In this chapter:
- *The problem with boards and monitors*
- *X configuration: the theory*
- *XF86Config*
- *Multiple monitors and servers*
- *X in the network*

28

XFree86 in depth

The information in Chapter 6, should be enough to get X up and running. There's a lot more to X than that, however, enough to fill many books. In this chapter we'll look at some of the more interesting topics:

- The next section describes the technical background of running X displays.
- On page 516 we'll look at setting up the *XF86Config* file.
- On page 522 we'll look at using more than one monitor with X.
- On page 523 we'll look at using X in a network.

X configuration: the theory

Setting up your *XF86Config* file normally takes a few minutes, but sometimes you can run into problems that make grown men cry. In the rest of this chapter, we'll look at the technical background:

- How display boards and monitors work.
- How to set up XFree86 to work with your hardware.
- How to tune your hardware for maximum display performance.
- How to fry your monitor.

I mean the last point seriously: conventional wisdom says that you can't damage

hardware with a programming mistake, but in this case, you can, and people do it from time to time. When you've read the section on how monitors work, you'll understand, but *please* don't start tuning until you understand the dangers involved.

How TVs and monitors work

You don't have to be a computer expert to see the similarity between monitors and TVs: current monitor technology is derived from TV technology, and many older display boards have modes that can use TVs instead of monitors. Those of us who were on the microcomputer scene 20 to 25 years ago will remember the joy of getting a computer display on a portable TV, a ''glass tty'' connected by a serial line running at 300 or 1200 bps.

There are at least two ways to create pictures on a cathode ray tube: one is derived from oscilloscopes, where each individual character is scanned by the electron beam, rather like writing in the sand with your finger. Some early terminals used this technology, but it has been obsolete for several decades.

TVs and monitors display the picture by scanning equally spaced lines across the entire screen. Like in a book, the first line starts at the top left of the screen and goes to the top right. Each successive line starts slightly below the previous line. This continues until the screen is full. The picture is formed by altering the intensity of the electron beam as it scans the lines.

To perform this scan, the TV has two *deflection units*: one scans from left to right, and the other scans, much more slowly, from top to bottom. Not surprisingly, these units are called the *horizontal* and *vertical* deflection units. You may also encounter the terms *line* and *frame* deflection.

Figure 28-1 shows the resultant pattern.

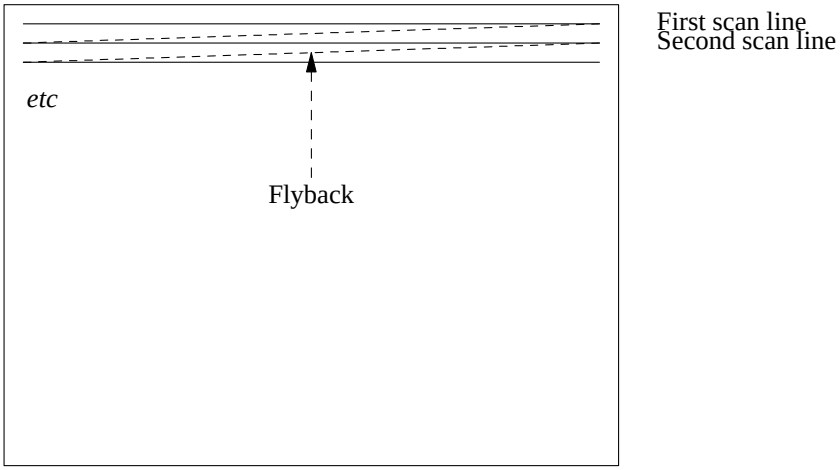

Figure 28-1: Scanning pattern on the monitor

The tube can only move the electron beam at a finite speed. When the electron beam reaches the right hand side of the screen, it needs to be deflected back again. This part of the scan is called the *horizontal flyback*, and it is not used for displaying picture data. The actual time that the hardware requires for the flyback depends on the monitor, but it is in the order of 5% to 10% of the total line scan time. Similarly, when the vertical deflection reaches the bottom of the screen, it performs a *vertical flyback*, which is also not used for display purposes.

It's not enough to just deflect, of course: somehow you need to ensure that the scanning is synchronized with the incoming signal, so that the scan is at the top of the screen when the picture information for the top of the screen arrives. You've seen what happens when synchronization doesn't work: the picture runs up and down the screen (incorrect vertical synchronization) or tears away from the left of the screen (incorrect horizontal synchronization). Synchronization is achieved by including synchronization pulses in the horizontal and vertical flyback periods. They have a voltage level outside the normal picture data range to ensure that they are recognized as synchronization pulses.

As if that wasn't enough, the video amplifier, the part of the TV that alters the intensity of the spot as it travels across the screen, needs time to ensure that the flyback is invisible, so there are brief pauses between the end of the line and the start of the sync pulse, and again between the end of the sync pulse and the beginning of the data. This process is called *blanking*, and the delays are called the *front porch* (before the sync pulse) and the *back porch* (after the sync pulse). Figure 28-2 depicts a complete scan line.

