NAME

X − a portable, network-transparent window system

SYNOPSIS

The X Window System is a network transparent window system which runs on a wide range of computing and graphics machines. It should be relatively straightforward to build the X Consortium software distribution on most ANSI C and POSIX compliant systems. Commercial implementations are also available for a wide range of platforms.

The X Consortium requests that the following names be used when referring to this software:

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

X Window System is a trademark of X Consortium, Inc.

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 the functions that are available, see the *Xlib - C Language X Interface* manual, the *X Window System Protocol* specification, the *X Toolkit Intrinsics - C Language Interface* manual, and various toolkit documents.

The number of programs that use *X* is quite large. Programs provided in the core X Consortium distribution include: a terminal emulator (*xterm*), a window manager (*twm*), a display manager (*xdm*), a console redirect program (*xconsole*), a mail interface (*xmh*), a bitmap editor (*bitmap*), resource listing/manipulation tools (*appres*, *editres*), access control programs (*xauth*, *xhost*, and *iceauth*), user preference setting programs (*xrdb*, *xcmsdb*, *xset*, *xsetroot*, *xstdcmap*, and *xmodmap*), clocks (*xclock* and *oclock*), a font displayer (*xfd*), utilities for listing information about fonts, windows, and displays (*xlsfonts*, *xwininfo*, *xlsclients*, *xdpyinfo*, *xlsatoms*, and *xprop*), screen image manipulation utilities (*xwd*, *xwud*, and *xmag*), a performance measurement utility (*x11perf*), a font compiler (*bdftopcf*), a font server and related utilities (*xfs*, *fsinfo*, *fslsfonts*, *fstobdf*), an X Image Extension exerciser (*xieperf*), a display server and related utilities (*Xserver*, *rgb*, *mkfontdir*), remote execution utilities (*rstart* and *xon*), a clipboard manager (*xclipboard*), a keyboard description compiler (*xkbcomp*), a utility to terminate clients (*xkill*), and a utility to cause part or all of the screen to be redrawn (*xrefresh*).

Many other utilities, window managers, games, toolkits, etc. are included as user-contributed software in the X Consortium distribution, or are available using anonymous ftp on the Internet. See your site administrator for details.

STARTING UP

There are two main ways of getting the X server and an initial set of client applications started. The particular method used depends on what operating system you are running and whether or not you use other window systems in addition to X.

xdm (the X Display Manager)

If you want to always have X running on your display, your site administrator can set your machine up to use the X Display Manager *xdm*. This program is typically started by the system at boot time and takes care of keeping the server running and getting users logged in. If you are running *xdm*, you will see a window on the screen welcoming you to the system and asking for your username and password. Simply type them in as you would at a normal terminal, pressing the Return key after each. If you make a mistake, *xdm* will display an error message and ask you to try again. After you have successfully logged in, *xdm* will start up your X environment. By default, if you have an executable file named *.xsession* in your home directory, *xdm* will treat it as a program (or shell script) to run to start up your initial clients

(such as terminal emulators, clocks, a window manager, user settings for things like the background, the speed of the pointer, etc.). Your site administrator can provide details.

xinit **(run manually from the shell)**
Sites that support more than one window system might choose to use the *xinit* program for starting X manually. If this is true for your machine, your site administrator will probably have provided a program named "x11", "startx", or "xstart" that will do site-specific initialization (such as loading convenient default resources, running a window manager, displaying a clock, and starting several terminal emulators) in a nice way. If not, you can build such a script using the *xinit* program. This utility simply runs one user-specified program to start the server, runs another to start up any desired clients, and then waits for either to finish. Since either or both of the user-specified programs may be a shell script, this gives substantial flexibility at the expense of a nice interface. For this reason, *xinit* is not intended for end users.

