48 The X Window System

The X Window System, also called X11 or simply X, is the foundation for most graphical user environments for UNIX and Linux. X is the natural successor to a window system called (believe it or not) W, which was developed as part of MIT's Project Athena in the early 1980s. Version 10 of the X Window System, released in 1985, was the first to achieve widespread deployment, and version 11 (X11) followed shortly thereafter. Thanks to the system's relatively liberal licensing terms, X spread quickly to other platforms, and multiple implementations emerged. Much as in the case of TCP/IP, X's elegant architecture and flexibility have positioned it as the world's predominant non-Windows GUI.

In 1988, the MIT X Consortium was founded to set the overall direction for the X protocol. Over the next decade, this group and its successors issued a stream of protocol updates. X11R7.5 is today's latest and greatest, with the trend apparently heading toward adding new numbers to the version designation instead of incrementing the existing ones.

XFree86 became the de facto X server implementation for most platforms until a licensing change in 2004 motivated many systems to switch to a fork of XFree86 that was unencumbered by the new licensing clause. That fork is maintained by the nonprofit X.Org Foundation and is the predominant implementation in use today.

In addition, the X.Org server has been ported to Windows for use in the Cygwin Linux compatibility environment. (Several commercial X servers for Windows are also available; see page 1436 for more information.)

This chapter describes the X.Org version of X, which is used by all of our example systems except HP-UX. The implementations of X.Org and XFree86 have diverged architecturally, but most of the administrative details remain the same. It is often possible to substitute **xf86** for **xorg** in commands and filenames to guess at the appropriate XFree86 version.

Solaris systems through version 10 included both the X.Org server and Xsun, yet another implementation of X.[1] Xsun remains common on SPARC systems running Solaris 10, but x86 systems typically run X.Org. However, X.Org now supports SPARC, and the OpenSolaris project has stated that X.Org will be the only supported X platform in the future. Therefore, we do not discuss Xsun here.

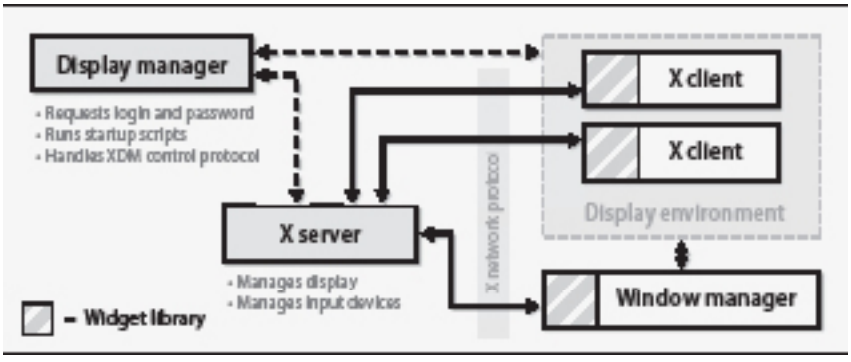
By default, AIX does not include an X Window System environment. To install one, run **smitty easy-install**, select the **lpp** library source, and then choose either CDE (for the traditional IBM-blessed Motif platform) or KDE (for the more modern option).[2] What you get is a highly customized version of the X.Org environment that has been stripped down to look more like the older X systems of the Motif era. However, it supports X11R7.5 under the hood.

The X Window System can be broken down into a few key components. First, it provides a *display manager* whose main job is to authenticate users, log them in, and start up an initial environment from startup scripts. The display manager also starts the *X server*, which defines an abstract interface to the system's bitmapped displays and input devices (e.g., keyboard and mouse). The startup scripts also run a *window manager*, which allows the user to move, resize, minimize, and maximize windows, as well as to manage separate virtual desktops. Finally, at the lowest level, applications are linked to a *widget library* that implements high-level user interface mechanisms such as buttons and menus. Exhibit A illustrates the relationship between the display manager, the X server, and client applications.

The X server understands only a very basic set of drawing primitives over a network API; it does not define a programming interface to high-level entities such as buttons, text boxes, menus, and sliders. This design achieves two important goals. First, it allows the X server to run on a computer that is completely separate from that of the client application. Second, it allows the server to support a variety of different window managers and widget sets.

1. Xsun included support for Display PostScript, which once upon a time was thought to be the display language of the future.
2. It is possible, but not recommended, to have both environments installed simultaneously. See page 1430 for more information about desktop environments.

Exhibit A The X client/server model



Application developers have their choice of several common widget libraries and user interface standards. Unfortunately, the choice often depends more on religious affiliation than on any real design considerations. Although freedom of choice is good, X's UI agnosticism and lack of design leadership did result in many years of poor user interfaces. Fortunately, the fit and finish of the mainstream X environments has improved markedly. Both the KDE and GNOME desktop environments sport modern web browsers, user-friendly file managers, and modern multimedia capabilities.

In this chapter, we explain how to run programs on a remote display and how to enable authentication. We then discuss how to configure the X.Org server and how to troubleshoot configuration errors. Finally, we touch briefly on some of the available window managers and desktop environments.