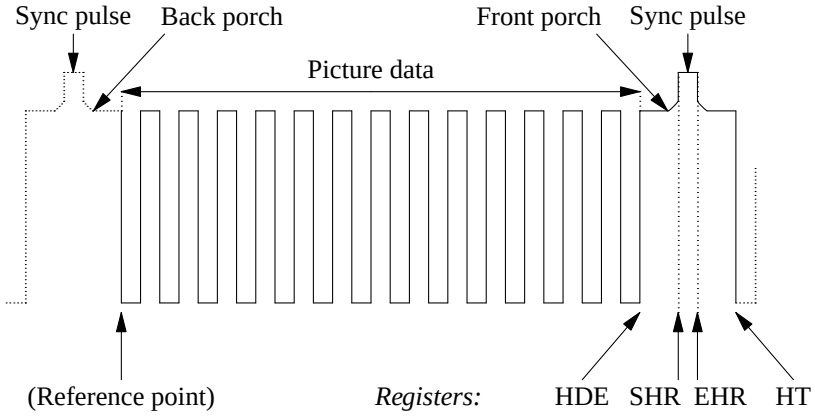

Figure 28-2: Scan line and register values

The register information at the bottom of the picture refers to the video controller registers. We'll look at how to interpret them on page 511.

That, in a nutshell, is how horizontal deflection works. Vertical deflection works in

almost the same way, just slower, with one minor exception. This basic display mechanism was developed for TVs in the 1930s, at a time when terms like high-tech (or even electronics) hadn't even been invented, and even today we're stuck with the low data rates that they decided upon in those days. Depending on the country, conventional TVs display only 25 or 30 frames (pages of display) per second. This would cause an unpleasant flicker in the display. This flicker is minimized with a trick called *interlacing*: instead of displaying the frame in one vertical scan, the odd and even lines are displayed in two alternating half frames, which doubles the apparent frame frequency.

How monitors differ from TVs

So how do we apply this to computer displays? Let's look at the US standard NTSC system—the international PAL and SECAM systems are almost identical except for the number of lines and a minor difference in the frequencies. NTSC specifies 525 lines, but that includes the vertical flyback time, and in fact only about 480 lines are visible. The aspect ratio of a normal TV is 4:3, in other words the screen is one-third wider than it is high, so if we want square pixels,[1] we need to have one-third more pixels per line. This means that we can display 640 pixels per line on 480 lines.[2] This resolution is normally abbreviated to ''640x480.'' PAL and SECAM have lower vertical frequencies, which allows a nominal 625 lines, of which about 600 are displayed. Either way, these values have two huge disadvantages: first, the resolution is barely acceptable for modern graphics displays, and secondly they are interlaced displays. Older PC display hardware, such as the CGA and some EGA modes, was capable of generating these signal frequencies, but normal graphic cards can no longer do it. Instead, dedicated TV output cards are available if that's what you want to do.

The first problem is interlace: it works reasonably for TVs, but it's a pain for computer displays—there's still more flicker than a real 50 Hz or 60 Hz display. Modern display boards can still run in interlace mode, but don't even think about doing so unless you're forced to—the resultant picture looks out of focus and is very tiring to read.

The second problem is the resolution: nowadays, 1024x768 is a minimum resolution, and some monitors display up to 2048x1536 pixels. On the other hand, even 60 Hz refresh rate is barely adequate: read any marketing literature and you'll discover that 72 Hz is the point at which flicker suddenly disappears. To get high-resolution, high refresh rate displays, you need some very high internal frequencies—we'll look at that further down.

How to fry your monitor

Remember that a monitor is just a glorified TV? Well, one of the design constraints of real TVs is that they have only a single horizontal frequency and only a single vertical frequency. This simplifies the hardware design considerably: the horizontal deflection uses a tuned circuit to create both the deflection frequency and the high voltage required to run the tube. This circuit is comprised of a transformer (the *line transformer*) and a condenser. Run a line transformer even fractionally off its intended frequency and it runs

1. A square pixel is one with the same height and width. They don't have to be that way, but it makes graphics software much simpler.
2. Does this look familiar?

much less efficiently and use more current, which gets converted to heat. If you run a conventional monitor off spec for any length of time, it will burn out the line transformer.

You don't have to roll your own X configuration to burn out the monitor: 20 years ago, the standard display boards were CGAs and HDAs,[1] and they had different line frequencies and thus required different monitors. Unfortunately, they both used the same data connector. If you connected an HDA (18.43 kHz line frequency) to a CGA monitor (15.75 kHz, the NTSC line frequency), you could expect smoke signals within a few minutes.

All modern PC monitors handle at least a range of line frequencies. This doesn't mean that an out of spec signal can't damage them—you might just burn out something else, frequently the power supply. Most better monitors recognize out-of-spec signals and refuse to try to display them; instead, you get an error display. Unfortunately, there are plenty of other monitors, especially older or cheaper models, which don't protect themselves against out of spec signals. In addition, just because the monitor displays correctly doesn't mean that it is running in spec. The moral of the story:

> ***Never run your monitor out of spec. If your display is messed up, there's a good chance that the frequencies are out, so turn off the monitor.***

Monitors aren't the only thing that you can burn out, of course. If you try hard, you can also burn out chips on some display boards by running them at frequencies that are out of spec. In practice, though, this doesn't happen nearly as often.