DISPLAY NAMES

From the user's prospective, every X server has a *display name* of the form:

hostname:displaynumber.screennumber

This information is used by the application to determine how it should connect to the server and which screen it should use by default (on displays with multiple monitors):

hostname
The *hostname* specifies the name of the machine to which the display is physically connected. If the hostname is not given, the most efficient way of communicating to a server on the same machine will be used.

displaynumber
The phrase "display" is usually used to refer to collection of monitors that share a common keyboard and pointer (mouse, tablet, etc.). Most workstations tend to only have one keyboard, and therefore, only one display. Larger, multi-user systems, however, frequently have several displays so that more than one person can be doing graphics work at once. To avoid confusion, each display on a machine is assigned a *display number* (beginning at 0) when the X server for that display is started. The display number must always be given in a display name.

screennumber
Some displays share a single keyboard and pointer among two or more monitors. Since each monitor has its own set of windows, each screen is assigned a *screen number* (beginning at 0) when the X server for that display is started. If the screen number is not given, screen 0 will be used.

On POSIX systems, the default display name is stored in your DISPLAY environment variable. This variable is set automatically by the *xterm* terminal emulator. However, when you log into another machine on a network, you will need to set DISPLAY by hand to point to your display. For example,

```
% setenv DISPLAY myws:0
$ DISPLAY=myws:0; export DISPLAY
```

The *xon* script can be used to start an X program on a remote machine; it automatically sets the DISPLAY variable correctly.

Finally, most X programs accept a command line option of **-display** *displayname* to temporarily override the contents of DISPLAY. This is most commonly used to pop windows on another person's screen or as part of a "remote shell" command to start an xterm pointing back to your display. For example,

```
% xeyes -display joesws:0 -geometry 1000x1000+0+0
% rsh big xterm -display myws:0 -ls </dev/null &
```

X servers listen for connections on a variety of different communications channels (network byte streams, shared memory, etc.). Since there can be more than one way of contacting a given server, The *hostname* part of the display name is used to determine the type of channel (also called a transport layer) to be used. X servers generally support the following types of connections:

local

The hostname part of the display name should be the empty string. For example: *:0*, *:1*, and *:0.1*. The most efficient local transport will be chosen.

TCPIP

The hostname part of the display name should be the server machine's IP address name. Full Internet names, abbreviated names, and IP addresses are all allowed. For example: *x.org:0*, *expo:0*, *198.112.45.11:0*, *bigmachine:1*, and *hydra:0.1*.

DECnet

The hostname part of the display name should be the server machine's nodename, followed by two colons instead of one. For example: *myws::0*, *big::1*, and *hydra::0.1*.

ACCESS CONTROL

An X server can use several types of access control. Mechanisms provided in Release 6 are:

Host Access	Simple host-based access control.
MIT-MAGIC-COOKIE-1	Shared plain-text "cookies".
XDM-AUTHORIZATION-1	Secure DES based private-keys.
SUN-DES-1	Based on Sun's secure rpc system.
MIT-KERBEROS-5	Kerberos Version 5 user-to-user.

Xdm initializes access control for the server and also places authorization information in a file accessible to the user. Normally, the list of hosts from which connections are always accepted should be empty, so that only clients with are explicitly authorized can connect to the display. When you add entries to the host list (with *xhost*), the server no longer performs any authorization on connections from those machines. Be careful with this.

The file from which *Xlib* extracts authorization data can be specified with the environment variable **XAUTHORITY**, and defaults to the file **.Xauthority** in the home directory. *Xdm* uses **$HOME/.Xauthority** and will create it or merge in authorization records if it already exists when a user logs in.

If you use several machines and share a common home directory across all of the machines by means of a network file system, you never really have to worry about authorization files, the system should work correctly by default. Otherwise, as the authorization files are machine-independent, you can simply copy the files to share them. To manage authorization files, use *xauth*. This program allows you to extract records and insert them into other files. Using this, you can send authorization to remote machines when you login, if the remote machine does not share a common home directory with your local machine. Note that authorization information transmitted ''in the clear'' through a network file system or using *ftp* or *rcp* can be ''stolen'' by a network eavesdropper, and as such may enable unauthorized access. In many environments, this level of security is not a concern, but if it is, you need to know the exact semantics of the particular authorization data to know if this is actually a problem.

For more information on access control, see the *Xsecurity* manual page.