48.1 THE DISPLAY MANAGER

The display manager presents the user with a (graphical) login screen and is usually the first thing a user sees when sitting down at the computer. It is not required; many users disable the display manager and start X from the text console or from their **.login** script by running **startx** (which itself is a wrapper for the **xinit** program, which starts the X server).

xdm (for X display manager) is the original display manager, but modern replacements such as **gdm** (the GNOME display manager) and **kdm** (the KDE display manager) deliver additional features and are more aesthetically pleasing. The display manager can manage remote logins to other X servers through the XDMCP protocol, and it can also handle display authentication (see *Client authentication* on page 1417).

Configuration files in the **xdm**, **gdm**, or **kdm** subdirectory of **/etc/X11** specify how the display manager will run. For example, you can edit the **Xservers** file to change the display number used for this server if multiple servers will be running on other

virtual terminals. Or, you might alter the server layout with the **-layout** option if you have defined layouts to suit multiple systems.

See page 1041 for more information about PAM.

In the typical scenario, the display manager prompts for a username and password. The user's password is then authenticated according to the PAM configuration specified in **/etc/pam.d/xdm** (or **gdm/kdm** if you are using the GNOME or KDE display managers). The login screen can also offer several alternative desktop environments, including the important failsafe option discussed below.

The display manager's final duty is to execute the **Xsession** shell script, which sets up the user's desktop environment. The **Xsession** script, also most often found in **/etc/X11/**{**xdm,gdm,kdm**}, is a system-wide startup script. It sets application defaults, installs standard key bindings, and selects language settings. The **Xsession** script then executes the user's own personal startup script, usually **~/.xsession**, to start up the window manager, task bar, helper applets, and possibly other programs. GNOME and KDE also have their own startup scripts that configure the user's desktop in accordance with GNOME's and KDE's configuration tools; this scheme is less error-prone than users' editing of their own startup scripts.

When the execution of **~/.xsession** completes, the user is logged out of the system and the display manager goes back to prompting for a username and password. Therefore, **~/.xsession** must start all programs in the background (by appending an **&** to the end of each command) *except for the last one,* which is normally the window manager. (If all commands in **~/.xsession** are run in the background, the script terminates right away and the user is logged out immediately after logging in.) With the window manager as the final, foreground process, the user is logged out only after the window manager exits.

The failsafe login option lets users log in to fix their broken startup scripts. This option can usually be selected from the display manager's login screen. It opens only a simple terminal window; once the window closes, the system logs the user out. Every system should allow the failsafe login option; it helps users fix their own messes rather than having to page you in the middle of the night.

Forgetting to leave a process in the foreground is the most common startup problem, but it's hardly the only possibility. If the cause of problems is not obvious, you may have to refer to the **~/.xsession-errors** file, which contains the output of the commands run from **~/.xsession**. Look for errors or other unexpected behavior. In a pinch, move the **~/.xsession** script aside and make sure you can log in without it. Then restore one or two lines at a time until you find the offending line.

48.2 Process for running an X application

The process required to run an X application may at first seem overly complex. However, you will soon discover the flexibility afforded by the client/server display model. Because display updates are transmitted over the network, an application (the client) can run on a completely separate computer from the one that displays

its graphical user interface (the server). An X server can have connections from many different applications, all of which run on separate computers.

To make this model work, clients must be told what display to connect to and what screen to inhabit on that display. Once connected, clients must authenticate themselves to the X server to ensure that the person sitting in front of the display has authorized the connection.

See page XXX for more information about SSH.

Even with authentication, X's intrinsic security is weak. You can manage connections somewhat more securely by routing them through SSH (see page 1418). We strongly recommend the use of SSH for X connections over the Internet. It's not unreasonable for local traffic, either.

The DISPLAY environment variable

X applications consult the DISPLAY environment variable to find out where to display themselves. The variable contains the hostname or IP address of the server, the display number (identifying the particular instance of an X server to connect to), and an optional screen number (for displays with multiple monitors). When applications run on the same computer that displays their interfaces, you can omit most of these parameters.

The following example shows both the format of the display information and the **bash** syntax used to set the environment variable:

```
client$ DISPLAY=servername.domain.com:10.2; export DISPLAY
```

This setting points X applications at the machine servername.domain.com, display 10, screen 2. Applications establish a TCP connection to the server on port number 6000 plus the display number (in this example, port 6010), where the X server handling that display should be listening.

Keep in mind that every process has its own environment variables. When you set the DISPLAY variable for a shell, its value is inherited only by programs that you run from that shell. If you execute the commands above in one **xterm** and then try to run your favorite X application from another, the application won't have access to your carefully constructed DISPLAY variable.

Another point worth mentioning is that although X applications send their graphical displays to the designated X server, they still have local `stdout` and `stderr` channels. Some error output may still come to the terminal window from which an X application was run.

See page 516 for more information about DNS resolver configuration.

If the client and server are both part of your local organization, you can usually omit the server's full domain name from the DISPLAY variable, depending on how your name server's resolver has been configured. Also, since most systems run only a single X server, the display is usually 0. The screen number can be omitted, in which case screen 0 is assumed. Ergo, most of the time it's fine to set the value of DISPLAY to *servername*:0.

If the client application happens to be running on the same machine as the X server, you can simplify the DISPLAY variable even further by omitting the hostname. This feature is more than just cosmetic: with a null hostname, the client libraries use a UNIX domain socket instead of a network socket to contact the X server. In addition to being faster and more efficient, this connection method bypasses any firewall restrictions on the local system that are trying to keep out external X connections. The simplest possible value for the DISPLAY environment variable, then, is simply ":0".

