the current message. *Xmh* hides "cur" from the user, instead placing a "+" by the current message. Also, *xmh* does not support the "unseen" sequence, so that one is also hidden from the user.)

The message sequences for a folder are displayed as buttons containing the names of the sequences (including one for "all"). The table of contents (aka "toc") is at any one time displaying one message sequence. This is called the "viewed sequence"; if it's not "all", its name will be displayed in the title bar just after the folder name. Also, at any time one of the sequence buttons will be highlighted. This is called the "selected sequence". Note that the viewed sequence and the selected sequence are not necessarily the same. (This all pretty much corresponds to the way the folder buttons work.)

The **Open Sequence**, **Add to Sequence**, **Remove from Sequence**, and **Delete Sequence** buttons are active only if the viewed folder contains message-sequences.

Note that none of the above actually effect whether a message is in the folder. Remember that a sequence is a set of messages within the folder; the above operations just affect what messages are in that set.

To create a new sequence, press the "Pick" button. A new window will appear, with lots of places to enter text. Basically, you can describe the sequence's initial set of messages based on characteristics of the message. Thus, you can define a sequence to

be all the messages that were from a particular person, or with a particular subject, and so on. You can also connect things up with boolean operators, so you can select all things from "weissman" with the subject "xmh".

Hopefully, the layout is fairly obvious. The simplest cases are the easiest: just point to the proper field and type. If you enter in more than one field, it will only select messages which match all non-empty fields.

The more complicated cases arise when you want things that match one field or another one, but not necessarily both. That's what all the "or" buttons are for. If you want all things with the subject "xmh" or "xterm", just press the "or" button next to the "Subject:" field. Another box will appear where you can enter another subject.

If you want all things either from "weissman" or with subject "xmh", but not necessarily both, select the "-Or-" button. This will essentially double the size of the form. You can then enter "weissman" in a from: box on the top half, and "xmh" in a subject: box on the lower part.

If you ever select the "Skip" button, then only those messages that *don't* match the fields on that row are included.

Finally, in the bottom part of the window will appear several more boxes. One is the name of the sequence you're defining. (It defaults to the name of the selected sequence when "Pick" was pressed, or to "temp" if "all" was the selected sequence.) Another box defines which sequence to look through for potential members of this sequence; it defaults to the viewed sequence when "Pick" was pressed.

Two more boxes define a date range; only messages within that date range will be considered. These dates must be entered in 822-style format: each date is of the form "dd mmm yy hh:mm:ss zzz", where dd is a one or two digit day of the month, mmm is the three-letter abbreviation for a month, and yy is a year. The remaining fields are optional: hh, mm, and ss specify a time of day, and zzz selects a time zone. Note that if the time is left out, it defaults to midnight; thus if you select a range of "7 nov 86" - "8 nov 86", you will only get messages from the 7th, as all messages on the 8th will have arrived after midnight.

"Date field" specifies which date field in the header to look at for this date range; it probably won't be useful to anyone. If the sequence you're defining already exists, you can optionally merge the old set with the new; that's what the "Yes" and "No" buttons are all about. Finally, you can "OK" the whole thing, or "Cancel" it.

In general, most people will rarely use these features. However, it's nice to occasionally use "Pick" to find some messages, look through them, and then hit "Delete Sequence" to put things back in their original state.

CUSTOMIZING XMH

As with all standard X applications, *xmh* may be customized through entries in the resource manager. The following resource manager entries are defined: [Note: the entry names must be entered in either all lower-case, or in the exact case shown below.]

BackGround

Background color. Currently, this will effect only buttons. (Default is white.)

ButtonFont
What font to use for button names. (Default is "timrom10".)

CheckNewMail
If True, *xmh* will check at regular intervals to see if new mail has arrived for any of the folders. A visual indication will be given if new mail is waiting to be retrieved. (Default is True.)

CompButtonLines
How many rows of buttons to display under a composition. (Default is 1.)

CompFont
What font to use when composing a message. (Default is "6x13".)

CompGeometry
Initial geometry for windows containing compositions.

CompLines
How many lines of a composition to display. (Default is 20.)

ConfirmFont
What font to use for confirmation windows. (Default is "timrom10b".)

FolderButtonLines
How many rows of folder command buttons to display. (Default is 1.)

FolderLines
How many rows of foldername buttons to display. (Default is 1.)

ForeGround
Foreground color. Currently, this will effect only title bars and buttons. (Default is black.)

Geometry
Default geometry to use. (Default is none.)

HideBoringHeaders
If "on", then *xmh* will attempt to skip uninteresting header lines within messages by scrolling them off. (Default is "on".)

InitialFolder
Which folder to display on startup. May also be set with the command-line option **-initial**. (Default is "inbox".)

InitialIncFile
The file name of your incoming mail drop. *xmh* tries to construct a filename for the "inc -file" command, but in some installations (e.g. those using the Post Office Protocol) no file is appropriate. In this case, **InitialIncFile** should be specified as the empty string, and *inc* will be invoked without a -file argument.

LabelFont
What font to use for the title bars. (Default is "timrom10i".)

MailPath
The full path prefix for locating your mail folders. May also be set with the command-line option, **-path**. (Default is the "Path" component in $HOME/.mh_profile, or "$HOME/Mail" if none.)

MailWaitingFlag
If True, *xmh* will attempt to set an indication in it's icon when new mail is waiting to be retrieved. If this option is True, then CheckNewMail is assumed to be True as well. The **-flag** command line option is a quick way to turn MailWaitingFlag on.

MhPath What directory in which to find the mh commands. If a command isn't found here, then the directories in the user's path are searched. (Default is "/usr/local/mh6".)

PickGeometry
Initial geometry for pick windows.

PickEntryFont
What font to use for user text fields in pick windows. (Default is "timrom10".)

PickTextFont
What font to use for static text fields in pick windows. (Default is "timrom10".)

PrintCommand
What sh command to execute to print a message. Note that stdout and stderr must be specifically redirected! If a message or range of messages is selected for printing, the full file paths of each message file is appended to the specified print command. (Default is "enscript >/dev/null 2>/dev/null").

TempDir
Directory for *xmh* to store temporary directories. For privacy, a user might want to change this to a private directory. (Default is "/tmp".)

TocButtonLines
How many rows of message command buttons to display. (Default is 1.)

TocFont
What font to use for a folder's table of contents. (Default is "6x13".)

TocGeometry
Initial geometry for master *xmh* windows.

TocLines
How messages to display in a folder's table of contents. (Default is 10.)

TocWidth
How many characters to generate for each message in a folder's table of contents. (Default is 100. Use 80 if you plan to use *mhe* a lot.)

ViewButtonLines
How many rows of buttons to display under a view of a message. (Default is 1.)

ViewFont
What font to use for a view of a message. (Default is "6x13".)

ViewGeometry
Initial geometry for windows showing only a view of a message.

ViewLines
How many lines of a message to display. (Default is 20.)

If TocGeometry, ViewGeometry, CompGeometry, or PickGeometry are not specified, then the value of Geometry is used instead. If the resulting height is not specified (e.g., "", "=500", "+0-0"), then the default height is calculated from the fonts and line counts specified above. If the width is not specified (e.g., "", "=x300", "-0+0), then half of the display width is used. If unspecified, the height of a pick window defaults to half the height of the display.

Any of these options may also be specified on the command line by using the

standard X Toolkit resource specification mechanism. Thus, to run *xmh* showing all message headers,

% xmh -xrm '*HideBoringHeaders:off'

FILES

~/Mail

~/.mh_profile

SEE ALSO

X(1), xrdb(1), X Toolkit, mh(1) - the mh Mail Handler

BUGS

Printing support is minimal.

Keyboard shortcuts for commands would be nice.

Should handle the "unseen" message-sequence.

Should determine by itself if the user hasn't used *mh* before, and offer to set things up for him or her.

Still a few commands missing (rename folder, remail message).

Needs sub-folder support.

COPYRIGHT

AUTHOR

Terry Weissman, Digital Western Research Laboratory

NAME

xmodmap – utility for modifying keymaps in X

SYNOPSIS

xmodmap [-options ...] [filename]

DESCRIPTION

The *xmodmap* program is used to edit and display the keyboard *modifier map* and *keymap table* that are used by client applications to convert event keycodes into keysyms. It is usually run from the user's session startup script to configure the keyboard according to personal tastes.

OPTIONS

The following options may be used with *xmodmap*:

–display *display*
This option specifies the host and display to use.

–help This option indicates that a brief description of the command line arguments should be printed on the standard error. This will be done whenever an unhandled argument is given to *xmodmap*.

–grammar
This option indicates that a help message describing the expression grammar used in files and with -e expressions should be printed on the standard error.

–verbose
This option indicates that *xmodmap* should print logging information as it parses its input.

–quiet This option turns off the verbose logging. This is the default.

–n This option indicates that *xmodmap* should not change the mappings, but should display what it would do, like *make(1)* does when given this option.

–e *expression*
This option specifies an expression to be executed. Any number of expressions may be specified from the command line.

–pm This option indicates that the current modifier map should be printed on the standard output.

–pk This option indicates that the current keymap table should be printed on the standard output.

–pp This option indicates that the current pointer map should be printed on the standard output.

– A lone dash means that the standard input should be used as the input file.

The *filename* specifies a file containing *xmodmap* expressions to be executed. This file is usually kept in the user's home directory with a name like *.xmodmaprc*.