GEOMETRY SPECIFICATIONS

One of the advantages of using window systems instead of 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 (described below), most X programs accept a command line argument of the form **-geometry** *WIDTHx-HEIGHT+XOFF+YOFF* (where *WIDTH*, *HEIGHT*, *XOFF*, and *YOFF* are numbers) for specifying a preferred size and location for this application's main window.

The *WIDTH* and *HEIGHT* parts of the geometry specification are usually measured in either pixels or characters, depending on the application. The *XOFF* and *YOFF* parts are measured in pixels and are used to specify the distance of the window from the left or right and top and bottom edges of the screen, respectively. Both types of offsets are measured from the indicated edge of the screen to the corresponding edge of the window. The X offset may be specified in the following ways:

+XOFF The left edge of the window is to be placed *XOFF* pixels in from the left edge of the screen (i.e., the X coordinate of the window's origin will be *XOFF*). *XOFF* may be negative, in which case the window's left edge will be off the screen.

-XOFF The right edge of the window is to be placed *XOFF* pixels in from the right edge of the screen. *XOFF* may be negative, in which case the window's right edge will be off the screen.

The Y offset has similar meanings:

+YOFF The top edge of the window is to be *YOFF* pixels below the top edge of the screen (i.e., the Y coordinate of the window's origin will be *YOFF*). *YOFF* may be negative, in which case the window's top edge will be off the screen.

-YOFF The bottom edge of the window is to be *YOFF* pixels above the bottom edge of the screen. *YOFF* may be negative, in which case the window's bottom edge will be off the screen.

Offsets must be given as pairs; in other words, in order to specify either *XOFF* or *YOFF* both must be present. Windows can be placed in the four corners of the screen using the following specifications:

+0+0 upper left hand corner.

-0+0 upper right hand corner.

-0-0 lower right hand corner.

+0-0 lower left hand corner.

In the following examples, a terminal emulator is placed in roughly the center of the screen and a load average monitor, mailbox, and clock are placed in the upper right hand corner:

```
xterm -fn 6x10 -geometry 80x24+30+200 &
xclock -geometry 48x48-0+0 &
xload -geometry 48x48-96+0 &
xbiff -geometry 48x48-48+0 &
```

WINDOW MANAGERS

The layout of windows on the screen is controlled by special programs called *window managers*. Although many window managers will honor geometry specifications as given, others may choose to ignore them (requiring the user to explicitly draw the window's region on the screen with the pointer, for example).

Since window managers are regular (albeit complex) client programs, a variety of different user interfaces can be built. The X Consortium distribution comes with a window manager named *twm* which supports overlapping windows, popup menus, point-and-click or click-to-type input models, title bars, nice icons (and an icon manager for those who don't like separate icon windows).

See the user-contributed software in the X Consortium distribution for other popular window managers.

FONT NAMES

Collections of characters for displaying text and symbols in X are known as *fonts*. A font typically contains images that share a common appearance and look nice together (for example, a single size, boldness, slant, and character set). Similarly, collections of fonts that are based on a common type face (the variations are usually called roman, bold, italic, bold italic, oblique, and bold oblique) are called *families*.

Fonts come in various sizes. The X server supports *scalable* fonts, meaning it is possible to create a font of arbitrary size from a single source for the font. The server supports scaling from *outline* fonts and *bitmap* fonts. Scaling from outline fonts usually produces significantly better results than scaling from bitmap fonts.

An X server can obtain fonts from individual files stored in directories in the file system, or from one or more font servers, or from a mixtures of directories and font servers. The list of places the server looks when trying to find a font is controlled by its *font path*. Although most installations will choose to have the server start up with all of the commonly used font directories in the font path, the font path can be changed at any time with the *xset* program. However, it is important to remember that the directory names are on the **server**'s machine, not on the application's.