Client authentication

Although the X environment is generally thought to be relatively insecure, every precaution helps prevent unauthorized access. In the days before security was such a pressing concern, it was common for X servers to welcome connections from any client running on a host that had been marked as safe with the **xhost** command. But since any user on that host could then connect to your display and wreak havoc (either intentionally or out of confusion), the **xhost** method of granting access to clients was eventually deprecated. We do not discuss it further.

The most prevalent alternative to host-based security is called magic cookie authentication. While the thought of magic cookies might inspire flashbacks in some of our readers, in this context they are used to authenticate X connections. The basic idea is that the X display manager generates a large random number, called a cookie, early in the login procedure. The cookie for the server is written to the **~/.Xauthority** file in the user's home directory. Any clients that know the cookie are allowed to connect. Users can run the **xauth** command to view existing cookies and to add new ones to this file.

The simplest way to show how this works is with an example. Suppose you have set your DISPLAY variable on the client system to display X applications on the machine at which you are sitting. However, when you run a program, you get an error that looks something like this:

```
client$ xprogram -display server:0
Xlib: connection to "server:0.0" refused by server
xprogram:  unable to open display 'server:0'
```

This message tells you that the client does not have the right cookie, so the remote server refused the connection. To get the right cookie, log in to the server (which you have probably already done if you are trying to display on it) and list the server's cookies by running **xauth list**:

```
server$ xauth list
server:0  MIT-MAGIC-COOKIE-1  f9d888df6077819ef4d788fab778dc9f
server/unix:0  MIT-MAGIC-COOKIE-1  f9d888df6077819ef4d788fab778dc9f
localhost:0  MIT-MAGIC-COOKIE-1  cb6cbf9e5c24128749feddd47f0e0779
```

Each network interface on the server has an entry. This example shows a cookie for the Ethernet, a cookie for the UNIX domain socket used for local connections, and a cookie for the localhost loopback network interface.

The easiest way to get the cookie onto the client (when not using SSH, which negotiates the cookie for you) is with good old cut-and-paste. Most terminal emulators (e.g., **xterm**[1]) let you select text with the mouse and paste it into another window, usually by pressing the middle mouse button. Conveniently, the **xauth add** command accepts as input the same format that **xauth list** displays. You can add the cookie to the client like this:

```
client$ xauth add server:0  MIT-MAGIC-COOKIE-1
    9d888df6077819ef4d788fab778dc9f
```

You should verify that the cookie was added properly by running **xauth list** on the client. With the DISPLAY environment variable set and the correct magic cookie added to the client, applications should now display correctly on the server.

If you are having trouble getting cookies to work, you can drop back temporarily to **xhost** authentication just to verify that there are no other problems (for example, firewalls or local network restrictions that are preventing the client from accessing the server). Always run **xhost -** (that is, **xhost** with a dash as its only argument) to disable **xhost** authentication once your test is complete.

X connection forwarding with SSH

Magic cookies increase security, but they're hardly foolproof. Any user who can obtain your display's cookie can connect to the display and run programs that monitor your activities. Even without your cookie, the X protocol transfers data over the network without encryption, allowing it to be sniffed by virtually anyone.

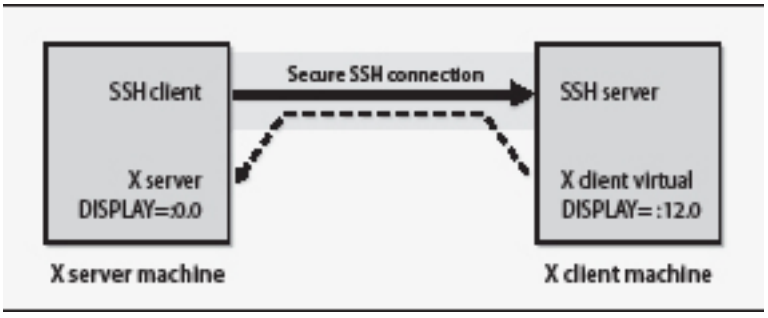
See page XXX for more information about SSH.

You can boost security with SSH, the secure shell protocol. SSH provides an authenticated and encrypted terminal service. However, SSH can also forward arbitrary network data, including X protocol data, over a secure channel. X forwarding is similar to generic SSH port forwarding, but because SSH is X-aware, you gain some additional features, including a pseudo-display on the remote machine and the negotiated transfer of magic cookies.

You typically **ssh** from the machine running the X server to the machine on which you want to run X programs. This arrangement can be confusing to read about because the SSH *client* is run on the same machine as the X *server*, and it connects to an SSH *server* that is on the same machine as the X *client* applications. To make it worse, the virtual display that SSH creates for your X server is local to the remote system. Exhibit B on the next page shows how X traffic flows through the SSH connection.

1. Or **aixterm** on AIX. Clever, hmm?

Exhibit B Using SSH with X



Your DISPLAY variable and authentication information are set up automatically by **ssh**. The display number starts at :10.0 and increments for each SSH connection that is forwarding X traffic.