EXPRESSION GRAMMAR

The *xmodmap* program reads a list of expressions and parses them all before attempting execute any of them. This makes it possible to refer to keysyms that are being redefined in a natural way without having to worry as much about name conflicts.

keycode *NUMBER* = *KEYSYMNAME ...*
The list of keysyms is assigned to the indicated keycode (which may be specified in decimal, hex or octal and can be determined by running the *xev* program in the examples directory). Usually only one keysym is assigned to a given code.

keysym *KEYSYMNAME* = *KEYSYMNAME* ...
The *KEYSYMNAME* on the left hand side is looked up to find its current keycode and the line is replaced with the appropriate **keycode** expression. Note that if you have the same keysym bound to multiple keys, this might not work.

clear *MODIFIERNAME*
This removes all entries in the modifier map for the given modifier, where valid name are: Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4 and Mod5 (case does not matter in modifier names, although it does matter for all other names). For example, "clear Lock" will remove all any keys that were bound to the shift lock modifier.

add *MODIFIERNAME* = *KEYSYMNAME* ...
This adds the given keysyms to the indicated modifier map. The keysym names are evaluated after all input expressions are read to make it easy to write expressions to swap keys (see the EXAMPLES section).

remove *MODIFIERNAME* = *KEYSYMNAME* ...
This removes the given keysyms from the indicated modifier map. Unlike **add**, the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether or not they have been reassigned.

pointer = default
This sets the pointer map back to its default settings (button 1 generates a code of 1, button 2 generates a 2, etc.).

pointer = *NUMBER* ...
This sets to pointer map to contain the indicated button codes. The list always starts with the first physical button.

Lines that begin with an exclamation point (!) are taken as comments.

If you want to change the binding of a modifier key, you must also remove it from the appropriate modifier map.

EXAMPLES

Many pointers are designed such the first button is pressed using the index finger of the right hand. People who are left-handed frequently find that it is more comfortable to reverse the button codes that get generated so that the primary button is pressed using the index finger of the left hand. This could be done on a 3 button pointer as follows:

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

Many editor applications support the notion of Meta keys (similar to Control keys except that Meta is held down instead of Control). However, some servers do not have a Meta keysym in the default keymap table, so one needs to be added by hand. The following command will attach Meta to the Multi-language key (sometimes label Compose Character). It also takes advantage of the fact that applications that need a Meta key simply need to get the keycode and don't require the keysym to be in the first column of the keymap table. This means that applications that are looking for a Multi_key (including the default modifier map) won't notice any change.

```
% keysym Multi_key = Multi_key Meta_L
```

One of the more simple, yet convenient, uses of *xmodmap* is to set the keyboard's "rubout" key to generate an alternate keysym. This frequently involves exchanging Backspace with Delete to be more comfortable to the user. If the *ttyModes* resource in

xterm is set as well, all terminal emulator windows will use the same key for erasing characters:

```
% xmodmap -e "keysym BackSpace = Delete"
% echo "XTerm*ttyModes: erase ^?" | xrdb -merge
```

Some keyboards do not automatically generate less than and greater than characters when the comma and period keys are shifted. This can be remedied with *xmodmap* by resetting the bindings for the comma and period with the following scripts:

```
!
! make shift-, be < and shift-. be >
!
keysym comma = comma less
keysym period = period greater
```

One of the more irritating differences between keyboards is the location of the Control and Shift Lock keys. A common use of *xmodmap* is to swap these two keys as follows:

```
!
! Swap Caps_Lock and Control_L
!
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

The *keycode* command is useful for assigning the same keysym to multiple keycodes. Although unportable, it also makes it possible to write scripts that can reset the keyboard to a known state. The following script sets the backspace key to generate Delete (as shown above), flushes all existing caps lock bindings, makes the CapsLock key be a control key, make F5 generate Escape, and makes Break/Reset be a shift lock.

```
!
! On the HP, the following keycodes have key caps as listed:
!
!   101  Backspace
!    55  Caps
!    14  Ctrl
!    15  Break/Reset
!    86  Stop
!    89  F5
!
keycode 101 = Delete
keycode 55 = Control_R
clear Lock
add Control = Control_R
keycode 89 = Escape
keycode 15 = Caps_Lock
add Lock = Caps_Lock
```

ENVIRONMENT

DISPLAY
to get default host and display number.

SEE ALSO

X(1)

BUGS

Every time a **keycode** expression is evaluated, the server generates a *MappingNotify* event on every client. This can cause some thrashing. All of the changes should be batched together and done at once. Clients that receive keyboard input and ignore *MappingNotify* events will not notice any changes made to keyboard mappings.

Xmodmap should generate "add" and "remove" expressions automatically whenever a keycode that is already bound to a modifier is changed.

There should be a way to have the *remove* expression accept keycodes as well as keysyms for those times when you really mess up your mappings.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium, rewritten from an original by David Rosenthal of Sun Microsystems.

NAME

xmore - File browsing program for the X Window System.

SYNOPSIS

xmore [-options ...] filename

DESCRIPTION

Xmore pops up a window on the display specified, containing the file specified on the command line. This file may easily be viewed using the scrollbar to the left of the window.

For further information on using xmore please read the online help information. The rest of this manual page will dicsuss customization of xmore to suit the needs of a particular user.

OPTIONS

Xmore is build upon the *XToolkit (Xtk)* and as such understands all default command line options of the Xtk.

WIDGET AND RESOURCE NAMES

In order to change the default values for the widget resources you need to have the names, the name of the help widget is 'help'. In addition to the standard widget resources Xmore has the following application resources:

textFontNormal
textFontItalic
The fonts to use for the two types of text.

topCursor
helpCursor
The cursors to use in the main window and the help window, repectively.

helpFile
Use this rather than the system default helpfile.

SEE ALSO

X(1), X(8C), more(1)

BUGS

The probably are some.

AUTHOR

Chris Peterson, MIT Project Athena, from the V10 version written by Barry Shein of Boston University.

NAME

xpr – print an X window dump

SYNOPSIS

xpr [**–scale** *scale*] [**–height** *inches*] [**–width** *inches*] [**–left** *inches*] [**–top** *inches*] [**–header** *string*] [**–trailer** *string*] [**–landscape**] [**–portrait**] [**–rv**] [**–compact**] [**–output** *filename*] [**–append** *filename*] [**–noff**] [**–split** *n*] [**–device** *dev*] [*filename*]

DESCRIPTION

Xpr takes as input a window dump file produced by *xwd(1)* and formats it for output on the LN03, LA100, PostScript printers, IBM PP3812 page printer or Ardent Printer Format. If no file argument is given, the standard input is used. By default, *xpr* prints the largest possible representation of the window on the output page. Options allow the user to add headers and trailers, specify margins, adjust the scale and orientation, and append multiple window dumps to a single output file. Output is to standard output unless **–output** is specified.

Command Options

–scale *scale*
Affects the size of the window on the page. The LN03 and PostScript printers are able to translate each bit in a window pixel map into a grid of a specified size. For example each bit might translate into a 3x3 grid. This would be specified by **–scale** 3. By default a window is printed with the largest scale that will fit onto the page for the specified orientation.

–height *inches*
Specifies the maximum height of the window on the page.

–width *inches*
Specifies the maximum width of the window.

–left *inches*
Specifies the left margin in inches. Fractions are allowed. By default the window is centered in the page.

–top *inches*
Specifies the top margin for the picture in inches. Fractions are allowed.

–header *header*
Specifies a header string to be printed above the window.

–trailer *trailer*
Specifies a trailer string to be printed below the window.

–landscape
Forces the window to printed in landscape mode. By default a window is printed such that its longest side follows the long side of the paper.

–portrait
Forces the window to be printed in portrait mode. By default a window is printed such that its longest side follows the long side of the paper.

–rv Forces the window to be printed in reverse video.

–compact
Uses simple run-length encoding for compact representation of windows with lots of white pixels.

-output *filename*
Specifies an output file name. If this option is not specified, standard output is used.

-append *filename*
Specifies a filename previously produced by *xpr* to which the window is to be appended.

-noff
When specified in conjunction with **-append**, the window will appear on the same page as the previous window.

-split *n*
This option allows the user to split a window onto several pages. This might be necessary for very large windows that would otherwise cause the printer to overload and print the page in an obscure manner.

-device *device*
Specifies the device on which the file will be printed. Currently only the LN03 (-device ln03), LA100 (-device la100), PostScript printers (-device ps), IBM PP3812 (-device pp) and Ardent Printer Format (-device apf) are supported. **-device lw** (LaserWriter) is equivalent to -device ps and is provided only for backwards compatibility.

SEE ALSO

xwd(1), xdpr(1), xwud(1), X(1)

LIMITATIONS

The current version of *xpr* can generally print out on the LN03 most X windows that are not larger than two-thirds of the screen. For example, it will be able to print out a large Emacs window, but it will usually fail when trying to print out the entire screen. The LN03 has memory limitations that can cause it to incorrectly print very large or complex windows. The two most common errors encountered are "band too complex" and "page memory exceeded." In the first case, a window may have a particular six pixel row that contains too many changes (from black to white to black). This will cause the printer to drop part of the line and possibly parts of the rest of the page. The printer will flash the number '1' on its front panel when this problem occurs. A possible solution to this problem is to increase the scale of the picture, or to split the picture onto two or more pages. The second problem, "page memory exceeded," will occur if the picture contains too much black, or if the picture contains complex half-tones such as the background color of a display. When this problem occurs the printer will automatically split the picture into two or more pages. It may flash the number '5' on its from panel. There is no easy solution to this problem. It will probably be necessary to either cut and paste, or rework to application to produce a less complex picture.

Xpr provides some support for the LA100. However, there are several limitations on its use: The picture will always be printed in portrait mode, there is no scaling and the aspect ratio will be slightly off.

Support for PostScript output currently cannot handle the **-append**, **-noff** or **-split** options.

The **-compact** option is *only* supported for PostScript output. It compresses white space but not black space, so it is not useful for reverse-video windows.

COPYRIGHT

See *X(1)* for a full statement of rights and permissions.

AUTHORS

Michael R. Gretzinger, MIT Project Athena, Jose Capo, MIT Project Athena (PP3812 support), Marvin Solomon (University of Wisconsin). Mark Patrick (Ardent Printer Format support) Ardent Computer.

NAME

xmodmap - utility for modifying keymaps in X

SYNOPSIS

xmodmap [-options ...] [filename]

DESCRIPTION

The *xmodmap* program is used to edit and display the keyboard *modifier map* and *keymap table* that are used by client applications to convert event keycodes into keysyms. It is usually run from the user's session startup script to configure the keyboard according to personal tastes.

OPTIONS

The following options may be used with *xmodmap*:

-display *display*
This option specifies the host and display to use.

-help This option indicates that a brief description of the command line arguments should be printed on the standard error. This will be done whenever an unhandled argument is given to *xmodmap*.

-grammar
This option indicates that a help message describing the expression grammar used in files and with -e expressions should be printed on the standard error.

-verbose
This option indicates that *xmodmap* should print logging information as it parses its input.

-quiet This option turns off the verbose logging. This is the default.

-n This option indicates that *xmodmap* should not change the mappings, but should display what it would do, like *make(1)* does when given this option.

-e *expression*
This option specifies an expression to be executed. Any number of expressions may be specified from the command line.

-pm This option indicates that the current modifier map should be printed on the standard output.

-pk This option indicates that the current keymap table should be printed on the standard output.

-pp This option indicates that the current pointer map should be printed on the standard output.

- A lone dash means that the standard input should be used as the input file.

The *filename* specifies a file containing *xmodmap* expressions to be executed. This file is usually kept in the user's home directory with a name like *.xmodmaprc*.

EXPRESSION GRAMMAR

The *xmodmap* program reads a list of expressions and parses them all before attempting execute any of them. This makes it possible to refer to keysyms that are being redefined in a natural way without having to worry as much about name conflicts.

keycode *NUMBER* = *KEYSYMNAME ...*
The list of keysyms is assigned to the indicated keycode (which may be specified in decimal, hex or octal and can be determined by running the *xev* program in the examples directory). Usually only one keysym is assigned to a given code.

keysym *KEYSYMNAME* = *KEYSYMNAME* ...
The *KEYSYMNAME* on the left hand side is looked up to find its current keycode and the line is replaced with the appropriate **keycode** expression. Note that if you have the same keysym bound to multiple keys, this might not work.

clear *MODIFIERNAME*
This removes all entries in the modifier map for the given modifier, where valid name are: Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4 and Mod5 (case does not matter in modifier names, although it does matter for all other names). For example, "clear Lock" will remove all any keys that were bound to the shift lock modifier.

add *MODIFIERNAME* = *KEYSYMNAME* ...
This adds the given keysyms to the indicated modifier map. The keysym names are evaluated after all input expressions are read to make it easy to write expressions to swap keys (see the EXAMPLES section).

remove *MODIFIERNAME* = *KEYSYMNAME* ...
This removes the given keysyms from the indicated modifier map. Unlike **add,** the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether or not they have been reassigned.

pointer = default
This sets the pointer map back to its default settings (button 1 generates a code of 1, button 2 generates a 2, etc.).

pointer = *NUMBER* ...
This sets to pointer map to contain the indicated button codes. The list always starts with the first physical button.

Lines that begin with an exclamation point (!) are taken as comments.

If you want to change the binding of a modifier key, you must also remove it from the appropriate modifier map.

EXAMPLES

Many pointers are designed such the first button is pressed using the index finger of the right hand. People who are left-handed frequently find that it is more comfortable to reverse the button codes that get generated so that the primary button is pressed using the index finger of the left hand. This could be done on a 3 button pointer as follows:

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

Many editor applications support the notion of Meta keys (similar to Control keys except that Meta is held down instead of Control). However, some servers do not have a Meta keysym in the default keymap table, so one needs to be added by hand. The following command will attach Meta to the Multi-language key (sometimes label Compose Character). It also takes advantage of the fact that applications that need a Meta key simply need to get the keycode and don't require the keysym to be in the first column of the keymap table. This means that applications that are looking for a Multi_key (including the default modifier map) won't notice any change.

