The X Window User HOWTO

Table of Contents

Table of Contents

The X Window User HOWTO

by Ray Brigleb, `ray@aracnet.com`

v2.0, 1 September 1999

This document contains information on configuring the X Window environment for the Linux user, as well as for the beginning system administrator attempting to sort through the many configuration options and details of X Window. A basic knowledge of software configuration and installation is assumed, as is the presence of X on the users system.

1.Introduction

2.Getting Started

3.Choosing a Window Manager

4.Working In X

5.X Startup

6.Configuring the Window Manager

7.Fonts and Colors

8.The X Resources

9.Clients and Application Tips

10.Advanced X Usage

11.Bibliography and Other Resources

1.Introduction

The *X Window System* is an advanced, network transparent, windowing, graphical environment, developed at the *Massachusetts Institute of Technology*, and first released in 1984. This document assumes that you have installed X and it is functional. We intend to learn how to use X productively, not so much how to compile the programs; most Linux distributions come with X as an option during installation, compiled and ready to go.

1.1 Other Sources Of Information

If you are just starting out, you may find the *XFree86 HOWTO* and *XFree86 Video Timings HOWTO*s to be more helpful, and you should be able to find that in the same place you found this. At the end of the document you will also find a Bibliography And Resources section, to find even more information. Oh, and don't forget to read the **man pages**.

1.2 Versions Of This Document

New versions of this HOWTO may be periodically posted to comp.os.linux.help They will also be uploaded to various Linux WWW and FTP sites, including Linux Documentation Project web site. The latest version can usually be found at http://www.croftj.net/~ray/howto, but the Metalab repository is the canonical distribution point.

Changes in version 1.1 include more coverage of **rxvt**, expanded coverage of X Resources, more trivial history of X, corrected references to SunSite to point to MetaLab, and instructions on using Type 1 fonts with X.

Changes in version 1.2 include much−requested coverage of using TrueType fonts in X, more details about setting up `xdm`, and a few more added resources. There is also a correction to my discussion of screen blanking features (thanks to Heinrich Langos for the correction!).

Changes in version 1.3 include some fixes, info for corrupted font paths in some recent distributions, and lots more info about basic X security. I've also added some tips about KDE and an e−mail address update.

Changes in version 1.4 include corrections and additions from Anthony J., and some very good security tips from Tomasz Motylewski.

Changes in this version (2.0) includes corrections from Guus Bosch, Brian J. Miller, and myself, as well as lots of new updates and info, and a plea for a new maintainer!

1.3 Feedback And Corrections

If you have questions or comments about this document, please feel free to email Ray Brigleb at my current (`ray@aracnet.com`) or ``permanent'' (`rayola@mindless.com`) address. I readily welcome any suggestions or criticisms. If you find any mistakes with this document, please let me know so I can correct them in the next revision. If you have information you would like to see in future revisions, or you would like to contribute to a future revision, drop me a line. I'm also looking for more resources to add to the sections and bibliography. While we're at it, I'm also looking for someone interested in taking over maintenance of this document, please let me know if you're interested. I think another maintainer might be able to breathe life into this project that I have not had as of late, and judging from the feedback I receive this is still a very viable and useful text.

1.4 Acknowledgments

A special thanks to the HOWTO coordinator Tim Bynum for help and suggestions, and the XFree86 Group for the man pages and help files that come with XFree86. I really need to thank all of those who have written documentation and descriptions of their offerings in the past, I have just gleaned a selection of the most relevant items from their material for this document. Also, thanks are due to Matt Chapman, author of the Window Managers Guide website, and Scott Scriven, for the FVWM2 Themes tips and tricks I stole from some of his hacks. Thanks are also due to Joe Croft for the Internet service, invaluable to the ongoing development of this HOWTO, and Claire Galper, for moral support and miscellaneous tips. Last but not least, thanks to the whole GNU/Linux community, for everything, and Mr. Stallman, for this text editor (and philosophy).

1.5 Copyright

Copyright (c) 1998, 1999 by Ray Brigleb.

Unless otherwise stated, Linux HOWTO documents are copyrighted by their respective authors. Linux HOWTO documents may be reproduced and distributed in whole or in part, in any medium physical or electronic, as long as this copyright notice is retained on all copies. Commercial redistribution is allowed and encouraged; however, the author would like to be notified of any such distributions.

All translations, derivative works, or aggregate works incorporating any Linux HOWTO documents must be covered under this copyright notice. That is, you may not produce a derivative work from a HOWTO and impose additional restrictions on its distribution. Exceptions to these rules may be granted under certain conditions; please contact the Linux HOWTO coordinator for more information.

In short, we wish to promote dissemination of this information through as many channels as possible. However, we do wish to retain copyright on the HOWTO documents, and would very much like to be notified of any plans to redistribute the HOWTOs, this one in particular!

Many of the terms mentioned in this document are trade names. Unless otherwise stated, all trademarks are property of their respective owners.