Another difference between TVs and monitors is the kind of signal they take. A real TV includes a receiver, of course, so you have an antenna connection, but modern TVs also have connections for inputs from VCRs, which are usually two audio signals and a video signal. The video signal contains five important components: the *red*, *green* and *blue* signals, and the horizontal and vertical sync pulses. This kind of signal is called *composite video*. By contrast, most modern monitors separate these signals onto separate signal lines, and older boards, such as the EGA, even used several lines per colour. Unfortunately, there is no complete agreement about how these signals should work: the polarity of the sync pulses can vary, and some boards cheat and supply the sync pulses on the green signal line. This is mainly of historical interest, but occasionally you'll come across a real bargain 20" monitor that only has three signal connections, and you may not be able to get it to work—this could be one of the reasons.

The CRT controller

The display controller, usually called a CRT (Cathode Ray Tube) controller, is the part of the display board that creates the signals we've just been talking about. Early display controllers were designed to produce signals that were compatible with TVs: they had to produce a signal with sync pulses, front and back porches, and picture data in between. Modern display controllers can do a lot more, but the principles remain the same.

1. Color Graphics Adapter and Hercules Display Adapter.

The first part of the display controller creates the framework we're looking for: the horizontal and vertical sync pulses, blanking and picture information, which is represented as a series of points or *dots*. To count, we need a pulse source, which also determines the duration of individual dots, so it is normally called a *dot clock*. For reasons lost in history, CRT controllers start counting at the top left of the display, and not at the vertical sync pulse, which is the real beginning of the display. To define a line to the horizontal deflection, we need to set four CRTC registers to tell it—see the diagram on page 509:

- The *Horizontal Display End* register (HDE) specifies how many dots we want on each line. After the CRTC has counted this many pixels, it stops outputting picture data to the display.
- The *Start Horizontal Retrace* register (SHR) specifies how many dot clock pulses occur before the sync pulse starts. The difference between the contents of this register and the contents of the HDE register defines the length of the front porch.
- The *End Horizontal Retrace* register (EHR) defines the end of the sync pulse. The width of the sync pulse is the difference between the contents of this register and the SHR register.
- The *Horizontal Total* register (HT) defines the total number of dot clocks per line. The width of the back porch is the difference between the contents of this register and the EHR register.

In addition, the *Start Horizontal Blanking* and *End Horizontal Blanking* registers (SHB and EHB) define when the video signals are turned off and on. The server sets these registers automatically, so we don't need to look at them in more detail.

The control of the vertical deflection is similar. In this case, the registers are *Vertical Display End* (VDE), *Start Vertical Retrace* (SVR), *End Vertical Retrace* (EVR), *Vertical Total* (VT), *Start Vertical Blanking* (SVB), and *End Vertical Blanking* (EVB). The values in these registers are counted in lines.

VGA hardware evolved out of older 8 bit character-based display hardware, which counted lines in characters, not dot clocks. As a result, all of these registers are 8 bits wide. This is adequate for character displays, but it's a problem when counting dots: the maximum value you can set in any of these registers is 255. The designers of the VGA resorted to a number of nasty kludges to get around this problem: the horizontal registers count in groups of 8 dot clocks, so they can represent up to 2048 dot clocks. The vertical registers overflow into an overflow register. Even so, the standard VGA can't count beyond 1024 lines. Super VGAs vary in how they handle this problem, but typically they add additional overflow bits. To give you an idea of how clean the VGA design is, consider the way the real Vertical Total (total number of lines on the display) is defined on a standard VGA. It's a 10 bit quantity, but the first 8 bits are in the VT register, the 9th bit is in bit 0 of the overflow register, and the 10th bit is in bit 5 of the overflow register.

The XF86Config mode line

One of the steps in setting up XFree86 is to define these register values. Fortunately, you don't have to worry about which bits to set in the overflow register: the mode lines count in dots, and it's up to the server to convert the dot count into something that the display board can understand. A typical Mode line looks like:

```
Modeline "640x480a" 28 640 680 728 776 480 480 482 494
```

These ten values are required. In addition, you may specify modifiers at the end of the line. The values are:

- A label for the resolution line. This must be enclosed in quotation marks, and is used to refer to the line from other parts of the *XF86Config* file. Traditionally, the label represents the resolution of the display mode, but it doesn't have to. In this example, the resolution really is 640x480, but the a at the end of the label is a clue that it's an alternative value.
- The clock frequency, 28 MHz in this example.
- The Horizontal Display End, which goes into the HDE register. This value and all that follow are specified in dots. The server mangles them as the display board requires and puts them in the corresponding CRTC register.
- The Start Horizontal Retrace (SHR) value.
- The End Horizontal Retrace (EHR) value.
- The Horizontal Total (HT) value.
- The Vertical Display End (VDE) value. This value and the three following are specified in lines.
- The Start Vertical Retrace (SVR) value.
- The End Vertical Retrace (EVR) value.
- The Vertical Total (VT) value.