```
% keysym Multi_key = Multi_key Meta_L
```

One of the more simple, yet convenient, uses of *xmodmap* is to set the keyboard's "rubout" key to generate an alternate keysym. This frequently involves exchanging Backspace with Delete to be more comfortable to the user. If the *ttyModes* resource in

xterm is set as well, all terminal emulator windows will use the same key for erasing characters:

```
% xmodmap -e "keysym BackSpace = Delete"
% echo "XTerm*ttyModes: erase ^?" | xrdb -merge
```

Some keyboards do not automatically generate less than and greater than characters when the comma and period keys are shifted. This can be remedied with *xmodmap* by resetting the bindings for the comma and period with the following scripts:

```
!
! make shift-, be < and shift-. be >
!
keysym comma = comma less
keysym period = period greater
```

One of the more irritating differences between keyboards is the location of the Control and Shift Lock keys. A common use of *xmodmap* is to swap these two keys as follows:

```
!
! Swap Caps_Lock and Control_L
!
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

The *keycode* command is useful for assigning the same keysym to multiple keycodes. Although unportable, it also makes it possible to write scripts that can reset the keyboard to a known state. The following script sets the backspace key to generate Delete (as shown above), flushes all existing caps lock bindings, makes the CapsLock key be a control key, make F5 generate Escape, and makes Break/Reset be a shift lock.

```
!
! On the HP, the following keycodes have key caps as listed:
!
!   101 Backspace
!    55 Caps
!    14 Ctrl
!    15 Break/Reset
!    86 Stop
!    89 F5
!
keycode 101 = Delete
keycode 55 = Control_R
clear Lock
add Control = Control_R
keycode 89 = Escape
keycode 15 = Caps_Lock
add Lock = Caps_Lock
```

ENVIRONMENT

DISPLAY
to get default host and display number.

SEE ALSO

X(1)

BUGS

Every time a **keycode** expression is evaluated, the server generates a *MappingNotify* event on every client. This can cause some thrashing. All of the changes should be batched together and done at once. Clients that receive keyboard input and ignore *MappingNotify* events will not notice any changes made to keyboard mappings.

Xmodmap should generate "add" and "remove" expressions automatically whenever a keycode that is already bound to a modifier is changed.

There should be a way to have the *remove* expression accept keycodes as well as keysyms for those times when you really mess up your mappings.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium, rewritten from an original by David Rosenthal of Sun Microsystems.

NAME

xrdb – X server resource database utility

SYNOPSIS

xrdb [-option ...] [*filename*]

DESCRIPTION

Xrdb is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0. You would normally run this program from your X startup file.

The resource manager (used by the Xlib routine *XGetDefault(3X)* and the X Toolkit) uses the RESOURCE_MANAGER property to get user preferences about color, fonts, and so on for applications. Having this information in the server (where it is available to all clients) instead of on disk, solves the problem in previous versions of X that required you to maintain *defaults* files on every machine that you might use. It also allows for dynamic changing of defaults without editting files.

For compatibility, if there is no RESOURCE_MANAGER property defined (either because xrdb was not run or if the property was removed), the resource manager will look for a file called *.Xdefaults* in your home directory.

The *filename* (or the standard input if - or no input file is given) is optionally passed through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

HOST=hostname
: the hostname portion of the display to which you are connected.

WIDTH=num
: the width of the screen in pixels.

HEIGHT=num
: the height of the screen in pixels.

X_RESOLUTION=num
: the x resolution of the screen in pixels per meter.

Y_RESOLUTION=num
: the y resolution of the screen in pixels per meter.

PLANES=num
: the number of bit planes for the default visual.

BITS_PER_RGB=num
: the number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it is not related to the number of planes, which the log base 2 of the size of the colormap.

CLASS=visualclass
: one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor.

COLOR
: only defined if the default visual's type is one of the color options.

Lines that begin with an exclamation mark (!) are ignored and may be used as comments.

OPTIONS

xrdb program accepts the following options:

-help This option (or any unsupported option) will cause a brief description of the allowable options and parameters to be printed.

-display *display*
This option specifies the X server to be used; see *X(1)*.

-cpp *filename*
This option specifies the pathname of the C preprocessor program to be used. Although *xrdb* was designed to use CPP, any program that acts as a filter and accepts the -D, -I, and -U options may be used.

-nocpp This option indicates that *xrdb* should not run the input file through a preprocessor before loading it into the RESOURCE_MANAGER property.

-symbols
This option indicates that the symbols that are defined for the preprocessor should be printed onto the standard output. It can be used in conjunction with **-query,** but not with the options that change the RESOURCE_MANAGER property.

-query This option indicates that the current contents of the RESOURCE_MANAGER property should be printed onto the standard output. Note that since preprocessor commands in the input resource file are part of the input file, not part of the property, they won't appear in the output from this option. The **-edit** option can be used to merge the contents of the property back into the input resource file without damaging preprocessor commands.

-load This option indicates that the input should be loaded as the new value of the RESOURCE_MANAGER property, replacing whatever what there (i.e. the old contents are removed). This is the default action.

-merge This option indicates that the input should be merged with, instead of replacing, the current contents of the RESOURCE_MANAGER property. Since *xrdb* can read the standard input, this option can be used to the change the contents of the RESOURCE_MANAGER property directly from a terminal or from a shell script.

-remove
This option indicates that the RESOURCE_MANAGER property should be removed from its window.

-edit *filename*
This option indicates that the contents of the RESOURCE_MANAGER property should be edited into the given file, replacing any values already listed there. This allows you to put changes that you have made to your defaults back into your resource file, preserving any comments or preprocessor lines.

-backup *string*
This option specifies a suffix to be appended to the filename used with **-edit** to generate a backup file.

-D*name[=value]*
This option is passed through to the preprocessor and is used to define symbols for use with conditionals such as *#ifdef*.

-U*name* This option is passed through to the preprocessor and is used to remove any definitions of this symbol.