Bitmap font files are usually created by compiling a textual font description into binary form, using *bdftopcf*. Font databases are created by running the *mkfontdir* program in the directory containing the source or compiled versions of the fonts. Whenever fonts are added to a directory, *mkfontdir* should be rerun so that the server can find the new fonts. To make the server reread the font database, reset the font path with the *xset* program. For example, to add a font to a private directory, the following commands could be used:

```
% cp newfont.pcf ~/myfonts
% mkfontdir ~/myfonts
% xset fp rehash
```

The *xfontsel* and *xlsfonts* programs can be used to browse through the fonts available on a server. Font names tend to be fairly long as they contain all of the information needed to uniquely identify individual fonts. However, the X server supports wildcarding of font names, so the full specification

-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1

might be abbreviated as:

--courier-medium-r-normal--*-100-*-*-*-*-iso8859-1*

Because the shell also has special meanings for * and *?*, wildcarded font names should be quoted:

```
% xlsfonts -fn '-*-courier-medium-r-normal--*-100-*-*-*-*-*-*'
```

The *xlsfonts* program can be used to list all of the fonts that match a given pattern. With no arguments, it lists all available fonts. This will usually list the same font at many different sizes. To see just the base scalable font names, try using one of the following patterns:

--*-*-*-*-*-0-0-0-0-*-0-*-**
--*-*-*-*-*-0-0-75-75-*-0-*-**
--*-*-*-*-*-0-0-100-100-*-0-*-**

To convert one of the resulting names into a font at a specific size, replace one of the first two zeros with a nonzero value. The field containing the first zero is for the pixel size; replace it with a specific height in pixels to name a font at that size. Alternatively, the field containing the second zero is for the point size; replace it with a specific size in decipoints (there are 722.7 decipoints to the inch) to name a font at that size. The last zero is an average width field, measured in tenths of pixels; some servers will anamorphically scale if this value is specified.

FONT SERVER NAMES

One of the following forms can be used to name a font server that accepts TCP connections:

tcp/*hostname*:*port*
tcp/*hostname*:*port*/*cataloguelist*

The *hostname* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *port* is the decimal TCP port on which the font server is listening for connections. The *cataloguelist* specifies a list of catalogue names, with '+' as a separator.

Examples: *tcp/x.org:7100*, *tcp/198.112.45.11:7100/all*.

One of the following forms can be used to name a font server that accepts DECnet connections:

decnet/*nodename*::font$*objname*
decnet/*nodename*::font$*objname*/*cataloguelist*

The *nodename* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *objname* is a normal, case-insensitive DECnet object name. The *cataloguelist* specifies a list of catalogue names, with '+' as a separator.

Examples: *DECnet/SRVNOD::FONT$DEFAULT*, *decnet/44.70::font$special/symbols*.

COLOR NAMES

Most applications provide ways of tailoring (usually through resources or command line arguments) the colors of various elements in the text and graphics they display. A color can be specified either by an abstract color name, or by a numerical color specification. The numerical specification can identify a color in either device-dependent (RGB) or device-independent terms. Color strings are case-insensitive.

X supports the use of abstract color names, for example, "red", "blue". A value for this abstract name is obtained by searching one or more color name databases. *Xlib* first searches zero or more client-side databases; the number, location, and content of these databases is implementation dependent. If the

name is not found, the color is looked up in the X server's database. The text form of this database is commonly stored in the file *<XRoot>/lib/X11/rgb.txt*, where <XRoot> is replaced by the root of the X11 install tree.

A numerical color specification consists of a color space name and a set of values in the following syntax:

<color_space_name>:*<value>*/.../*<value>*

An RGB Device specification is identified by the prefix "rgb:" and has the following syntax:

rgb:*<red>*/*<green>*/*<blue>*

<red>, *<green>*, *<blue>* := *h* | *hh* | *hhh* | *hhhh*
h := single hexadecimal digits

Note that *h* indicates the value scaled in 4 bits, *hh* the value scaled in 8 bits, *hhh* the value scaled in 12 bits, and *hhhh* the value scaled in 16 bits, respectively. These values are passed directly to the X server, and are assumed to be gamma corrected.