An example might help show the sequence.

```
x-server$ ssh -v -X x-client.mydomain.com
SSH-2.0-OpenSSH_5.1
debug1: Reading configuration data /home/boggs/.ssh/config
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to x-client.mydomain.com [192.168.15.9] port 22.
debug1: Connection established.
Enter passphrase for key '/home/boggs/.ssh/id_rsa':
debug1: read PEM private key done: type RSA
debug1: Authentication succeeded (publickey).
debug1: Entering interactive session.
debug1: Requesting X11 forwarding with authentication spoofing.
debug1: Requesting authentication agent forwarding.
x-client$
```

You can see from the last two lines that the client is requesting forwarding for X11 applications. X forwarding must be enabled on both the SSH server and the SSH client, and the client must still have the correct cookie for the X server. If things do not seem to be working right, try the **-X** and **-v** flags as shown above (for OpenSSH) to explicitly enable X forwarding and to request verbose output.[1] Also check the global SSH configuration files in **/etc/ssh** to make sure that X11 forwarding has not been administratively disabled. Once logged in, you can check your display and magic cookies:

```
x-client$ echo $DISPLAY
localhost:12.0
x-client$ xauth list
```

1. Note that **ssh** also has a **-Y** flag that trusts all client connections. This feature may solve some forwarding problems, but use it only with extreme caution.

```
x-client/unix:12  MIT-MAGIC-COOKIE-1  a54b67121eb94c8a807f3ab0a67a5
    1f2
```

Notice that the DISPLAY points to a virtual display on the SSH server. Other SSH connections (both from you and from other users) are assigned different virtual display numbers. With the DISPLAY and cookie properly set, the client application can now be run.

```
x-client$ xeyes
debug1: client_input_channel_open: ctype x11 rchan 4 win 65536 max
    16384
debug1: client_request_x11: request from 127.0.0.1 35411
debug1: channel 1: new [x11]
debug1: confirm x11
debug1: channel 1: FORCE input drain
```

With the debugging information enabled with **ssh -v**, you can see that **ssh** has received the X connection request and dutifully forwarded it to the X server. The forwarding can be a little slow on a distant link, but the application should eventually appear on your screen.

48.3 X SERVER CONFIGURATION

The X.Org server, **Xorg**, was once notorious for being difficult to configure for a given hardware environment. However, a tremendous amount of effort has been put into making **Xorg** ready to eat right out of the box, and many modern systems run it successfully without any configuration file. However, it is still possible to manually adapt the **Xorg** server to a wide array of graphics hardware, input devices, video modes, resolutions, and color depths.

If your system is running fine without an **Xorg** configuration file, great! It may be using the KMS module, which is described later in this chapter. Otherwise, you have two options. Option one is to manually configure the **xorg.conf** file. The sections below describe manual configuration. Truth be told, this may be your only real option in some situations. Option two is to use the **xrandr** tool to configure your server; it's covered starting on page 1426.

The **Xorg** configuration file is normally located in **/etc/X11/xorg.conf**, but the X server searches a slew of directories to find it. The man page presents a complete list, but one point to note is that some of the paths **Xorg** searches contain the hostname and a global variable, making it easy for you to store configuration files for multiple systems in a central location.

AIX operates without an **xorg.conf** configuration file and instead tries to automatically recognize all AIX hardware display types. You can pass configuration hints as arguments to the X server.

Several programs can help you configure X (e.g., **xorgconfig**), but it's a good idea to understand how the configuration file is structured in case you need to view or edit

the configuration directly. You can gather some useful starting information directly from the X server by running **Xorg -probeonly** and looking through the output to identify your video chipset and any other probed values. You can also run **Xorg -configure** to have the X server create an initial configuration file that is based on the probed values. It's a good place to start if you have nothing else.

The **xorg.conf** file has several sections, each of which starts with the `Section` keyword and ends with `EndSection`. Table 48.1 lists the most common section types.

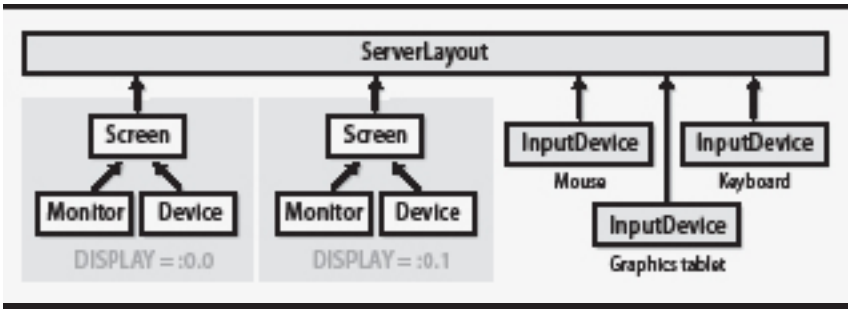
Table 48.1 Sections of the xorg.conf file

Section	Description
`ServerFlags`	Lists general X server configuration parameters
`Module`	Specifies dynamically loadable extensions for accelerated graphics, font renderers, and the like
`Device`	Configures the video card, driver, and hardware information
`Monitor`	Describes physical monitor parameters, including timing and display resolutions
`Screen`	Associates a monitor with a video card (`Device`) and defines the resolutions and color depths available in that configuration
`InputDevice`	Specifies input devices such as keyboards and mice
`ServerLayout`	Bundles input devices with a set of screens and positions the screens relative to each other

It is often simplest to build a configuration file from the bottom up by first defining sections for the input and output devices and then combining them into various layouts. With this hierarchical approach, a single configuration file can be used for many X servers, each with different hardware. It's also a reasonable approach for a single system that has multiple video cards and monitors.