–I*directory*
This option is passed through to the preprocessor and is used to specify a directory to search for files that are referenced with *#include*.

FILES

Generalizes ~/*.Xdefaults* files.

SEE ALSO

X(1), XGetDefault(3X), Xlib Resource Manager documentation

ENVIRONMENT

DISPLAY
to figure out which display to use.

BUGS

The default for no arguments should be to query, not to overwrite, so that it is consistent with other programs.

COPYRIGHT

AUTHORS

Phil Karlton, rewritten from the original by Jim Gettys

NAME

xrefresh – refresh all or part of an X screen

SYNOPSIS

xrefresh [-option ...]

DESCRIPTION

Xrefresh is a simple X program that causes all or part of your screen to be repainted. This is useful when system messages have messed up your screen. *Xrefresh* maps a window on top of the desired area of the screen and then immediately unmaps it, causing refresh events to be sent to all applications. By default, a window with no background is used, causing all applications to repaint "smoothly." However, the various options can be used to indicate that a solid background (of any color) or the root window background should be used instead.

ARGUMENTS

–white Use a white background. The screen just appears to flash quickly, and then repaint.

–black Use a black background (in effect, turning off all of the electron guns to the tube). This can be somewhat disorienting as everything goes black for a moment.

–solid *color*
Use a solid background of the specified color. Try green.

–root Use the root window background.

–none This is the default. All of the windows simply repaint.

–geometry *WxH+X+Y*
Specifies the portion of the screen to be repainted; see *X(1)*.

–display *display*
This argument allows you to specify the server and screen to refresh; see *X(1)*.

X DEFAULTS

The *xrefresh* program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

Black, White, Solid, None, Root
Determines what sort of window background to use.

Geometry
Determines the area to refresh. Not very useful.

ENVIRONMENT

DISPLAY - To get default host and display number.

SEE ALSO

X(1)

BUGS

It should have just one default type for the background.

COPYRIGHT

AUTHORS

Jim Gettys, Digital Equipment Corp., MIT Project Athena

NAME

xset – user preference utility for X

SYNOPSIS

xset [-display *display*] [-b] [b on/off] [b [*volume* [*pitch* [*duration*]]] [-c] [c on/off] [c [*volume*]] [[-+]fp[-+=] *path*[,*path*[,...]]] [fp default] [fp rehash] [[-]led [*integer*]] [led on/off] [m[ouse] [*acceleration* [*threshold*]]] [m[ouse] default] [p *pixel color*] [[-]r] [r on/off] [ssp *program_name*] [-ssp] [s [*length* [*period*]]] [s blank/noblank] [s expose/noexpose] [s on/off] [s default] [q]

DESCRIPTION

This program is used to set various user preference options of the display.

OPTIONS

–display *display*
This option specifies the server to use; see *X(1)*.

b the **b** option controls bell volume, pitch and duration. This option accepts up to three numerical parameters, a preceding dash(-), or a 'on/off' flag. If no parameters are given, or the 'on' flag is used, the system defaults will be used. If the dash or 'off' are given, the bell will be turned off. If only one numerical parameter is given, the bell volume will be set to that value, as a percentage of its maximum. Likewise, the second numerical parameter specifies the bell pitch, in hertz, and the third numerical parameter specifies the duration in milliseconds. Note that not all hardware can vary the bell characteristics. The X server will set the characteristics of the bell as closely as it can to the user's specifications.

c The **c** option controls key click. This option can take an optional value, a preceding dash(-), or an 'on/off' flag. If no parameter or the 'on' flag is given, the system defaults will be used. If the dash or 'off' flag is used, key-click will be disabled. If a value from 0 to 100 is given, it is used to indicate volume, as a percentage of the maximum. The X server will set the volume to the nearest value that the hardware can support.

fp= *path,...*
The **fp=** sets the font path to the directories given in the path argument. The directories are interpreted by the server, not by the client, and are server-dependent. Directories that do not contain font databases created by *mkfontdir* will be ignored by the server.

fp *default*
The *default* argument causes the font path to be reset to the server's default.

fp *rehash*
The *rehash* argument causes the server to reread the font databases in the current font path. This is generally only used when adding new fonts to a font directory (after running *mkfontdir* to recreate the font database).

–fp or **fp–**
The **–fp** and **fp–** options remove elements from the current font path. They must be followed by a comma-separated list of directories.

+fp or **fp+**
This **+fp** and **fp+** options prepend and append elements to the current font path, respectively. They must be followed by a comma-separated list of directories.

led The **led** option controls the keyboard LEDs. This controls the turning on or off of one or all of the LEDs. It accepts an optional integer, a preceding dash(-) or an 'on/off' flag. If no parameter or the 'on' flag is given, all LEDs

are turned on. If a preceding dash or the flag 'off' is given, all LEDs are turned off. If a value between 1 and 32 is given, that LED will be turned on or off depending on the existance of a preceding dash. A common LED which can be controlled is the "Caps Lock" LED. "xset led 3" would turn led #3 on. "xset -led 3" would turn it off. The particular LED values may refer to different LEDs on different hardware.

m The **m** option controls the mouse parameters. The parameters for the mouse are 'acceleration' and 'threshold'. The mouse, or whatever pointer the machine is connected to, will go 'acceleration' times as fast when it travels more than 'threshold' pixels in a short time. This way, the mouse can be used for precise alignment when it is moved slowly, yet it can be set to travel across the screen in a flick of the wrist when desired. One or both parameters for the **m** option can be omitted, but if only one is given, it will be interpreted as the acceleration. If no parameters or the flag 'default' is used, the system defaults will be set.

p The **p** option controls pixel color values. The parameters are the color map entry number in decimal, and a color specification. The root background colors may be changed on some servers by altering the entries for BlackPixel and WhitePixel. Although these are often 0 and 1, they need not be. Also, a server may choose to allocate those colors privately, in which case an error will be generated. The map entry must not be a read-only color, or an error will result.

r The **r** option controls the autorepeat. If a preceding dash or the 'off' flag is used, autorepeat will be disabled. If no parameters or the 'on' flag is used, autorepeat will be enabled.

ssp The **ssp** option allows you to set the screen saver program used by the server. The currently available options are *ardentlogo* which moves the Ardent logo around the screen and *stringart* which draws some nice vector patterns. You can also define your own

-ssp The **-ssp** option disables any previously set screen saver program. The server then either displays a blank screen or the X logo depending upon the options provided to the X server.

s The **s** option lets you set the screen saver parameters. This option accepts up to two numerical parameters, a 'blank/noblank' flag, an 'expose/noexpose' flag, an 'on/off' flag, or the 'default' flag. If no parameters or the 'default' flag is used, the system will be set to its default screen saver characteristics. The 'on/off' flags simply turn the screen saver functions on or off. The 'blank' flag sets the preference to blank the video (if the hardware can do so) rather than display a background pattern, while 'noblank' sets the preference to display a pattern rather than blank the video. The 'expose' flag sets the preference to allow window exposures (the server can freely discard window contents), while 'noexpose' sets the preference to disable screen saver unless the server can regenerate the screens without causing exposure events. The length and period parameters for the screen saver function determines how long the server must be inactive for screen saving to activate, and the period to change the background pattern to avoid burn in. The arguments are specified in seconds. If only one numerical parameter is given, it will be used for the length.

q The **q** option gives you information on the current settings.

These settings will be reset to default values when you log out.

Note that not all X implementations are guaranteed to honor all of these options.

SEE ALSO

X(1), Xserver(1), xmodmap(1), xrdb(1), xsetroot(1)

COPYRIGHT

AUTHOR

Bob Scheifler, MIT Laboratory for Computer Science
David Krikorian, MIT Project Athena (X11 version)
Mark Patrick, Ardent Computer (screen saver program added)

NAME

xsetroot – root window parameter setting utility for X

SYNOPSIS

xsetroot [-help] [-def] [-display *display*] [-cursor *cursorfile maskfile*] [-bitmap *filename*] [-mod *x y*] [-gray] [-grey] [-fg *color*] [-bg *color*] [-rv] [-solid *color*] [-name *string*]

DESCRIPTION

The *setroot* program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with *xsetroot* until you find a personalized look that you like, then put the *xsetroot* command that produces it into your X startup file. If no options are specified, or if *-def* is specified, the window is reset to its default state. The *-def* option can be specified along with other options and only the non-specified characteristics will be reset to the default state.

Only one of the background color/tiling changing options (-solid, -gray, -grey, -bitmap, and -mod) may be specified at a time.

OPTIONS

The various options are as follows:

-help
Print a usage message and exit.

-def Reset unspecified attributes to the default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)

-cursor *cursorfile maskfile*
This lets you change the pointer cursor to whatever you want when the pointer cursor is outside of any window. Cursor and mask files are bitmaps (little pictures), and can be made with the *bitmap(1)* program. You probably want the mask file to be all black until you get used to the way masks work.

-bitmap *filename*
Use the bitmap specified in the file to set the window pattern. You can make your own bitmap files (little pictures) using the *bitmap(1)* program. The entire background will be made up of repeated "tiles" of the bitmap.

-mod *x y*
This is used if you want a plaid-like grid pattern on your screen. x and y are integers ranging from 1 to 16. Try the different combinations. Zero and negative numbers are taken as 1.

-gray
Make the entire background gray. (Easier on the eyes.)

-grey
Make the entire background grey.

-fg *color*
Use "color" as the foreground color. Foreground and background colors are meaningful only in combination with -cursor, -bitmap, or -mod.

-bg *color*
Use "color" as the background color.

-rv This exchanges the foreground and background colors. Normally the foreground color is black and the background color is white.

-solid *color*
Set the window color to "color".

-name *string*
Set the name of the root window to "string". There is no default value. Usually a name is assigned to a window so that the window manager can use a text representation when the window is iconified. This option is unused since you can't iconify the background.