The eight primary colors can be represented as:

black	rgb:0/0/0
red	rgb:ffff/0/0
green	rgb:0/ffff/0
blue	rgb:0/0/ffff
yellow	rgb:ffff/ffff/0
magenta	rgb:ffff/0/ffff
cyan	rgb:0/ffff/ffff
white	rgb:ffff/ffff/ffff

For backward compatibility, an older syntax for RGB Device is supported, but its continued use is not encouraged. The syntax is an initial sharp sign character followed by a numeric specification, in one of the following formats:

#RGB	(4 bits each)
#RRGGBB	(8 bits each)
#RRRGGGBBB	(12 bits each)
#RRRRGGGGBBBB	(16 bits each)

The R, G, and B represent single hexadecimal digits. When fewer than 16 bits each are specified, they represent the most-significant bits of the value (unlike the "rgb:" syntax, in which values are scaled). For example, #3a7 is the same as #3000a0007000.

An RGB intensity specification is identified by the prefix "rgbi:" and has the following syntax:

rgbi:*<red>*/*<green>*/*<blue>*

The red, green, and blue are floating point values between 0.0 and 1.0, inclusive. They represent linear intensity values, with 1.0 indicating full intensity, 0.5 half intensity, and so on. These values will be gamma corrected by *Xlib* before being sent to the X server. The input format for these values is an optional sign, a string of numbers possibly containing a decimal point, and an optional exponent field containing an E or e followed by a possibly signed integer string.

The standard device-independent string specifications have the following syntax:

CIEXYZ:*<X>*/*<Y>*/*<Z>*	(*none*, 1, *none*)
CIEuvY:*<u>*/*<v>*/*<Y>*	(˜.6, ˜.6, 1)
CIExyY:*<x>*/*<y>*/*<Y>*	(˜.75, ˜.85, 1)
CIELab:*<L>*/*<a>*/*<b>*	(100, *none*, *none*)
CIELuv:*<L>*/*<u>*/*<v>*	(100, *none*, *none*)
TekHVC:*<H>*/*<V>*/*<C>*	(360, 100, 100)

All of the values (C, H, V, X, Y, Z, a, b, u, v, y, x) are floating point values. Some of the values are constrained to be between zero and some upper bound; the upper bounds are given in parentheses above.

The syntax for these values is an optional '+' or '-' sign, a string of digits possibly containing a decimal point, and an optional exponent field consisting of an 'E' or 'e' followed by an optional '+' or '-' followed by a string of digits.

For more information on device independent color, see the *Xlib* reference manual.

KEYBOARDS

The X keyboard model is broken into two layers: server-specific codes (called *keycodes*) which represent the physical keys, and server-independent symbols (called *keysyms*) which represent the letters or words that appear on the keys. Two tables are kept in the server for converting keycodes to keysyms:

modifier list

Some keys (such as Shift, Control, and Caps Lock) are known as *modifier* and are used to select different symbols that are attached to a single key (such as Shift-a generates a capital A, and Control-l generates a control character ^L). The server keeps a list of keycodes corresponding to the various modifier keys. Whenever a key is pressed or released, the server generates an *event* that contains the keycode of the indicated key as well as a mask that specifies which of the modifier keys are currently pressed. Most servers set up this list to initially contain the various shift, control, and shift lock keys on the keyboard.

keymap table

Applications translate event keycodes and modifier masks into keysyms using a *keysym table* which contains one row for each keycode and one column for various modifier states. This table is initialized by the server to correspond to normal typewriter conventions. The exact semantics of how the table is interpreted to produce keysyms depends on the particular program, libraries, and language input method used, but the following conventions for the first four keysyms in each row are generally adhered to:

The first four elements of the list are split into two groups of keysyms. Group 1 contains the first and second keysyms; Group 2 contains the third and fourth keysyms. Within each group, if the first element is alphabetic and the the second element is the special keysym *NoSymbol*, then the group is treated as equivalent to a group in which the first element is the lowercase letter and the second element is the uppercase letter.

Switching between groups is controlled by the keysym named MODE SWITCH, by attaching that keysym to some key and attaching that key to any one of the modifiers Mod1 through Mod5. This modifier is called the ''group modifier.'' Group 1 is used when the group modifier is off, and Group 2 is used when the group modifier is on.