This is pretty dry stuff. To make it easier to understand, let's look at how we would set a typical VGA display with 640x480 pixels. Sure, you can find values for this setup in any release of XFree86, but that doesn't mean that they're the optimum for *your system*. We want a non-flicker display, which we'll take to mean a vertical frequency of at least 72 Hz, and of course we don't want interlace. Our monitor can handle any horizontal frequency between 15 and 40 kHz: we want the least flicker, so we'll aim for 40 kHz.

First, we need to create our lines. They contain 640 pixels, two porches and a sync pulse. The only value we really know for sure is the number of pixels. How long should the porches and the sync pulses be? If you have a good monitor with good documentation, it should tell you, but most monitor manufacturers don't seem to believe in good documentation. When they do document the values, they vary significantly from monitor to monitor, and even from mode to mode: they're not as critical as they look. For example, here are some typical values from my NEC 5D handbook:

Horizontal sync pulse: 1 to 4 μs, front porch 0.18 to 2.1 μs, back porch 1.25 to 3.56 μs.

As we'll see, the proof of these timing parameters is in the display. If the display looks good, the parameters are OK. I don't know of any way to damage the monitor purely by modifying these parameters, but there are other good reasons to stick to this range. As a rule of thumb, if you set each of the three values to 2 μs to start with, you won't go too far wrong. Alternatively, you could start with the NTSC standard values: the standard specifies that the horizontal sync pulse lasts for 4.2 to 5.1 μs, the front porch must be at least 1.27 μs. NTSC doesn't define the length of the back porch—instead it defines the total line blanking, which lasts for 8.06 to 10.3 μs. For our purposes, we can consider the back porch to be the length of the total blanking minus the lengths of the front porch and the sync pulse. If you take values somewhere in the middle of the ranges, you get a front porch of 1.4 μs, a sync pulse of 4.5 μs, and total blanking 9 μs, which implies a back porch of 9 - 1.4 - 4.5 = 3.1 μs.

For our example, let's stick to 2 μs per value. We have a horizontal frequency of 40 kHz, or 25 μs per line. After taking off our 6 μs for flyback control, we have only 19 μs left for the display data. To get 640 pixels in this time, we need one pixel every 19 ÷ 640 μs, or about 30 ns. This corresponds to a frequency of 33.6 MHz. This is our desired dot clock.

The next question is: do we have a dot clock of this frequency? Maybe. This should be in your display board documentation, but I'll take a bet that it's not. Never mind, the XFree86 server is clever enough to figure this out for itself. At the moment, let's assume that you do have a dot clock of 33 MHz.

> If you don't have a suitable clock, you'll have to take the next lower clock frequency that you do have: you can't go any higher, since this example assumes the highest possible horizontal frequency.

You now need to calculate four register values to define the horizontal lines:

- The first value is the Horizontal Display End, the number of pixels on a line. We know this one: it's 640.
- You calculate SHR by adding the number of dot clocks that elapse during the front porch to the value of HDE. Recall that we decided on a front porch of 2 μs. In this time, a 33 MHz clock counts 66 cycles. So we add 66, right? Wrong. Remember that the VGA registers count in increments of 8 pixels, so we need to round the width of the front porch to a multiple of 8. In this case, we round it to 64, so we set SHR to 640 + 64 = 704.
- The next value we need is EHR, which is SHR plus the width of the horizontal retrace, again 64 dot clocks, so we set that to 704 + 64 = 768.
- The final horizontal value is HT. Again, we add the front porch—64 dot clocks—to EHR and get 768 + 64 = 832.

At this point, our vestigial mode line looks like:

```
Modeline "640x480"   28   640 704 768 832
```

Next, we need another four values to define the vertical scan. Again, of the four values we need, we only know the number of lines. How many lines do we use for the porches and the vertical sync? As we've seen, NTSC uses about 45 lines for the three combined, but modern monitors can get by with much less. Again referring to the Multisync manual, we get a front porch of betwwen 0.014 and 1.2 ms, a sync pulse of between 0.06 and 0.113 ms, and a back porch of between 0.54 and 1.88 ms. But how many lines is that?

To figure that out, we need to know our *real* horizontal frequency. We were aiming at 40 kHz, but we made a couple of tradeoffs along the way. The real horizontal frequency is the dot clock divided by the horizontal total, in this case 33 MHz ÷ 832, which gives us 39.66 kHz—not too bad. At that frequency, a line lasts 1÷39660 seconds, or just over 25 μs, so our front porch can range between ½ and 48 lines, our sync pulse between 2 and 5 lines, and the back porch between 10 and 75 lines. Do these timings make any sense? No, they don't—they're just values that the monitor can accept.

To get the highest refresh rate, we can go for the lowest value in each case. It's difficult to specify a value of ½, so we'll take a single line front porch. We'll take two lines of sync pulse and 10 lines of back porch. This gives us:

- VDE is 480.
- SVR is 481.
- EVR is 483.
- VT is 493.