-display *display*
Specifies the server to connect to; see *X(1)*.

SEE ALSO

X(1), xset(1), xrdb(1)

COPYRIGHT

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xstart – Start X Window System

SYNOPSIS

xstart

DESCRIPTION

The *xstart* program is used to start the X Window System server, the Ardent Window Manager (awm) and execute the users $HOME/.xdesktop file. The .xdesktop file typically makes calls to the xset(1) program to modify the servers behaviour, downloads the user's prefrences using xrdb(1) and starts up any initial clients. This command is intended to be run from the console and will terminate with an error if it is not entered from there.

The menu option "Exit X Window's" from the "Window Op's" menu terminates the X server and clients returning the user to the console.

SEE ALSO

X(1), Xserver(1).
The Ardent End User Interface Guide.

COPYRIGHT

NAME

xterm – terminal emulator for X

SYNOPSIS

xterm [*-toolkitoption* ...] [-option ...]

DESCRIPTION

The *xterm* program is a terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), *xterm* will use the facilities to notify programs running in the window whenever it is resized.

The VT102 and Tektronix 4014 terminals each have their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (height/width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the "active" window for receiving keyboard input and terminal output. This is the window that contains the text cursor and whose border highlights whenever the pointer is in either window. The active window can be choosen through escape sequences, the "Modes" menu in the VT102 window, and the "Tektronix" menu in the 4014 window.

OPTIONS

The *xterm* terminal emulator accepts all of the standard X Toolkit command line options along with the additional options listed below (if the option begins with a '+' instead of a '–', the option is restored to its default value):

–help This causes *xterm* to print out a verbose message describing its options.

–132 Normally, the VT102 DECCOLM escape sequence that switches between 80 and 132 column mode is ignored. This option causes the DECCOLM escape sequence to be recognized, and the *xterm* window will resize appropriately.

–ah This option indicates that *xterm* should always highlight the text cursor and borders. By default, *xterm* will display a hollow text cursor whenever the focus is lost or the pointer leaves the window.

+ah This option indicates that *xterm* should do text cursor highlighting.

–b *number*
This option specifies the size of the inner border (the distance between the outer edge of the characters and the window border) in pixels. The default is 2.

–cc *characterclassrange:value*[,...]
This sets classes indicated by the given ranges for using in selecting by words. See the section specifying character classes.

–cr *color* This option specifies the color to use for text cursor. The default is to use the same foreground color that is used for text.

–cu This option indicates that *xterm* should work around a bug in the *curses*(3x) cursor motion package that causes the *more*(1) program to display lines that are exactly the width of the window and are followed by a line beginning with a tab to be displayed incorrectly (the leading tabs are not displayed).

+cu This option indicates that that *xterm* should not work around the *curses*(3x) bug mentioned above.

–e *program [arguments ...]*
This option specifies the program (and its command line arguments) to be run in the *xterm* window. It also sets the window title and icon name to be the basename of the program being executed if neither *-T* nor *-n* are given on the command line. **This must be the last option on the command line.**

–fb *font* This option specifies a font to be used when displaying bold text. This font must be the same height and width as the normal font. If only one of the normal or bold fonts is specified, it will be used as the normal font and the bold font will be produced by overstriking this font. The default bold font is "vtbold."

–j This option indicates that *xterm* should do jump scrolling. Normally, text is scrolled one line at a time; this option allows *xterm* to move multiple lines at a time so that it doesn't fall as far behind. Its use is strongly recommended since it make *xterm* much faster when scanning through large amounts of text. The VT100 escape sequences for enabling and disabling smooth scroll as well as the "Modes" menu can be used to turn this feature on or off.

+j This option indicates that *xterm* should not do jump scrolling.

–l This option indicates that *xterm* should send all terminal output to a log file as well as to the screen. This option can be enabled or disabled using the "xterm X11" menu.

+l This option indicates that *xterm* should not do logging.

–lf *filename*
This option specifies the name of the file to which the output log described above is written. If *file* begins with a pipe symbol (|), the rest of the string is assumed to be a command to be used as the endpoint of a pipe. The default filename is "**XtermLog.***XXXXX*" (where *XXXXX* is the process id of *xterm*) and is created in the directory from which *xterm* was started (or the user's home directory in the case of a login window).

–ls This option indicates that the shell that is started in the *xterm* window be a login shell (i.e. the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile).

+ls This option indicates that the shell that is started should not be a login shell (i.e. it will be a normal "subshell").

–mb This option indicates that *xterm* should ring a margin bell when the user types near the right end of a line. This option can be turned on and off from the "Modes" menu.

+mb This option indicates that margin bell should not be rung.

–ms *color*
This option specifies the color to be used for the pointer cursor. The default is to use the foreground color.

–nb *number*
This option specifies the number of characters from the right end of a line at which the margin bell, if enabled, will ring. The default is 10.

–rw This option indicates that reverse-wraparound should be allowed. This allows the cursor to back up from the leftmost column of one line to the rightmost column of the previous line. This is very useful for editing long

shell command lines and is encouraged. This option can be turned on and off from the "Modes" menu.

+rw This option indicates that reverse-wraparound should not be allowed.

-s This option indicates that *xterm* may scroll asynchronously, meaning that the screen does not have to be kept completely up to date while scrolling. This allows *xterm* to run faster when network latencies are very high and is typically useful when running across a very large internet or many gateways.

+s This option indicates that *xterm* should scroll synchronously.

-sb This option indicates that some number of lines that are scrolled off the top of the window should be saved and that a scrollbar should be displayed so that those lines can be viewed. This option may be turned on and off from the "Modes" menu.

+sb This option indicates that a scrollbar should not be displayed.

-sf This option indicates that Sun Function Key escape codes should be generated for function keys.

+sf This option indicates that the standard escape codes should be generated for function keys.

-si This option indicates that output to a window should not automatically reposition the screen to the bottom of the scrolling region. This option can be turned on and off from the "Modes" menu.

+si This option indicates that output to a window should cause it to scroll to the bottom.

-sk This option indicates that pressing a key while using the scrollbar to review previous lines of text should cause the window to be repositioned automatically in the normal postion at the bottom of the scroll region.

+sk This option indicates that pressing a key while using the scrollbar should not cause the window to be repositioned.

-sl *number*
This option specifies the number of lines to save that have been scrolled off the top of the screen. The default is 64.

-t This option indicates that *xterm* should start in Tektronix mode, rather than in VT102 mode. Switching between the two windows is done using the "Modes" menus.

+t This option indicates that *xterm* should start in VT102 mode.

-tm *string*
This option specifies a series of terminal setting keywords followed by the characters that should be bound to those functions, similar to the *stty* program. This is ignored when -L is given since *getty* resets the terminal. Allowable keywords include: intr, quit, erase, kill, eof, eol, swtch, start, stop, brk, susp, dsusp, rprnt, flush, weras, and lnext. Control characters may be specified as ^char (e.g. ^c or ^u) and ^? may be used to indicate delete.

-tn *name*
This option specifies the name of the terminal type to be set in the TERM environment variable. This terminal type must exist in the *termcap(5)* database and should have *li#* and *co#* entries.

-ut This option indicates that *xterm* shouldn't write a record into the the system log file */etc/utmp*.

+ut This option indicates that *xterm* should write a record into the system log file */etc/utmp*.

-vb This option indicates that a visual bell is prefered over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed.

+vb This option indicates that a visual bell should not be used.

-C This option indicates that this window should receive console output. This is not supported on all systems.

-S*ccn* This option specifies the last two letters of the name of a pseudoterminal to use in slave mode. This allows *xterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications.

The following command line arguments are provided for compatibility with older versions. They may not be supported in the next release as the X Toolkit provides standard options that accomplish the same task.

%geom This option specifies the prefered size and position of the Tektronix window. It is shorthand for specifying the "**tekGeometry*" resource.

#geom This option specifies the prefered position of the icon window. It is shorthand for specifying the "**iconGeometry*" resource.

-T *string*
This option specifies the title for *xterm*'s windows. It is equivalent to **-title**.

-n *string*
This option specifies the icon name for *xterm*'s windows. It is shorthand for specifying the "**iconName*" resource. Note that this is not the same as the toolkit option **-name** (see below). The default icon name is the application name.

-r This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to **-reversevideo** or **-rv**.

-w *number*
This option specifies the width in pixels of the border surrounding the window. It is equivalent to **-borderwidth** or **-bw**.

-L This option indicates that *xterm* was started by *init*. In this mode, *xterm* does not try to allocate a new pseudoterminal as *init* has already done so. In addition, the system program *getty* is run instead of the user's shell. **This option has been superceeded by the new *xdm* program; furthermore, this option should never be used by users when starting terminal windows.**

The following standard X Toolkit command line arguments are commonly used with *xterm*:

-bg *color*
This option specifies the color to use for the background of the window. The default is "white."

-bd *color*
This option specifies the color to use for the border of the window. The default is "black."

-bw *number*
This option specifies the width in pixels of the border surrounding the window.

-fg *color* This option specifies the color to use for displaying text. The default is "black".

-fn *font* This option specifies the font to be used for displaying normal text. The default is "vtsingle."

-name *name*
This option specifies the application name under which resources are to be obtained, rather than the default executable file name. *Name* should not contain "." or "*" characters.

-title *string*
This option specifies the window title string, which may be displayed by window managers if the user so chooses. The default title is the command line specified after the **-e** option, if any, otherwise the application name.

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the prefered size and position of the VT102 window; see *X(1)*.

-display *display*
This option specifies the X server to contact; see *X(1)*.

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

-iconic This option indicates that *xterm* should ask the window manager to start it as an icon rather than as the normal window.