Within a group, the modifier state determines which keysym to use. The first keysym is used when the Shift and Lock modifiers are off. The second keysym is used when the Shift modifier is on, when the Lock modifier is on and the second keysym is uppercase alphabetic, or when the Lock modifier is on and is interpreted as ShiftLock. Otherwise, when the Lock modifier is on and is interpreted as CapsLock, the state of the Shift modifier is applied first to select a keysym; but if that keysym is lowercase alphabetic, then the corresponding uppercase keysym is used instead.

OPTIONS

Most X programs attempt to use the same names for command line options and arguments. All applications written with the X Toolkit Intrinsics automatically accept the following options:

−display *display*

This option specifies the name of the X server to use.

−geometry *geometry*

This option specifies the initial size and location of the window.

−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.

−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.

−iconic
This option indicates that the user would prefer that the application's windows initially not be visible as if the windows had be immediately iconified by the user. Window managers may choose not to honor the application's request.

−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.

−selectionTimeout
This option specifies the timeout in milliseconds within which two communicating applications must respond to one another for a selection request.

−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.

−xnllanguage *language[_territory][.codeset]*
This option specifies the language, territory, and codeset for use in resolving resource and other filenames.

−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 explicit command line arguments.

RESOURCES

To make the tailoring of applications to personal preferences easier, X provides a mechanism for storing default values for program resources (e.g. background color, window title, etc.) Resources are specified as strings that are read in from various places when an application is run. Program components are named in a hierarchical fashion, with each node in the hierarchy identified by a class and an instance name. At the top level is the class and instance name of the application itself. By convention, the class name of the application is the same as the program name, but with the first letter capitalized (e.g. *Bitmap* or *Emacs*) although some programs that begin with the letter ''x'' also capitalize the second letter for historical reasons.

The precise syntax for resources is:

ResourceLine	=	Comment | IncludeFile | ResourceSpec | <empty line>
Comment	=	"!" {<any character except null or newline>}
IncludeFile	=	"#" WhiteSpace "include" WhiteSpace FileName WhiteSpace
FileName	=	<valid filename for operating system>
ResourceSpec	=	WhiteSpace ResourceName WhiteSpace ":" WhiteSpace Value
ResourceName	=	[Binding] {Component Binding} ComponentName
Binding	=	"." | "*"

WhiteSpace	=	{<space> | <horizontal tab>}
Component	=	"?" | ComponentName
ComponentName	=	NameChar {NameChar}
NameChar	=	"a"−"z" | "A"−"Z" | "0"−"9" | "_" | "-"
Value	=	{<any character except null or unescaped newline>}

Elements separated by vertical bar (|) are alternatives. Curly braces ({...}) indicate zero or more repetitions of the enclosed elements. Square brackets ([...]) indicate that the enclosed element is optional. Quotes ("...") are used around literal characters.

IncludeFile lines are interpreted by replacing the line with the contents of the specified file. The word "include" must be in lowercase. The filename is interpreted relative to the directory of the file in which the line occurs (for example, if the filename contains no directory or contains a relative directory specification).

If a ResourceName contains a contiguous sequence of two or more Binding characters, the sequence will be replaced with single "." character if the sequence contains only "." characters, otherwise the sequence will be replaced with a single "*" character.

A resource database never contains more than one entry for a given ResourceName. If a resource file contains multiple lines with the same ResourceName, the last line in the file is used.

Any whitespace character before or after the name or colon in a ResourceSpec are ignored. To allow a Value to begin with whitespace, the two-character sequence ‘‘*space*’’ (backslash followed by space) is recognized and replaced by a space character, and the two-character sequence ‘‘*tab*’’ (backslash followed by horizontal tab) is recognized and replaced by a horizontal tab character. To allow a Value to contain embedded newline characters, the two-character sequence ‘‘\n’’ is recognized and replaced by a newline character. To allow a Value to be broken across multiple lines in a text file, the two-character sequence ‘‘*newline*’’ (backslash followed by newline) is recognized and removed from the value. To allow a Value to contain arbitrary character codes, the four-character sequence ‘‘*nnn*’’, where each *n* is a digit character in the range of ‘‘0’’−‘‘7’’, is recognized and replaced with a single byte that contains the octal value specified by the sequence. Finally, the two-character sequence ‘‘\\’’ is recognized and replaced with a single backslash.