Now our mode line is complete:

```
Modeline "640x480" 28  640 704 768 832  480 481 483 493
```

Now we can calculate our vertical frequency, which is the horizontal frequency divided by the Vertical Total, or 39.66 ÷ 493 kHz, which is 80.4 Hz—that's not bad either. By comparison, if you use the default value compiled into the server, you get a horizontal frequency of 31.5 kHz and a vertical frequency of only 60 Hz.

If you know the technical details of your monitor and display board, it really is that simple. This method doesn't require much thought, and it creates results that work.

Note that the resultant mode line may not work on other monitors. If you are using a laptop that you want to connect to different monitors or overhead display units, don't use this method. Stick to the standard frequencies supplied by the X server. Many overhead projectors understand only a very small number of frequencies, and the result of using a tweaked mode line is frequently that you can't synchronize with the display, or that it cuts off a large part of the image.

XF86Config

The main configuration file for XFree86 is called *XF86Config*. It has had a long and varied journey through the file system. At the time of writing, it's located at */usr/X11R6/lib/X11/XF86Config*, but previously it has been put in */etc/X11/XF86Config*, */etc/XF86Config* or */usr/X11R6/etc/X11/XF86Config*, and the server still looks for it in many of these places. If you're upgrading a system, you should ensure that you don't have old configuration files in one of the alternative places.

As we saw on page 102, there are a couple of ways to automatically create an *XF86Config* file. On that page we saw how to do it with *xf86cfg*. An alternative way is to run the X server in configuration mode:

```
# X -configure
XFree86 Version 4.2.0 / X Window System
(protocol Version 11, revision 0, vendor release 6600)
Release Date: 18 January 2002
        If the server is older than 6-12 months, or if your card is
        newer than the above date, look for a newer version before
        reporting problems.  (See http://www.XFree86.Org/)
Build Operating System: FreeBSD 5.0-CURRENT i386 [ELF]
Module Loader present
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/XFree86.0.log", Time: Sat Apr  6 13:51:10 2002
List of video drivers:
        atimisc
(the list is long, and will change; it's omitted here)
(++) Using config file: "/root/XF86Config.new"


Your XF86Config file is /root/XF86Config.new

To test the server, run 'XFree86 -xf86config /root/XF86Config.new'
```

Note that *X* does not place the resultant configuration file in the default location. The intention is that you should test it first and then move it to the final location when you're happy with it. As generated above, it's good enough to run XFree86, but you'll possibly want to change it. For example, it only gives you a single resolution, the highest it can find. In this section we'll look at the configuration file in more detail, and how to change it.

XF86Config is divided into several sections, as shown in Table 28-1. We'll look at them in the order they appear in the generated *XF86Config* file, which is not the same order as in the man page.

Table 28-1: XF86Config sections

Section	Description
ServerLayout	Describes the overall layout of the X configuration. X can handle more than one display card and monitor. This section is the key to the other sections
Files	Sets the default font and RGB paths.
ServerFlags	Set some global options.
Module	Describes the software modules to load for the configuration.
InputDevice	Sets up keyboards, mice and other input devices.
Monitor	Describes your monitor to the server.
Device	Describes your video hardware to the server.
Screen	Describes how to use the monitor and video hardware.

The server layout

The ServerLayout section describes the relationships between the individual hardware components under the control of an X server. For typical hardware, `X -configure` might generate:

```
Section "ServerLayout"
        Identifier     "XFree86 Configured"
        Screen      0  "Screen0" 0 0
        InputDevice    "Mouse0" "CorePointer"
        InputDevice    "Keyboard0" "CoreKeyboard"
EndSection
```

This shows that the server has one screen and two input devices. The names `Mouse0` and `Keyboard0` suggest that they're a mouse and a keyboard, but any name is valid. These entries are pointers to sections elsewhere in the file, which must contain definitions for `Screen0`, `Mouse0` and `Keyboard0`.

Normally you only have one screen, one mouse and one keyboard, so this section might seem rather unnecessary. As we will see when we look at multiple monitor configurations, it's quite important to be able to describe these relationships.

The Files section

The *Files* section of the *XF86Config* file contains the path to the RGB database file, which should never need to be changed, and the default font path. You may want to add more font paths, and some ports do so: the `FontPath` lines in your *XF86Config* are concatenated to form a search path. Ensure that each directory listed exists and is a valid font directory.

The standard *Files* section looks like:

```
Section "Files"
        RgbPath      "/usr/X11R6/lib/X11/rgb"
        ModulePath   "/usr/X11R6/lib/modules"
        FontPath     "/usr/X11R6/lib/X11/fonts/misc/"
        FontPath     "/usr/X11R6/lib/X11/fonts/Speedo/"
        FontPath     "/usr/X11R6/lib/X11/fonts/Type1/"
        FontPath     "/usr/X11R6/lib/X11/fonts/CID/"
        FontPath     "/usr/X11R6/lib/X11/fonts/75dpi/"
        FontPath     "/usr/X11R6/lib/X11/fonts/100dpi/"
EndSection
```

If you are running a high-resolution display, this sequence may be sub-optimal. For example, a 21" monitor running at 1600x1200 pixels has a visible display of approximately 16" wide and 12" high, exactly 100 *dpi* (*dots per inch*, really pixels per inch). As a result, you'll probably be happier with the 100 dpi fonts. You can change this by swapping the last two lines in the section:

```
        FontPath     "/usr/X11R6/lib/X11/fonts/100dpi/"
        FontPath     "/usr/X11R6/lib/X11/fonts/75dpi/"
EndSection
```

Don't just remove the 75 dpi fonts: some fonts may be available only in the 75 dpi directory.