Exhibit C shows how some of these sections fit together into the X.Org configuration hierarchy. A physical display `Monitor` plus a video card `Device` combine to form a `Screen`. A set of `Screens` plus `InputDevices` form a `ServerLayout`. Multiple server layouts can be defined in a configuration file, though only one is active for a given instance of the **Xorg** process.

Exhibit C Relationship of xorg.conf configuration sections



Some of the sections that make up the **xorg.conf** file are relatively fixed. The defaults can often be used straight from an existing or example configuration file. Others, such as the `Device`, `Monitor`, `Screen`, `InputDevice`, and `ServerLayout` sections, depend on the host's hardware setup. We discuss the most interesting of these sections in more detail in the following subsections.

Device sections

A `Device` section describes a particular video card. You must provide a string to identify the card and a driver appropriate for the device. The driver is loaded only if the device is referenced by a corresponding `Screen` section. A typical device section might look like this:

```
Section "Device"
  Identifier                    "Videocard0"
  Driver                        "radeon"
  option                        value
  ...
EndSection
```

The manual page for the driver, `radeon` in this example, describes the hardware that's driven as well as the options the driver supports. If you are experiencing strange video artifacts, you might try setting options to turn off hardware acceleration (if supported), slowing down video memory access, or modifying interface parameters. It is generally a good idea to check the web to see if other people have experienced similar problems before you start randomly changing values.

Monitor sections

The `Monitor` section describes the displays attached to your computer. It can specify detailed timing values. The timing information is necessary for older hardware, but most modern monitors can be probed for it. Display specifications can usually be obtained from the manufacturer's web site, but nothing beats having the original manual that came with the monitor. Either way, you will want to know at least the horizontal sync and vertical refresh frequencies for your model.

A typical Monitor section looks like this:

```
Section "Monitor"
  Identifier                "ViewSonic"
  Option                    "DPMS"
  HorizSync                 30-65
  VertRefresh                       50-120
EndSection
```

As with all of the sections, the Identifier line assigns a name by which you later refer to this monitor. Here we have turned on DPMS (Display Power Management Signaling) so that the X server can power down the monitor when we sneak away for a donut and some coffee.

The HorizSync and VertRefresh lines, which apply only to CRT monitors, should be filled in with values appropriate for your monitor. They may be specified as a frequency range (as above) or as discrete values separated by commas. The driver can theoretically probe for supported modes, but specifying the parameters keeps the driver from attempting to use unsupported frequencies.

Screen sections

A Screen section ties a device (video card) to a monitor at a specific color depth and set of display resolutions. Here's an example that uses the video card and monitor specified above.

```
Section "Screen"
  Identifier                "Screen0"
  Device                    "Videocard0"
  Monitor                   "ViewSonic"
  DefaultDepth                      24
  Subsection "Display"
    Depth           8
    Modes           "640x400"
  EndSubsection
  Subsection "Display"
    Depth           16
    Modes           "640x400" "640x480" "800x600" "1024x768"
  EndSubsection
  Subsection "Display"
    Depth           24
    Modes           "1280x1024" "1024x768" "800x600" "640x400"
    "640x480"
  EndSubsection
EndSection
```

As you might expect, the screen is named with an Identifier, and the identifiers for the previously defined video device and monitor are mentioned. This is the first section we have introduced that has subsections. One subsection is defined for each color depth, with the default being specified by the DefaultDepth field.

A given instance of the X server can run at only one color depth. At startup, the server determines what resolutions are supported for that depth. The possible resolutions generally depend on the video card. *Special keyboard combinations for X* on page 1427 describes how to cycle through the resolutions that are defined here.

Any modern video card should be able to drive your monitor at its full resolution in 24-bit or 32-bit color. If you want to run old programs that require a server running in 8-bit color, run a second X server on a separate virtual console. Use the **-depth 8** flag on the **Xorg** command line to override the `DefaultDepth` option.

InputDevice sections

An `InputDevice` section describes a source of input events such as a keyboard or mouse. Each device gets its own `InputDevice` section, and as with other sections, each is named with an `Identifier` field. If you are sharing a single configuration file among machines with different hardware, you can define all the input devices; only those referenced in the `ServerLayout` section are used. Here is a typical keyboard definition:

```
Section "InputDevice"
  Identifier                "Generic Keyboard"
  Driver                    "Keyboard"
  Option                    "AutoRepeat" "500 30"
  Option                    "XkbModel" "pc104"
  Option                    "XkbLayout" "us"
EndSection
```

You can set options in the keyboard definition to express your particular religion's stance on the proper position of the Control and Caps Lock keys, among other things. In this example, the `AutoRepeat` option specifies how long a key needs to be held down before it starts repeating and how fast it repeats.

The mouse is configured in a separate `InputDevice` section:

```
Section "InputDevice"
  Identifier                "Generic Mouse"
  Driver                    "mouse"
  Option                    "CorePointer"
  Option                    "Device" "/dev/input/mice"
  Option                    "Protocol" "IMPS/2"
  Option                    "Emulate3Buttons" "off"
  Option                    "ZAxisMapping" "4 5"
EndSection
```

The `CorePointer` option designates this mouse as the system's primary pointing device. The device file associated with the mouse is specified as an `Option`; Table 48.2 lists the mouse device multiplexer files for our example systems.