When an application looks for the value of a resource, it specifies a complete path in the hierarchy, with both class and instance names. However, resource values are usually given with only partially specified names and classes, using pattern matching constructs. An asterisk (*) is a loose binding and is used to represent any number of intervening components, including none. A period (.) is a tight binding and is used to separate immediately adjacent components. A question mark (?) is used to match any single component name or class. A database entry cannot end in a loose binding; the final component (which cannot be "?") must be specified. The lookup algorithm searches the resource database for the entry that most closely matches (is most specific for) the full name and class being queried. When more than one database entry matches the full name and class, precedence rules are used to select just one.

The full name and class are scanned from left to right (from highest level in the hierarchy to lowest), one component at a time. At each level, the corresponding component and/or binding of each matching entry is determined, and these matching components and bindings are compared according to precedence rules. Each of the rules is applied at each level, before moving to the next level, until a rule selects a single entry over all others. The rules (in order of precedence) are:

1. An entry that contains a matching component (whether name, class, or "?") takes precedence over entries that elide the level (that is, entries that match the level in a loose binding).
2. An entry with a matching name takes precedence over both entries with a matching class and entries that match using "?". An entry with a matching class takes precedence over entries that match using "?".
3. An entry preceded by a tight binding takes precedence over entries preceded by a loose binding.

Programs based on the X Tookit Intrinsics obtain resources from the following sources (other programs usually support some subset of these 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 of the first screen using the *xrdb* program. This is frequently taken care of when the user starts up X through the display

manager or *xinit*.

SCREEN_RESOURCES root window property
Any resources specific to a given screen (e.g. colors) that should be available to clients on all machines should be stored in the SCREEN_RESOURCES property on the root window of that screen. The *xrdb* program will sort resources automatically and place them in RESOURCE_MANAGER or SCREEN_RESOURCES, as appropriate.

application-specific files
Directories named by the environment variable XUSERFILESEARCHPATH or the environment variable XAPPLRESDIR (which names a single directory and should end with a '/' on POSIX systems), plus directories in a standard place (usually under <XRoot>/lib/X11/, but this can be overridden with the XFILESEARCHPATH environment variable) are searched for for application-specific resources. For example, application default resources are usually kept in <XRoot>/lib/X11/app-defaults/. See the *X Toolkit Intrinsics - C Language Interface* manual for details.

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, a file named *$HOME*/.Xdefaults-*hostname* is looked for instead, where *hostname* is the name of the host where the application is executing.

−xrm *resourcestring*
Resources can also be 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 resources (each of which are called *instances*) 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 concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit Intrinsics 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 applications using the X Toolkit Intrinsics 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. For example,

```
bitmap*Dashed: off
XTerm*cursorColor: gold
XTerm*multiScroll: on
XTerm*jumpScroll: on
XTerm*reverseWrap: on
XTerm*curses: on
XTerm*Font: 6x10
XTerm*scrollBar: on
XTerm*scrollbar*thickness: 5
XTerm*multiClickTime: 500
XTerm*charClass: 33:48,37:48,45-47:48,64:48
```

```
XTerm*cutNewline: off
XTerm*cutToBeginningOfLine: off
XTerm*titeInhibit:  on
XTerm*ttyModes:  intr ^c erase ^? kill ^u
XLoad*Background: gold
XLoad*Foreground: red
XLoad*highlight: black
XLoad*borderWidth: 0
emacs*Geometry:  80x65-0-0
emacs*Background:  rgb:5b/76/86
emacs*Foreground:  white
emacs*Cursor:  white
emacs*BorderColor:  white
emacs*Font:  6x10
xmag*geometry: -0-0
xmag*borderColor:  white
```

If these resources were stored in a file called *.Xresources* in your home directory, they could be added to any existing resources in the server with the following command:

```
% xrdb -merge $HOME/.Xresources
```

This is frequently how user-friendly startup scripts merge user-specific defaults into any site-wide defaults. All sites are encouraged to set up convenient ways of automatically loading resources. See the *Xlib* manual section *Resource Manager Functions* for more information.