Sometimes the server complains:

```
Can't open default font 'fixed'
```

This is almost certainly the result of an invalid entry in your font path. Try running *mkfontdir* in each directory if you are certain that each one is correct. The *XF86Config* man page describes other parameters that may be in this section of the file.

The ServerFlags section

The ServerFlags section allows you to specify a number of global options. By default it is not present, and you will probably not find any reason to set it. See the man page *XF86Config(5)* for details of the options.

The Module section

The `Module` section describes binary modules that the server loads:

```
Section "Module"
        Load  "extmod"
        Load  "xie"
        Load  "pex5"
        Load  "glx"
        Load  "GLcore"
        Load  "dbe"
        Load  "record"
        Load  "type1"
EndSection
```

We won't look at modules in more detail; see the XFree86 documentation.

The InputDevice section

The *InputDevice* section specifies each input device, typically mice and keyboards. Older versions of XFree86 had separate `Mouse` and `Keyboard` sections to describe these details. The default `XF86Config` looks something like this:

```
Section "InputDevice"
        Identifier  "Keyboard0"
        Driver      "keyboard"
EndSection

Section "InputDevice"
        Identifier  "Mouse0"
        Driver      "mouse"
        Option      "Protocol" "auto"
        Option      "Device" "/dev/mouse"
EndSection
```

There's not much to be said for the keyboard. Previous versions of XFree86 allowed you to set things like *NumLock* handling and repeat rate, but the former is no longer needed, and the latter is easier to handle with the *xset* program.

Mice are still not as standardized as keyboards, so you still need a `Protocol` line and a device name. The defaults shown here are correct for most modern mice; the mouse driver can detect the mouse type correctly. If you're using the mouse daemon, *moused*, you need to change this entry: *moused* opens */dev/mouse* exclusively. Use the *moused* device, */dev/sysmouse*, instead.

If you're using a serial mouse or one with only two buttons, *and* if you're not using *moused*, you need to change the device entries and specify the `Emulate3Buttons` option. That's all described in the man page, but in general it's easier to use *moused*.

The Monitor section

Next comes the description of the monitor. Modern monitors can identify themselves to the system. In that case, you get a section that looks like this:

```
Section "Monitor"
        Identifier   "Monitor0"
        VendorName   "IBM"
        ModelName    "P260"
        HorizSync    30.0 - 121.0
        VertRefresh  48.0 - 160.0
```

This tells the server that the monitor is an IBM P260, that it can handle horizontal frequencies between 30 kHz and 121 kHz, and vertical frequencies between 48 Hz and 160 Hz. Less sophisticated monitors don't supply this information, so you might end up with an entry like this:

```
Section "Monitor"
        Identifier   "Monitor0"
        VendorName   "Monitor Vendor"
        ModelName    "Monitor Model"
EndSection
```

This may seem like no information at all, but in fact it does give the identifier. Before you use it, you should add at least the horizontal and vertical frequency range, otherwise the server assumes it's a standard (and obsolete) VGA monitor capable of only 640x480 resolution.

This is also the place where you can add mode lines. For example, if you have created a mode line as described in the first part of this chapter, you should add it here:

```
Section "Monitor"
        Identifier   "right"
        VendorName   "iiyama"
        ModelName    "8221T"
        HorizSync    24.8 - 94.0
        VertRefresh  50.0 - 160.0

ModeLine  "640x480"    73   640  672  768  864    480  488  494  530
# 62 Hz!
ModeLine  "800x600"   111   800  864  928 1088    600  604  610  640
# 143 Hz
ModeLine  "1024x768"  165  1024 1056 1248 1440    768  771  781  802
# 96 Hz
ModeLine  "1280x1024" 195  1280 1312 1440 1696   1024 1031 1046 1072 -hsync -vsync
# 76 Hz
ModeLine  "1600x1200" 195  1600 1616 1808 2080   1200 1204 1207 1244 +hsync +vsync
# 56 Hz!
ModeLine  "1920x1440" 200  1920 1947 2047 2396   1440 1441 1444 1483 -hsync +vsync
# 61 Hz
ModeLine  "1920x1440" 220  1920 1947 2047 2448   1440 1441 1444 1483 -hsync +vsync
        EndSection
```

It's possible to have multiple mode lines for a single frequency, and this even makes sense. The examples for 1920x1440 above have different pixel clocks. If you use this monitor with a card with a pixel clock that only goes up to 200 MHz, the server chooses the first mode line. If you use a card with up to 250 MHz pixel clock, it uses the second and gets a better page refresh rate.