X DEFAULTS

The program understands all of the core X Toolkit resource names and classes as well as:

iconGeometry (class IconGeometry)
Specifies the prefered size and position of the application when iconified. It is not necessarily obeyed by all window managers.

termName (class TermName)
Specifies the terminal type name to be set in the TERM environment variable.

title (class Title)
Specifies a string that may be used by the window manager when displaying this application.

ttyModes (class TtyModes)
Specifies a string containing terminal setting keywords and the characters to which they may be bound. This option is ignored when **-L** is given since *getty* resets the terminal. Allowable keywords include: intr, quit, erase, kill, eof, eol, swtch, start, stop, brk, susp, dsusp, rprnt, flush, weras, and lnext. Control characters may be specified as ^char (e.g. ^c or ^u) and ^? may be used to indicate delete. This is very useful for overriding the default terminal settings without having to do an *stty* every time an *xterm* is started.

utmpInhibit (class **UtmpInhibit**)
Specifies whether or not *xterm* should try to record the user's terminal in */etc/utmp*.

sunFunctionKeys (class **SunFunctionKeys**)
Specifies whether or not Sun Function Key escape codes should be generated for function keys instead of standard escape sequences.

The following resources are specified as part of the "vt100" widget (class "VT100"):

allowSendEvents (class **AllowSendEvents**)
Specifies whether or not synthetic key and button events (generated using the X protocol SendEvent request) should be interpreted or discarded. The default is "false" meaning they are discarded. Note that allowing such events creates a very large security hole.

alwaysHighlight (class **AlwaysHighlight**)
Specifies whether or not *xterm* should always display a highlighted text cursor. By default, a hollow text cursor is displayed whenever the pointer moves out of the window or the window loses the input focus.

font (class **Font**)
Specifies the name of the normal font. The default is "vtsingle."

boldFont (class **Font**)
Specifies the name of the bold font. The default is "vtbold."

c132 (class **C132**)
Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is "false."

charClass (class **CharClass**)
Specifies comma-separated lists of character class bindings of the form *[low-]high:value*. These are used in determining which sets of characters should be treated the same when doing cut and paste. See the section on specifying character classes.

curses (class **Curses**)
Specifies whether or not the last column bug in *curses*(3x) should be worked around. The default is "false."

background (class **Background**)
Specifies the color to use for the background of the window. The default is "white."

foreground (class **Foreground**)
Specifies the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the "text" color change color. The default is "black."

cursorColor (class **Foreground**)
Specifies the color to use for the text cursor. The default is "black."

geometry (class **Geometry**)
Specifies the prefered size and position of the VT102 window.

tekGeometry (class **Geometry**)
Specifies the prefered size and position of the Tektronix window.

internalBorder (class **BorderWidth**)
Specifies the number of pixels between the characters and the window border. The default is 2.

jumpScroll (class **JumpScroll**)
Specifies whether or not jump scroll should be used. The default is "false".

logFile (class **Logfile**)
Specifies the name of the file to which a terminal session is logged. The default is "**XtermLog.***XXXXX*" (where *XXXXX* is the process id of *xterm*).

logging (class **Logging**)
Specifies whether or not a terminal session should be logged. The default is "false."

logInhibit (class **LogInhibit**)
Specifies whether or not terminal session logging should be inhibited. The default is "false."

loginShell (class **LoginShell**)
Specifies whether or not the shell to be run in the window should be started as a login shell. The default is "false."

marginBell (class **MarginBell**)
Specifies whether or not the bell should be run when the user types near the right margin. The default is "false."

multiScroll (class **MultiScroll**)
Specifies whether or not asynchronous scrolling is allowed. The default is "false."

nMarginBell (class **Column**)
Specifies the number of characters from the right margin at which the margin bell should be run, when enabled.

pointerColor (class **Foreground**)
Specifies the color of the pointer. The default is "black."

pointerShape (class **Cursor**)
Specifies the name of the shape of the pointer. The default is "xterm."

reverseVideo (class **ReverseVideo**)
Specifies whether or not reverse video should be simulated. The default is "false."

reverseWrap (class **ReverseWrap**)
Specifies whether or not reverse-wraparound should be enabled. The default is "false."

saveLines (class **SaveLines**)
Specifies the number of lines to save beyond the top of the screen when a scrollbar is turned on. The default is 64.

scrollBar (class **ScrollBar**)
Specifies whether or not the scrollbar should be displayed. The default is "false."

scrollInput (class **ScrollCond**)
Specifies whether or not output to the terminal should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "true."

scrollKey (class **ScrollCond**)
Specifies whether or not pressing a key should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "false."

signalInhibit (class **SignalInhibit**)
Specifies whether or not the entries in the "xterm X11" menu for sending signals to *xterm* should be disallowed. The default is "false."

tekInhibit (class **TekInhibit**)
Specifies whether or not Tektronix mode should be disallowed. The default is "false."

tekStartup (class **TekStartup**)
Specifies whether or not *xterm* should start up in Tektronix mode. The default is "false."

titeInhibit (class **TiteInhibit**)
Specifies whether or not *xterm* should remove remove *ti* or *te* termcap entries (used to switch between alternate screens on startup of many screen-oriented programs) from the TERMCAP string.

translations (class **Translations**)
Specifies the key and button bindings for menus, selections, "programmed strings", etc. See **KEY/BUTTON BINDINGS** below.

visualBell (class **VisualBell**)
Specifies whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received. The default is "false."

The following resources are specified as part of the "tek4014" widget (class "Tek4014"):

width (class **Width**)
Specifies the width of the Tektronix window in pixels.

height (class **Height**)
Specifies the height of the Tektronix window in pixels.

The following resources are specified as part of the "menu" widget:

menuBorder (class **MenuBorder**)
Specifies the size in pixels of the border surrounding menus. The default is 2.

menuFont (class **Font**)
Specifies the name of the font to use for displaying menu items.

menuPad (class **MenuPad**)
Specifies the number of pixels between menu items and the menu border. The default is 3.

The following resources are useful when specified for the Athena Scrollbar widget:

thickness (class **Thickness**)
Specifies the width in pixels of the scrollbar.

background (class **Background**)
Specifies the color to use for the background of the scrollbar.

foreground (class **Foreground**)
Specifies the color to use for the foreground of the scrollbar. The "thumb" of the scrollbar is a simple checkerboard pattern alternating pixels for foreground and background color.

EMULATIONS

The VT102 emulation is fairly complete, but does not support the blinking character attribute nor the double-wide and double-size character sets. *Termcap*(5) entries that work with *xterm* include "xterm", "vt102", "vt100" and "ansi", and *xterm* automatically searches the termcap file in this order for these entries and then sets the "TERM" and the "TERMCAP" environment variables.

Many of the special *xterm* features (like logging) may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences. (See the *"Xterm Control Sequences"* document.)

The Tektronix 4014 emulation is also fairly good. Four different font sizes and five different lines types are supported. The Tektronix text and graphics commands are recorded internally by *xterm* and may be written to a file by sending the COPY escape sequence (or through the **Tektronix** menu; see below). The name of the file will be "**COPY***yy-MM-dd.hh:mm:ss*", where *yy*, *MM*, *dd*, *hh*, *mm* and *ss* are the year, month, day, hour, minute and second when the COPY was performed (the file is created in the directory *xterm* is started in, or the home directory for a login *xterm*).

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the "shift" key. The assignment of the functions described below to keys and buttons may be changed through the resource database; see **KEY/BUTTON BINDINGS** below.

Pointer button one (usually left) is used to save text into the cut buffer. Move the cursor to beginning of the text, and then hold the button down while moving the cursor to the end of the region and releasing the button. The selected text is highlighted and is saved in the global cut buffer and made the PRIMARY selection when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection. If the key/button bindings specify that an X selection is to be made, *xterm* will leave the selected text highlighted for as long as it is the selection owner.

Pointer button two (usually middle) 'types' (pastes) the text from the PRIMARY selection, if any, otherwise from the cut buffer, inserting it as keyboard input.

Pointer button three (usually right) extends the current selection. (Without loss of generality, that is you can swap "right" and "left" everywhere in the rest of this paragraph...) If pressed while closer to the right edge of the selection than the left, it extends/contracts the right edge of the selection. If you contract the selection past the left edge of the selection, *xterm* assumes you really meant the left edge, restores the original selection, then extends/contracts the left edge of the selection. Extension starts in the selection unit mode that the last selection or extension was performed in; you can multiple-click to cycle through them.

By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Since the

cut buffer is globally shared among different applications, you should regard it as a 'file' whose contents you know. The terminal emulator and other text programs should be treating it as if it were a text file, i.e. the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

Clicking button one with the pointer in the scroll region moves the adjacent line to the top of the display window.

Clicking button three moves the top line of the display window down to the pointer position.

Clicking button two moves the display to a position in the saved text that corresponds to the pointer's position in the scrollbar.

Unlike the VT102 window, the Tektronix window dows not allow the copying of text. It does allow Tektronix GIN mode, and in this mode the cursor will change from an arrow to a cross. Pressing any key will send that key and the current coordinate of the cross cursor. Pressing button one, two, or three will return the letters 'l', 'm', and 'r', respectively. If the 'shift' key is pressed when a pointer buton is pressed, the corresponding upper case letter is sent. To distinquish a pointer button from a key, the high bit of the character is set (but this is bit is normally stripped unless the terminal mode is RAW; see *tty*(4) for details).

MENUS

Xterm has three different menus, named **xterm**, **Modes**, and **Tektronix.** Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two section, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the "control" key and pointer button one are pressed in a window. The modes section contains items that apply to both the VT102 and Tektronix windows. The **Secure Keyboard** mode should be used when typing in passwords or other sensitive data; see **SECURITY** below. Notable entries in the command section of the menu are the **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate** and **Kill** which sends the SIGCONT, SIGTSTP, SIGINT, SIGHUP, SIGTERM and SIGKILL signals, respectively, to the process group of the process running under *xterm* (usually the shell). The **Continue** function is especially useful if the user has accidentally typed CTRL-Z, suspending the process.