EXAMPLES

The following is a collection of sample command lines for some of the more frequently used commands. For more information on a particular command, please refer to that command's manual page.

```
%  xrdb $HOME/.Xresources
%  xmodmap -e "keysym BackSpace = Delete"
%  mkfontdir /usr/local/lib/X11/otherfonts
%  xset fp+ /usr/local/lib/X11/otherfonts
%  xmodmap $HOME/.keymap.km
%  xsetroot -solid 'rgbi:.8/.8/.8'
%  xset b 100 400 c 50 s 1800 r on
%  xset q
%  twm
%  xmag
%  xclock -geometry 48x48-0+0 -bg blue -fg white
%  xeyes -geometry 48x48-48+0
%  xbiff -update 20
%  xlsfonts '*helvetica*'
%  xwininfo -root
%  xdpyinfo -display joesworkstation:0
%  xhost -joesworkstation
%  xrefresh
%  xwd | xwud
%  bitmap companylogo.bm 32x32
%  xcalc -bg blue -fg magenta
%  xterm -geometry 80x66-0-0 -name myxterm $*
%  xon filesysmachine xload
```

DIAGNOSTICS

A wide variety of error messages are generated from various programs. The default error handler in *Xlib* (also used by many toolkits) uses standard resources to construct diagnostic messages when errors occur. The defaults for these messages are usually stored in *<XRoot>/lib/X11/XErrorDB*. If this file is not present, error messages will be rather terse and cryptic.

When the X Toolkit Intrinsics encounter errors converting resource strings to the appropriate internal

format, no error messages are usually printed. This is convenient when it is desirable to have one set of resources across a variety of displays (e.g. color vs. monochrome, lots of fonts vs. very few, etc.), although it can pose problems for trying to determine why an application might be failing. This behavior can be overridden by the setting the *StringConversionsWarning* resource.

To force the X Toolkit Intrinsics to always print string conversion error messages, the following resource should be placed in the file that gets loaded onto the RESOURCE_MANAGER property using the *xrdb* program (frequently called *.Xresources* or *.Xres* in the user's home directory):

```
*StringConversionWarnings: on
```

To have conversion messages printed for just a particular application, the appropriate instance name can be placed before the asterisk:

```
xterm*StringConversionWarnings: on
```

SEE ALSO

XConsortium(1), XStandards(1), Xsecurity(1),

appres(1), bdftopcf(1), bitmap(1), editres(1), fsinfo(1), fslsfonts(1), fstobdf(1), iceauth(1), imake(1), makedepend(1), mkfontdir(1), oclock(1), rgb(1), resize(1), rstart(1), twm(1), x11perf(1), x11perfcomp(1), xauth(1), xclipboard(1), xclock(1), xcmsdb(1), xconsole(1), xdm(1), xdpyinfo(1), xfd(1), xfs(1), xhost(1), xieperf(1), xinit(1), xkbcomp(1), xkill(1), xlogo(1), xlsatoms(1), xlsclients(1), xlsfonts(1), xmag(1), xmh(1), xmodmap(1), xon(1), xprop(1), xrdb(1), xrefresh(1), xset(1), xsetroot(1), xstdcmap(1), xterm(1), xwd(1), xwininfo(1), xwud(1), Xserver(1), Xdec(1), XmacII(1), Xsun(1), Xnest(1), Xvfb(1), XF86_Acc(1), XF86_Mono(1), XF86_SVGA(1), XF86_VGA16(1), XFree86(1), kbd_mode(1), *Xlib − C Language X Interface*, and *X Toolkit Intrinsics − C Language Interface*

TRADEMARKS

AUTHORS

A cast of thousands, literally. The Release 6 distribution is brought to you by X Consortium, Inc. The names of all people who made it a reality will be found in the individual documents and source files. The staff members at the X Consortium responsible for this release are: Donna Converse, Gary Cutbill, Stephen Gildea, Jay Hersh, Kaleb Keithley, Matt Landau, Ralph Mor, Janet O'Halloran, Bob Scheifler, Ralph Swick, and Dave Wiggins.