The X server has a number of built-in mode lines, so it's quite possible to have a configuration file with no mode lines at all. The names correspond to the resolutions, and there can be multiple mode lines with the same name. The server chooses the mode line with the highest frequency compatible with the hardware.

The Device section

The *Device* section describes the video display board:

```
Section "Device"
        ### Available Driver options are:-
        ### Values: <i>: integer, <f>: float, <bool>: "True"/"False",
        ### <string>: "String", <freq>: "<f> Hz/kHz/MHz"
        ### [arg]: arg optional
        #Option     "SWcursor"                # [<bool>]
        #Option     "HWcursor"                # [<bool>]
```

```
        #Option     "PciRetry"                  # [<bool>]
        #Option     "SyncOnGreen"               # [<bool>]
        #Option     "NoAccel"                   # [<bool>]
        #Option     "ShowCache"                 # [<bool>]
        #Option     "Overlay"                   # [<str>]
        #Option     "MGASDRAM"                  # [<bool>]
        #Option     "ShadowFB"                  # [<bool>]
        #Option     "UseFBDev"                  # [<bool>]
        #Option     "ColorKey"                  # <i>
        #Option     "SetMclk"                   # <freq>
        #Option     "OverclockMem"              # [<bool>]
        #Option     "VideoKey"                  # <i>
        #Option     "Rotate"                    # [<str>]
        #Option     "TexturedVideo"             # [<bool>]
        #Option     "Crtc2Half"                 # [<bool>]
        #Option     "Crtc2Ram"                  # <i>
        #Option     "Int10"                     # [<bool>]
        #Option     "AGPMode"                   # <i>
        #Option     "DigitalScreen"             # [<bool>]
        #Option     "TV"                        # [<bool>]
        #Option     "TVStandard"                # [<str>]
        #Option     "CableType"                 # [<str>]
        #Option     "NoHal"                     # [<bool>]
        #Option     "SwappedHead"               # [<bool>]
        #Option     "DRI"                       # [<bool>]
        Identifier  "Card0"
        Driver      "mga"
        VendorName  "Matrox"
        BoardName   "MGA G200 AGP"
        BusID       "PCI:1:0:0"
EndSection
```

This example shows a Matrox G200 AGP display board. It includes a number of options that you can set by removing the comment character (#). Many of these options are board dependent, and none of them are required. See the X documentation for more details.

Note particularly the last line, `BusID`. This is a hardware-related address that tells the X server where to find the display board. If you move the board to a different PCI slot, the address will probably change, and you will need to re-run *X -configure* to find the new bus ID.

If your display board is older, much of this information will not be available, and you'll have to add it yourself. Unlike older monitors, it's hardly worth worrying about older boards, though: modern boards have become extremely cheap, and they're so much faster than older boards that it's not worth the trouble.

The Screen section

The final section is the Screen section, which describes the display on a monitor. The default looks something like this:

```
Section "Screen"
        Identifier "Screen0"
        Device     "Card0"
        Monitor    "Monitor0"
        SubSection "Display"
                Depth     1
        EndSubSection
        SubSection "Display"
```

```
                Depth     4
        EndSubSection
        SubSection "Display"
                Depth     8
        EndSubSection
        SubSection "Display"
                Depth     15
        EndSubSection
        SubSection "Display"
                Depth     16
        EndSubSection
        SubSection "Display"
                Depth     24
        EndSubSection
EndSection
```

The first three lines describe the relationship between the screen display, the video board that creates it, and the monitor on which it is displayed. Next come a number of subsections describing the possible bit depths that the screen may have. For each display depth, you can specify which mode lines you wish to use. Modern display hardware has plenty of memory, so you'll probably not want to restrict the display depth. On the other hand, you may want to have multiple mode lines. Your display card and monitor are good enough to display 2048x1536 at 24 bits per pixel, but occasionally you'll get images (in badly designed web pages, for example) so miniscule that you'll want to zoom in, maybe going all the way back to 640x480 in extreme cases. You can toggle through the available resolutions with the key combinations **Ctrl**-**Alt**-**Numeric +** and **Ctrl**-**Alt**-**Numeric -**. You're probably not interested in pixel depths lower than 640x480, so your Screen section might look like:

```
Section "Screen"
        Identifier "Screen0"
        Device     "Card0"
        Monitor    "Monitor0"
        DefaultDepth    24
        SubSection "Display"
                Depth     24
                Modes     "2048x1536" "1600x1200"  "1024x768"  "640x480"
        EndSubSection
EndSection
```

This section includes a `DefaultDepth` entry for the sake of example. In this case, it's not strictly needed, because there's only one pixel depth. If there were more than one Display subsection, it would tell *xinit* which depth to use by default.

Multiple monitors and servers

We've seen above that X provides for more than one monitor per server. If you have multiple display cards and monitors, let the server generate the *XF86Config* file: it generates a file that supports all identified devices. The resultant server layout section might look like this:

```
Section "ServerLayout"
        Identifier     "XFree86 Configured"
        Screen      0  "Screen0" 0 0
        Screen      1  "Screen1" RightOf "Screen0"
        Screen      2  "Screen2" RightOf "Screen1"
        InputDevice    "Mouse0" "CorePointer"
        InputDevice    "Keyboard0" "CoreKeyboard"
EndSection
```

The file will also have multiple monitor, device and screen sections. The server can't know about the real physical layout of the screen, of course, so you may have to change the ordering of the screens. When you run the server without any other specifications, it is assigned server number 0, so these screens will be numbered *:0.0*, *:0.1* and *:0.2*.