Table 48.2 Common mouse device files

OS	Device file
Linux	**/dev/input/mice**
Solaris	**/dev/mouse**
HP-UX	**/dev/deviceFileSystem/mouseMux**
AIX	**/dev/mouse0**

The communication protocol depends on the particular brand of mouse, its features, and its interface. You can set it to `auto` to make the server try to figure out the protocol for you. If your mouse wheel doesn't work, try setting the protocol to `IMPS/2`. If you have more than a few buttons, try using the `ExplorerPS/2` protocol. Some Solaris users report success with the `VUID` protocol.

The `Emulate3Buttons` option lets a two-button mouse emulate a three-button mouse by defining a click on both buttons to stand in for a middle-button click. The `ZAxisMapping` option is sometimes needed to support a scroll wheel or joystick device. Most mice these days have at least three buttons, a scroll wheel, a built-in MP3 player, a foot massager, and a beer chiller.[1]

ServerLayout sections

The `ServerLayout` section is the top-level node of the configuration hierarchy. Each hardware configuration on which the server will be run should have its own instance of the `ServerLayout` section. The layout used by a particular X server is usually specified on the server's command line.

This section ties together all the other sections to represent an X display. It starts with the requisite `Identifier`, which names this particular layout. It then associates a set of screens with the layout.[2] If multiple monitors are attached to separate video cards, each screen is specified along with optional directions to indicate how they are physically arranged. In this example, screen one is on the left and screen two is on the right.

Here is an example of a complete `ServerLayout` section:

```
Section "ServerLayout"
  Identifier         "Simple Layout"
  Screen             "Screen 1" LeftOf "Screen 2"
  Screen             "Screen 2" RightOf "Screen 1"
  InputDevice                "Generic Mouse" "CorePointer"
  InputDevice                "Generic Keyboard" "CoreKeyboard"
  Option             "BlankTime"            "10"           # Blank
```

1. Not all options are supported by **Xorg**. Some options sold separately.
2. Recall that screens identify a monitor/video card combination at a particular color depth.

```
    the screen in 10 minutes
  Option            "StandbyTime"          "20"          # Turn off
    screen in 20 minutes (DPMS)
  Option            "SuspendTime"          "60"          # Full
    hibernation in 60 minutes (DPMS)
  Option            "OffTime"              "120"         # Turn off
    DPMS monitor in 2 hours
EndSection
```

Some video cards can drive multiple monitors at once. In this case, only a single `Screen` is specified in the `ServerLayout` section. Following the screen list is the set of input devices to associate with the layout. The `CorePointer` and `CoreKeyboard` options are passed to the `InputDevice` section to indicate that the devices are to be active for the configuration. Those options can also be set directly in the corresponding `InputDevice` sections, but it's cleaner to set them in the `ServerLayout`.

The last few lines configure several layout-specific options. In the example above, these all relate to DPMS, which is the interface that tells Energy Star-compliant monitors when to power themselves down. The monitors must also have their DPMS options enabled in the corresponding `Monitor` sections.

xrandr: not your father's X server configurator

The X Resize and Rotate Extension (RandR) lets clients dynamically change the size, orientation, and reflection of their X server screens. **xrandr** is the command-line interface to this extension.

Of course, we would all love to spend a few days tediously crafting each line of the **xorg.conf** file to support that brand-new SUPERINATOR 3000 system with its four deluxe displays. But in many cases, you can have **xrandr** do the configuration for you and be done in time to grab a few beers. Run with no arguments, **xrandr** shows the available displays and their possible resolutions.

```
$ xrandr
VGA-0 connected 1024x768+0+0 (normal left inverted right x a...) 0mm x
    0mm
    1024x768  61.0     60.0    59.9        59.9
    800x600   60.3     61.0    59.9        56.2    59.8
    640x480   59.9     61.0    59.4        59.5
DVI-0 connected 1024x768+0+0 (normal left inverted right x a...) 0mm x
    0mm
    1024x768  60.0     60.0
    800x600   60.3     59.9
    640x480   59.9     59.4
```

You can specify the resolution to use for each display along with the display's placement relative to other displays.[1] For example:

1. Before using **xrandr** for the first time, run **Xorg -configure** to reset the **xorg.conf** file to a known, clean state.

```
$ xrandr --auto --output VGA-0 --mode 800x600 --right-of DVI-0
```

The --**auto** argument turns on all available monitors. The --**output** and --**mode** arguments set the VGA display to a resolution of 800 □ 600, and the --**right-of** argument specifies that the VGA display is physically located to the right of the DVI display. (The latter option is needed to properly implement desktop continuity.) Run **xrandr** --**help** to see the many available options.

If you want **xrandr** to run automatically when you start the X server, you can put it in your ~**/.xprofile** file, which is executed at server startup.

Kernel mode setting

To make the system's presentation more seamless and flicker free, responsibility for setting the initial mode of the graphics display is now being pushed into the Linux kernel through the "kernel mode setting" (KMS) module. As of kernel version 2.6.30-10.12, KMS defaults to initializing the video card very early in the kernel's boot sequence.

You enable or disable KMS through settings in the video driver configuration files in **/etc/modprobe.d**. For example, if you have an ATI Radeon video card, you can turn off KMS by adding the following line to **/etc/modprobe.d/radeon.conf**:

```
options radeon modeset=0
```

The KMS module is still young and it does not currently support all video cards. If you're lucky enough to have a supported card, your best bet is to rename the **xorg.conf** file so that the X server tries to start without it and defaults to the KMS configuration.