The **Modes** menu sets various modes in the VT102 emulation, and is popped up when the "control" key and pointer button two are pressed in the VT102 window. In the command section of this menu, the soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes (such as wrap and smooth scroll) to their initial states just after *xterm* has finished processing the command line options. The **Tektronix** menu sets various modes in the Tektronix emulation, and is popped up when the "control" key and pointer button two are pressed in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

SECURITY

X environments differ in their security conciousness. The servers provided by MIT use a host-based mechanism to control access to the server (see *xhost(1)*). If you enable access for a host, and other users are also permitted to run clients on that host, there is the possibility that someone will run an application that will attempt to use the basic services of the X protocol to snoop on your activities, and potentially capture a transcript of everything you type at the keyboard. This is of particular concern when you want to type in a password or other sensitive data. The best solution to this problem is for the industry to choose a standard authorization mechanism, with the necessary operating system support, and to incorporate this into the X protocol (which is already designed to handle such a mechanism). In the mean time, since passwords are most commonly typed to something running in an *xterm* window, a simple mechanism exists for protecting keyboard input in *xterm*.

The **xterm** menu (see **MENUS** above) contains a **Secure Keyboard** entry which, when enabled, ensures that all keyboard input is directed *only* to *xterm* (using the GrabKeyboard protocol request). When an application prompts you for a password (or other sensitive data), you can enable **Secure Keyboard** using the menu, type in the data, and then disable **Secure Keyboard** using the menu again. Only one X client at a time can secure the keyboard, so when you attempt to enable **Secure Keyboard** it may fail. In this case, the bell will sound. If the **Secure Keyboard** succeeds, the foreground and background colors will be exchanged (as if you selected the **Reverse Video** entry in the **Modes** menu); they will be exchanged again when you exit secure mode. If the colors do *not* switch, then you should be *very* suspicious that you are being spoofed. If the application you are running displays a prompt before asking for the password, it is safest to enter secure mode *before* the prompt gets displayed, and to make sure that the prompt gets displayed correctly (in the new colors), to minimize the probability of spoofing. You can also bring up the menu again and make sure that a check mark appears next to the entry.

Secure Keyboard mode will be disabled automatically if your xterm window becomes iconified (or otherwise unmapped), or if you start up a reparenting window manager (that places a title bar or other decoration around the window) while in **Secure Keyboard** mode. (This is a feature of the X protocol not easily overcome.) When this happens, the foreground and background colors will be switched back and the bell will sound in warning.

CHARACTER CLASSES

Clicking the middle mouse button twice in rapid succession will cause all characters of the same class (e.g. letters, white space, punctuation) to be selected. Since different people have different preferences for what should be selected (for example, should filenames be selected as a whole or only the separate subnames), the default mapping can be overridden through the use of the *charClass* (class *CharClass*) resource.

This resource is simply a list of *range:value* pairs where the range is either a single number or *low-high* in the range of 0 to 127, corresponding to the ASCII code for the character or characters to be set. The *value* is arbitrary, although the default table uses the character number of the first character occurring in the set.

The default table is:

```
static int charClass[128] = {
/* NUL SOH STX ETX EOT ENQ ACK BEL */
  32,  1,  1,  1,  1,  1,  1,  1,
/*  BS  HT  NL  VT  NP  CR  SO  SI */
   1, 32,  1,  1,  1,  1,  1,  1,
/* DLE DC1 DC2 DC3 DC4 NAK SYN ETB */
```

```
    1,  1,  1,  1,  1,  1,  1,  1,
/* CAN  EM SUB ESC  FS  GS  RS  US */
    1,  1,  1,  1,  1,  1,  1,  1,
/*  SP   !   "   #   $   %   &   ' */
   32, 33, 34, 35, 36, 37, 38, 39,
/*   (   )   *   +   ,   -   .   / */
   40, 41, 42, 43, 44, 45, 46, 47,
/*   0   1   2   3   4   5   6   7 */
   48, 48, 48, 48, 48, 48, 48, 48,
/*   8   9   :   ;   <   =   >   ? */
   48, 48, 58, 59, 60, 61, 62, 63,
/*   @   A   B   C   D   E   F   G */
   64, 48, 48, 48, 48, 48, 48, 48,
/*   H   I   J   K   L   M   N   O */
   48, 48, 48, 48, 48, 48, 48, 48,
/*   P   Q   R   S   T   U   V   W */
   48, 48, 48, 48, 48, 48, 48, 48,
/*   X   Y   Z   [   \   ]   ^   _ */
   48, 48, 48, 91, 92, 93, 94, 48,
/*   `   a   b   c   d   e   f   g */
   96, 48, 48, 48, 48, 48, 48, 48,
/*   h   i   j   k   l   m   n   o */
   48, 48, 48, 48, 48, 48, 48, 48,
/*   p   q   r   s   t   u   v   w */
   48, 48, 48, 48, 48, 48, 48, 48,
/*   x   y   z   {   |   }   ~ DEL */
   48, 48, 48,123,124,125,126,  1};
```

For example, the string "33:48,37:48,45-47:48,64:48" indicates that the exclamation mark, percent sign, dash, period, slash, and ampersand characters should be treated the same way as characters and numbers. This is very useful for cutting and pasting electronic mailing addresses and Unix filenames.

KEY TRANSLATIONS

It is possible to rebind keys (or sequences of keys) to arbitrary strings for input, by changing the translations for the vt100 or tek4014 widgets. Changing the translations for events other than key and button events is not expected, and will cause unpredictable behavior.

The actions available for key translations are:

secure() Toggles the **Secure Keyboard** mode; see **SECURITY**.

insert() Processes the key in the normal way; i.e. inserts the ASCII character code corresponding to the keysym found in the keyboard mapping table into the input stream.

string(*string***)** Rebinds the key or key sequence to the string value; that is, inserts the string argument into the input stream. Quotation is necessary if the string contains whitespace or non-alphanumeric characters. If the string argument begins with the characters "0x", it is interpreted as a hex character constant and the corresponding character is sent in the normal way.

keymap(*name***)** The **keymap** action takes a single string argument naming a resource to be used to dynamically define a new translation table; the name of the resource is obtained by appending the string "Keymap" to *name*. The keymap name **None** restores the original

translation table (the very first one; a stack is not maintained). Upper/lower case is significant.

insert-selection(*name*[**,***name*]**...)**
Retrieves the value of the first (left-most) named selection that exists or cut buffer that is non-empty and inserts the value into the input stream. *Name* is the name of any selection, for example, **PRIMARY** or **SECONDARY**, or the name of a cut buffer: **CUT_BUFFER0**, ..., **CUT_BUFFER7**. Upper/lower case is significant.

For example, a debugging session might benefit from the following bindings:

```
*VT100.Translations: #override <Key>F13 : keymap(dec)

*VT100.decKeymap.translations: \
                <Key>F14: keymap(None) \n\
                <Key>F17: string("next") string(0x0d) \n\
                <Key>F18: string("step") string(0x0d) \n\
                <Key>F19: string("continue") string(0x0d) \n\
                <Key>F20: string("print ") insert-selection(PRIMARY, CUT_BUFFER0)
```

KEY/BUTTON BINDINGS

Within the VT100 widget the key and button bindings for selecting text, pasting text, and activating the menus are controlled by the translation bindings. In addition to the actions listed above under **KEY TRANSLATIONS**, the following actions are available:

mode-menu() Posts one of the two mode menus, depending on which button is pressed.

select-start() Unselects any previously selected text and begins selecting new text.

select-extend() Continues selecting text from the previous starting position.

start-extend() Begins extending the selection from the farthest (left or right) edge.

select-end(*name*[**,***name*]**...)**
Ends the text selection. *Name* is the name of a selection, or the name of a cut buffer into which the text is to be copied. *Xterm* will assert ownership of all the selections named and will copy the text into each of the cut buffers. Upper/lower case is significant.

ignore() Quietly discards the key or button event.

bell([*volume*]**)** Rings the bell at the specified volume increment above/below the base volume.

The default bindings are:

```
                                <KeyPress>:             insert() \n\
Ctrl            ~Meta           <Btn1Down>:             mode-menu() \n\
                ~Meta           <Btn1Down>:             select-start() \n\
                ~Meta           <Btn1Motion>:           select-extend() \n\
Ctrl            ~Meta           <Btn2Down>:             mode-menu() \n\
~Ctrl           ~Meta           <Btn2Down>:             ignore() \n\
                ~Meta           <Btn2Up>:               insert-selection(PRIMARY, CUT_BUFFER0) \n\
~Ctrl           ~Meta           <Btn3Down>:             start-extend() \n\
                ~Meta           <Btn3Motion>:           select-extend() \n\
                ~Meta           <BtnUp>:                select-end(PRIMARY, CUT_BUFFER0) \n\
                                <BtnDown>:              bell(0)
```

STARTING XTERM FROM INIT

Warning, this feature is now obsolete and may not be supported in future releases. Sites using this method should switch to *xdm* instead.

On operating systems such as 4.3bsd and Ultrix, the server and initial login window are normally started automatically by *init(8)*.

By convention, the pseudoterminal with the highest minor device number (e.g. */dev/ttyqf* and */dev/ptyqf)* is renamed for the lowest display number (e.g. */dev/ttyv0* and */dev/ptyv0*). Machines that have more than one display can repeat this process using *ttyqe* for *ttyv1*, and so on.

Once the pseudoterminals are in place, a line similar to the following may be added to */etc/ttys* (replacing *Xqvss* with the appropriate server and putting it all on one line):

```
ttyv0 "/usr/bin/X11/xterm -L -geom 80x24+1+1 -display :0"
        xterm on secure window="/usr/bin/X11/Xqvss :0"
```

Sites that used to run X10 should note that the colon in the server display number is required.