Multiple servers

It's also possible to run more than one X server on a single system, even if it only has a single monitor. There can be some good reasons for this: you may share a system amongst your family members, so each of them can have their own server. Alternatively, you may have a laptop with a high-resolution display and need to do a presentation on overhead projectors that can't handle more than 1024x768 pixels. It's not practical to simply switch to a lower resolution, because the overall screen size doesn't change, and it's difficult to avoid sliding the image around when you move the cursor.

For each server, you require one virtual terminal—see page 109 for more details. If you're using the same hardware, you can also use the same *XF86Config* file. The only difference is in the way in which you start the server. For example, you could start three X servers, one with the *fvwm2* window manager, one with *KDE* and one with *GNOME*, with the following script:

```
xinit &
xinit .xinitrc-kde -- :1 &
xinit .xinitrc-gnome -- :2 -xf86config XF86Config.1024x768 &
```

Due to different command line options, you must use *xinit* here, and not *startx*. The first *xinit* starts a server with the default options: it reads its commands from *.xinitrc*, it has the server number 0, and it reads its configuration from the default *XF86Config* file. The second server reads its commands from *.xinitrc-kde*, it has the server number 1, and it reads its configuration from the default *XF86Config* file. The third server reads its commands from *.xinitrc-gnome*, it has the server number 2, and the configuration file is *XF86Config.1024x768*. Assuming that you reserve virtual terminals */dev/ttyv7*, */dev/ttyv8* and */dev/ttyv9* for the servers, you can switch between them with the key combinations **Ctrl-Alt-F8**, **Ctrl-Alt-F9** and **Ctrl-Alt-F10**.

X in the network

X is a network protocol. So far we have looked at the server. The clients are the individual programs, such as *xterm*, *emacs* or a web browser, and they don't have to be on the same machine. A special notation exists to address X servers and screens:

System name`:`*server number*`.`*screen number*

When looking at X client-server interaction, remember that the server is the software component that manages the display. This means that you're always sitting at the server, not at the client. For example, if you want to start an *xterm* client on *freebie* and display it on *presto*, you'll be sitting at *presto*. To do this, you could type in, on *presto*,

```
$ ssh freebie xterm -ls -display presto:0 &
```

The flag `-ls` tells *xterm* that this is a *login shell*, which causes it to read in the startup files.

For this to work, you must tell the X server to allow the connection. There are two things to do:

- Use *xhost* to specify the names of the systems that have access:

  ```
  $ xhost freebie presto bumble wait gw
  ```

  This enables access from all the systems on our reference network, including the one on which it is run. You don't need to include your own system, which is enabled by default, but if you do, you can use the same script on all systems on the network.

- *xhost* is not a very robust system, so by default *startx* starts X with the option `-nolisten tcp`. This completely blocks access from other systems. If you want to allow remote clients to access your X server, modify */usr/X11R6/bin/startx*, which contains the text:

  ```
  listen_tcp="-nolisten tcp"
  ```

  Change this line to read:

  ```
  listen_tcp=
  ```

  This enables remote connections the next time you start the server.

Multiple monitors across multiple servers

We saw above that a server can handle multiple monitors, and a system can handle multiple servers. One problem with multiple monitors is that most computers can only handle a small number of display boards: a single AGP board and possibly a number of PCI boards. But PCI boards are difficult to find nowadays, and they're slower and have less memory.

If you have a number machines physically next to each other, you have the alternative of running X on each of them and controlling everything from one keyboard and mouse. You do this with the *x11/x2x* port. For example: *freebie*, *presto* and *bumble* have monitors next to each other, and *presto* has two monitors. From left to right they are *freebie:0.0*, *presto:0.0*, *presto:0.1* and *bumble:0.0*. The keyboard and mouse are connected to *presto*. To incorporate *freebie:0.0* and *bumble:0.0* in the group, enter these commands on *presto*:

```
$ DISPLAY=:0.0 x2x -west -to freebie:0 &
$ DISPLAY=:0.1 x2x -east -to bumble:0 &
```

After this, you can move to the other machines by moving the mouse in the corresponding direction. It's not possible to continue to a further machine, but it is possible to connect in other directions (*north* and *south*) from each monitor on *presto*, which in this case would allow connections to at least six other machines. Before that limitation becomes a problem, you need to find space for all the monitors.

Stopping X

To stop X, press the key combination **Ctrl**-**Alt**-**Backspace**, which is deliberately chosen to resemble the key combination **Ctrl**-**Alt**-**Delete** used to reboot the machine. **Ctrl**-**Alt**-**Backspace** stops X and returns you to the virtual terminal in which you started it. If you run from *xdm*, it redisplays a login screen.