48.4 X SERVER TROUBLESHOOTING AND DEBUGGING

X server configuration has come a long way over the last decade, but it can still be difficult to get things working just the way you would like. You may need to experiment with monitor frequencies, driver options, proprietary drivers, or extensions for 3D rendering. Ironically, it is the times when the display is not working correctly that you are most interested in seeing the debugging output on your screen. Fortunately, the X.Org server gives you all the information you need (and a lot that you don't) to track down the problem.

Special keyboard combinations for X

Because the X server takes over your keyboard, display, mouse, and social life, you can imagine that it might leave you with little recourse but to power the system down if things are not working. However, there are a few things to try before it comes to that.

If you hold down the Control and Alt keys and press a function key (F1–F6), the X server takes you to one of the text-based virtual terminals. From there you can log in and debug the problem. To get back to the X server running on, say, virtual terminal 7, press <Alt-F7>.[1] If you are on a network, you can also try logging in from another computer to kill the X server before resorting to the reset button.

For virtual console support on Solaris, enable the svc:/system/vtdaemon:default SMF service and the console-login:vt[2-6] services.

If the monitor is not in sync with the card's video signal, try changing the screen resolution. The available resolutions are specified on a `Modes` line from the `Screen` section of the configuration file. The exact `Modes` line that is active depends on the color depth; see page 1423 for details. The X server defaults to the first resolution shown on the active `Modes` line, but you can cycle through the different resolutions by holding down Control and Alt and pressing the plus (+) or minus (-) key on the numeric keypad.

Pressing <Control-Alt-Backspace> kills the X server immediately. If you ran the server from a console, you will find yourself back there when the server exits. If a display manager started the server, it usually respawns a new server and prompts again for a login and password. You have to kill the display manager (**xdm**, **gdm**, etc.) from a text console to stop it from respawning new X servers.

When X servers attack

Once you have regained control of the machine, you can begin to track down the problem. The simplest place to start is the output of the X server. This output is occasionally visible on virtual terminal 1 (<Control-Alt-F1>), which is where startup program output goes. Most often, the X server output goes to a log file such as **/var/log/Xorg.0.log** (**/var/X11/Xserver/logs/Xf86.0.log** on HP-UX).

As seen below, each line is preceded by a symbol that categorizes it. You can use these symbols to spot errors (`EE`) and warnings (`WW`), as well as to determine how the server found out each piece of information: through default settings (`==`), in a config file (`**`), detected automatically (`--`), or specified on the X server command line (`++`).

Let's examine the following snippet from an Ubuntu system:

```
X.Org X Server 1.6.0
Release Date: 2009-2-25
X Protocol Version 11, Revision 0
Build Operating System: Linux 2.6.24-23-server i686 Ubuntu
Current Operating System: Linux nutrient 2.6.28-11-generic #42-Ubuntu
    SMP Fri Apr 17 01:57:59 UTC 2009 i686
Build Date: 09 April 2009  02:10:02AM
xorg-server 2:1.6.0-0ubuntu14 (buildd@rothera.buildd)
```

1. The X server requires the <Control> key to be held down along with the <Alt-Fn> key combination to switch virtual terminals, but the text console does not.

```
    Before reporting problems, check http://wiki.x.org
    to make sure that you have the latest version.
Markers: (--) probed, (**) from config file, (==) default setting,
    (++) from command line, (!!) notice, (II) informational,
    (WW) warning, (EE) error, (NI) not implemented, (??) unknown.
(==) Log file: "/var/log/Xorg.0.log", Time: Sun May 10 22:11:47 2009
(==) Using config file: "/etc/X11/xorg.conf"
(==) ServerLayout "MainLayout"
(**) |-->Screen "Screen 0" (0)
(**) |   |-->Monitor "Monitor 0"
(**) |   |-->Device "Console"
(**) |-->Input Device "Mouse0"
(**) |-->Input Device "Keyboard0"
...
```

The first lines tell you the version number of the X server and the X11 protocol version it implements. Subsequent lines tell you that the server is using default values for the log file location, the configuration file location, and the active server layout. The display and input devices from the config file are echoed in schematic form.

One common problem that shows up in the logs is difficulty with certain screen resolutions, usually evidenced by those resolutions not working or the X server bailing out with an error such as "Unable to validate any modes; falling back to the default mode." If you have not specified a list of frequencies for your monitor, the X server probes for them by using Extended Display Identification Data (EDID). If your monitor does not support EDID or if your monitor is turned off when X is started, you need to put the frequency ranges for X to use in the `Monitor` section of the configuration file.

Rounding error in the results obtained from an EDID probe can cause some resolutions to be unavailable even though they should be supported by both your video card and monitor. Log entries such as "No valid modes for 1280x1024; removing" are evidence of this. The solution is to tell the X server to ignore EDID information and use the frequencies you have specified; the following lines in the `Device` section are what you need:

```
  Option         "IgnoreEDID" "true"
    Option   "UseEdidFreqs" "false"
```

As another example, suppose you forgot to define the mouse section properly. The error would show up like this in the output:

```
(==) Using config file: "/etc/X11/xorg.conf"
Data incomplete in file /etc/X11/xorg.conf
    Undefined InputDevice "Mouse0" referenced by ServerLayout
    "MainLayout".
(EE) Problem parsing the config file
(EE) Error parsing the config file
Fatal server error:
no screens found
```

Once X is up and running and you have logged in, you can run the **xdpyinfo** command to get more information about the X server's configuration.[1] **xdpyinfo**'s output again tells you the name of the display and the X server version information. It also tells you the color depths that are available, the extensions that have been loaded, and the screens that have been defined, along with their dimensions and color configurations.

xdpyinfo's output can be parsed by a script (such as your **~/.xsession** file) to determine the size of the active screen and to set up the desktop parameters appropriately. For debugging, **xdpyinfo** is most useful for determining that the X server is up and listening to network queries, that it has configured the correct screen and resolution, and that it is operating at the desired color bit depth. If this step works, you are ready to start running X applications.