Although the release will install both the X server and *xterm* in /usr/bin/X11 by default, many sites choose to make a copy of both of these programs on the root partition (usually in /etc) so that they may still be used even if the partition containing /usr/bin/X11 isn't mounted.

Some versions of *init* have relatively small program name buffer sizes and treat all sharp signs as comment delimiters. Sites that wish to list large numbers of options on the xterm line will need to write a small shell script to exec the long xterm line. The best solution, of course, is to use *xdm*.

OTHER FEATURES

Xterm automatically highlights the window border and text cursor when the pointer enters the window (selected) and unhighlights them when the pointer leaves the window (unselected). If the window is the focus window, then the window is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When activated, the current screen is saved and replace with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The *termcap*(5) entry for *xterm* allows the visual editor *vi*(1) to switch to the alternate screen for editing, and restore the screen on exit.

In either VT102 or Tektronix mode, there are escape sequences to change the name of the windows and to specify a new log file name.

ENVIRONMENT

Xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use. The environment variable "WINDOWID" is set to the X window id number of the *xterm* window.

SEE ALSO

resize(1), X(1), pty(4), tty(4)
"Xterm Control Sequences"

BUGS

The **–L** option is no longer needed as the new *xdm* display manager system handles logging in in a much cleaner way. No more messing around with trying to match colors in */etc/ttys* or worrying about an unwanted login window. **This option may**

be removed in future releases.

Xterm will hang forever if you try to paste too much text at one time. It is both producer and consumer for the pty and can deadlock.

Variable-width fonts are not handled reasonably.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

The focus is considered lost if some other client (e.g., the window manager) grabs the pointer; it is difficult to do better without an addition to the protocol.

There needs to be a dialog box to allow entry of log file name and the COPY file name.

Many of the options are not resettable after *xterm* starts.

The Tek widget does not support key/button re-binding.

This manual page is too long. There should be a separate users manual defining all of the non-standard escape sequences.

All programs should be written to use X directly; then we could eliminate this program.

COPYRIGHT

AUTHORS

Far too many people, including:

Loretta Guarino Reid (DEC-UEG-WSL), Joel McCormack (DEC-UEG-WSL), Terry Weissman (DEC-UEG-WSL), Edward Moy (Berkeley), Ralph R. Swick (MIT-Athena), Mark Vandevoorde (MIT-Athena), Bob McNamara (DEC-MAD), Jim Gettys (MIT-Athena), Bob Scheifler (MIT X Consortium), Doug Mink (SAO), Steve Pitschke (Stellar), Ron Newman (MIT-Athena), Jim Fulton (MIT X Consortium), Dave Serisky (HP)

NAME

xtitle – set the title of an xterm window to hostname:cwd

SYNOPSIS

xtitle

DESCRIPTION

The *xtitle* program sets the title of an xterm window to hostname:cwd.

COPYRIGHT

AUTHORS

Mark Patrick, Ardent Computer Corp.

NAME

xwd – dump an image of an X window

SYNOPSIS

xwd [-debug] [-help] [-nobdrs] [-out *file*] [-xy] [-display *display*]

DESCRIPTION

Xwd is an X Window System window dumping utility. *Xwd* allows X users to store window images in a specially formatted dump file. This file can then be read by various other X utilities for redisplay, printing, editing, formatting, archiving, image processing, etc. The target window is selected by clicking the mouse in the desired window. The keyboard bell is rung once at the beginning of the dump and twice when the dump is completed.

OPTIONS

-display *display*
This argument allows you to specify the server to connect to; see *X(1)*.

-help Print out the 'Usage:' command syntax summary.

-nobdrs This argument specifies that the window dump should not include the pixels that compose the X window border. This is useful in situations where you may wish to include the window contents in a document as an illustration.

-out *file* This argument allows the user to explicitly specify the output file on the command line. The default is to output to standard out.

-xy This option applies to color displays only. It selects 'XY' format dumping instead of the default 'Z' format.

-add *value*
This option specifies an signed value to be added to every pixel.

ENVIRONMENT

DISPLAY
To get default host and display number.

FILES

XWDFile.h
X Window Dump File format definition file.

SEE ALSO

xwud(1), X(1)

COPYRIGHT

AUTHORS

Tony Della Fera, Digital Equipment Corp., MIT Project Athena
William F. Wyatt, Smithsonian Astrophysical Observatory

NAME

xwininfo – window information utility for X

SYNOPSIS

xwininfo [-help] [-id *id*] [-root] [-name *name*] [-int] [-tree] [-stats] [-bits] [-events] [-size] [-wm] [-all] [-english] [-metric] [-display *display*]

DESCRIPTION

Xwininfo is a utility for displaying information about windows. Various information is displayed depending on which options are selected. If no options are chosen, **-stats** is assumed.

The user has the option of selecting the target window with the mouse (by clicking any mouse button in the desired window) or by specifying its window id on the command line with the **-id** option. Or instead of specifying the window by its id number, the **-name** option may be used to specify which window is desired by name. There is also a special **-root** option to quickly obtain information on X's root window.

OPTIONS

-help Print out the 'Usage:' command syntax summary.

-id *id* This option allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the mouse might be impossible or interfere with the application.

-name *name*
This option allows the user to specify that the window named *name* is the target window on the command line rather than using the mouse to select the target window.

-root This option specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

-int This option specifies that all X window ids should be displayed as integer values. The default is to display them as hexadecimal values.

-tree This option causes the root, parent, and children windows' ids and names of the selected window to be displayed.

-stats This option causes the display of various attributes pertaining to the location and appearance of the selected window. Information displayed includes the location of the window, its width and height, its depth, border width, class, colormap id if any, map state, backing-store hint, and location of the corners.

-bits This option causes the display of various attributes pertaining to the selected window's raw bits and how the selected window is to be stored. Displayed information includes the selected window's bit gravity, window gravity, backing-store hint, backing-planes value, backing pixel, and whether or not the window has save-under set.

-events This option causes the selected window's event masks to be displayed. Both the event mask of events wanted by some client and the event mask of events not to propagate are displayed.

-size This option causes the selected window's sizing hints to be displayed. Displayed information includes: for both the normal size hints and the zoom size hints, the user supplied location if any; the program supplied location if any; the user supplied size if any; the program supplied size if any; the minimum size if any; the maximum size if any; the resize

increments if any; and the minimum and maximum aspect ratios if any.

-wm This option causes the selected window's window manager hints to be displayed. Information displayed may include whether or not the application accepts input, what the window's icon window # and name is, where the window's icon should go, and what the window's initial state should be.

-metric This option causes all individual height, width, and x and y positions to be displayed in millimeters as well as number of pixels, based on what the server thinks the resolution is. Geometry specifications that are in **+x+y** form are not changed.

-english
This option causes all individual height, width, and x and y positions to be displayed in inches (and feet, yards, and miles if necessary) as well as number of pixels. **-metric** and **-english** may both be enabled at the same time.

-all This option is a quick way to ask for all information possible.

-display *display*
This option allows you to specify the server to connect to; see *X(1)*.

EXAMPLE

The following is a sample summary taken with no options specified:

```
xwininfo ==> Please select the window about which you
    ==> would like information by clicking the
    ==> mouse in that window.

xwininfo ==> Window id: 0x60000f (xterm)

    ==> Upper left X: 4
    ==> Upper left Y: 19
    ==> Width: 726
    ==> Height: 966
    ==> Depth: 4
    ==> Border width: 3
    ==> Window class: InputOutput
    ==> Colormap: 0x80065
    ==> Window Bit Gravity State: NorthWestGravity
    ==> Window Window Gravity State: NorthWestGravity
    ==> Window Backing Store State: NotUseful
    ==> Window Save Under State: no
    ==> Window Map State: IsViewable
    ==> Window Override Redirect State: no
    ==> Corners:  +4+19  -640+19  -640-33  +4-33
```

ENVIRONMENT

DISPLAY
To get the default host and display number.

SEE ALSO

X(1), xprop(1)

BUGS

Using **-stats -bits** shows some redundant information.

Rotation Number of degrees to rotate the image counter-clockwise. Default is no rotation.

Borders If "on", then borders are included in the dump. Default is "on".

ENVIRONMENT

DISPLAY

To get default host and display number.

AUTHOR

Ken Wallich, Ardent Computer Corporation

Modeled after X10 version by:
Benjamin Chen, Electronics Research Laboratory

NAME

xwud – image displayer for X

SYNOPSIS

xwud [-debug] [-help] [-inverse] [-in *file*] [-display *display*]

DESCRIPTION

Xwud is an X Window System window image undumping utility. *Xwud* allows X users to display window images that were saved by *xwd* in a specially formatted dump file. The window image will appear at the coordinates of the original window from which the dump was taken. This is a crude version of a more advanced utility that has never been written. Monochrome dump files are displayed on a color monitor in the default foreground and background colors.

OPTIONS

-help Print out a short description of the allowable options.

-in *file* This option allows the user to explicitly specify the input file on the command line. If no input file is given, the standard input is assumed.

-inverse Applies to monochrome window dump files only. If selected, the window is undumped in reverse video. This is mainly needed because the display is 'write white', whereas dump files intended eventually to be written to a printer are generally 'write black'.

-display *display*
This option allows you to specify the server to connect to; see *X(1)*.

ENVIRONMENT

DISPLAY
To get default display.

FILES

XWDFile.h
X Window Dump File format definition file.

BUGS

Does not attempt to do color translation when the destination screen does not have a colormap exactly matching that of the original window.

SEE ALSO

xwd(1), xpr(1), X(1)

COPYRIGHT

AUTHOR

Tony Della Fera, Digital Equipment Corp., MIT Project Athena
William F. Wyatt, Smithsonian Astrophysical Observatory