48.5 A BRIEF NOTE ON DESKTOP ENVIRONMENTS

The flexibility of the X Window System client/server model has, over the years, led to an explosion of widget sets, window managers, file browsers, tool bar utilities, and utility programs. The first comprehensive environments, OpenLook and Motif, were elegant for their time but proprietary. Licensing fees for the development libraries and window manager made them inaccessible to the general public.

As applications became more advanced and demanded progressively more support from the underlying window system, it became clear that a comprehensive approach to advancing the platform was required. From this need were born the two big players in modern desktop environments: GNOME and KDE. Although some users have strong feelings regarding which is the One True Way, both are relatively complete desktop managers. In fact, just because you are running in one realm does not mean you cannot use applications from the other; just expect a different look and feel and a brief sense of discontinuity in the universe.

The freedesktop.org project is dedicated to creating an environment that will allow applications to be compatible with any desktop environment.

KDE

KDE, which stands for the K Desktop Environment, is written in C++ and built on the Qt tool kit library. It is often preferred by users who enjoy eye candy, such as transparent windows, shadows, and animated cursors. It looks nice, but it can be slow on anything but a high-end workstation. For users who spend a lot of time clicking around in the desktop rather than running applications, the tradeoff between efficiency and aesthetics may ultimately decide whether KDE is the appropriate choice.

1. We don't recommend logging into X as root because this operation may create a bunch of default start-up files in root's home directory, which is usually **/** or **/root**. It's also notably insecure. Instead, log in as a regular user and use **sudo**. Ubuntu enforces this discipline by default.

KDE is often preferred by people transitioning from a Windows or Mac environment because of its pretty graphics. It's also a favorite of technophiles who love to be able to fully customize their environment. For others, KDE is simply too much to deal with and GNOME is the simpler choice.

Applications written for KDE almost always contain a K somewhere in the name, for example, Konqueror (the web/file browser), Konsole (the terminal emulator), or KWord (a word processor). The default window manager, KWin, supports the freedesktop.org Window Manager Specification standard, configurable skins for changing the overall look and feel, and many other features. The KOffice application suite contains word processing, spreadsheet, and presentation utilities. KDE sports a comprehensive set of development tools, including an integrated development environment (IDE).

GNOME

GNOME is written in C and is based on the GTK+ widget set. The name GNOME was originally an acronym for GNU Network Object Model Environment, but that derivation no longer really applies; these days, GNOME is just a name.

With the recent addition of support for Compiz (compiz.org), GNOME has acquired many of the eye candy features that it previously lacked. Overall, GNOME is still less glitzy than KDE, is not as configurable, and is slightly less consistent. However, it is noticeably cleaner, faster, simpler, and more elegant. Most Linux distributions use GNOME as the default desktop environment.

Like KDE, GNOME has a rich application set. GNOME applications are usually identifiable by the presence of a G in their names. One of the exceptions is the standard GNOME window manager, called Metacity (pronounced like "opacity"), which supplies basic windowing functions and skinning of the GNOME UI. Following the GNOME model, Metacity is designed to be lean and mean.

If you want some of the extra features you may be used to, such as smart window placement, you need the support of external applications such as **brightside** or **devilspie**. Unfortunately, bling is one area in which KDE still has a leg up.

Office applications include AbiWord for word processing, Gnumeric as a spreadsheet, and one of the more impressive projects to come out of GNOME, The GIMP for image processing. A file manager called Nautilus is also included. Like KDE, GNOME provides an extensive infrastructure for application developers. Altogether, GNOME offers a powerful architecture for application development in an easy-to-use desktop environment.

Which is better, GNOME or KDE?

Ask this question on any public forum and you will see the definition of "flame war." Because of the tendency for people to turn desktop preference into a personal crusade, the following paragraphs may be some of the least opinionated in this book.

The best answer is to try both desktops and decide for yourself which best meets your needs. Keep in mind that your friends, your users, and your manager may all have different preferences for a desktop environment, and that is OK.

Remember that your choice of desktop environment does not dictate which applications you can run. No matter which desktop you choose, you can select applications from the full complement of excellent software made available by both of these (and other) open source projects.

48.6 Recommended reading

The X.Org home page, x.org, includes information on upcoming releases as well as links to the X.Org wiki, mailing lists, and downloads.

The man pages for **Xserver** and **Xorg** (or just **X** on AIX) cover generic X server options and **Xorg**-specific command-line options. They also include a general overview of X server operation.

The **xorg.conf** man page covers the config file and describes its various sections in detail. This man page also lists video card drivers in its REFERENCES section. Look up your video card here to learn the name of the driver, then read the driver's own man page to learn about driver-specific options.