2.Getting Started

2.1 The X Window System: History and Architecture

The *X Window System* was developed in the Laboratory for Computer Science at the *Massachusetts Institute of Technology*, as part of project Athena in cooperation with DEC, and first released in 1984. The project lead of the main development was Robert Scheifler, and the origins of X owe much debt to the ``*W''* Windowing package, developed by Paul Asente at *Stanford*. In September of 1987, MIT issued the first release of the X11 that we know and use today. As of X11R2, control passed from MIT to the *X Consortium*, formed in January of 1988.

Many of the ideas that went into X Window also came from research at Xerox Corporation's *Palo Alto Research Center* (PARC), where they were working on computers like the Parc and the Star in the late seventies. None of these computers made it to market, but when Xerox demonstrated a window system custom built to run *Smalltalk 80*, people were hooked. These series of three computers demonstrated the WIMP (Windows, Icons, Menus, Pointer) interface so well that it spawned a revolution in computing almost overnight. Within a few years many computer users got a taste of a windowing system of some kind, and you might say they never looked back.

X Window is currently developed and distributed by the X Consortium, however, a liberal license permits the existence of free and low−cost implementations. The version of X used on Linux is XFree86. XFree86 is a collection of X servers for UNIX−like OSs on Intel x86 platforms. The work is derived from X386, and much of it is contributed back into X11R6 thereafter. We can think of XFree86, for all intents and purposes, to be X Window for Linux, unless you have purchased another X server.

X Window is built upon a great many toolkits, or libraries. It is built upon the X Toolkit Intrinsics and the Athena Widgets. Many programs use XView or Motif tools. More still are part of a newer, unified windowing and communication system, like GNOME or KDE. If you find many of your programs not compiling, or are getting strange and inexplicable errors, you may wish to make sure that you have installed X correctly, because most of these libraries (with the exception of Motif) are free and most likely preinstalled with X on your system.

And remember, it's called X Window, *not* X Windows!

2.2 Anatomy of Your Desktop

There are a few basic principles and terms you should familiarize yourself with to make using X much more straightforward. These terms will appear over and over again in the manual pages and help files, which it is suggested that you consult whenever necessary.

The **screen** is your whole ``desktop'', and the words may be used interchangeably. Technically it means the primary video display you view X with, and you can have more than one screen, in fact you can have more than one computer running off a single X server. This is beyond the scope of this humble document, but you should be aware of the distinction.

The **root window** is the background of your screen. It is referred to a window in name alone, it does not behave like any other window, but rather you run your applications on the root window, or put a picture on it, or just a solid color.

The **window manager** is the main interface between the X Window system and the user. Without the window manager, the system would be rather difficult to use, and would certainly not be a very productive tool. The window manager provides such functionality as window borders, menus, icons, virtual desktops, button bars, tool bars, and allows the user to customize it at will, often adding to its functionality in the process.

The **pointer** is the arrow or indicator of any given shape which represents the location your mouse (or other pointing device) corresponds to on the screen. The pointer often changes to give you contextual feedback as to what will happen when you use the mouse at that point on the screen.

The **window** is a frame in which any given application resides which is ``managed'' by the window manager. This includes pretty much anything except the so−called root window. Even windows which do not appear to have frames, titles, or normal borders of any kind are being managed by your window manger. The **active window** is the window you are currently using, the window that will receive text when you type, and is traditionally denoted by the fact that your mouse cursor is pointing at it, though this is not always the case. The active window is said to have ``focus,'' the rest of the windows on your display being ``unfocused.''

Menus and **icons** behave in X similar to the way they behave in other windowing systems, and the same general principles apply. Windows with text only are called **terminal emulators**, an example would be xterm, and these basically emulate a console text−only display, but let you multiplex and use more than one at a time, and have many other advantages available due to their being used in X. We shall discuss many of these later on.

2.3 Invoking X Window

Starting X can be done in several ways. On your system it may be set up to start automatically, and you will not need to read this section. Most Linux systems, however, presently start at the command line upon login, and you have to decipher this for yourself.

The most basic way to start X is with **xinit**. This will put you at a blank desktop, by default, and with no window manager loaded. If no client program is specified on the command line, **xinit** will look for the *.xinitrc* file to run as a shell script, to start up client programs. If this file does not exist, **xinit** will use the following command as a default:

```
        xterm -geometry +1+1 -n login -display :0
```

As you see, this is not very helpful. The most common way to start X is with the command **startx**. This is the most civilized fashion to start the windowing system, but requires that you log in from a text shell, and start the windowing system yourself. For many Linux users this is the most common way to start X, it is also the most flexible. You can issue commands such as the following:

```
        startx -- -bpp 8    #start x in 256 color mode
        startx -- -bpp 32   #start x in true color mode
```

The double dashes pass arguments directly to xinit, and this way you can start up X in the resolution your work will require, and still have it use the configuration files we will cover later in this document.

2.4 The X Display Manager

The program **xdm** provides similar services to **getty** and **login**, which allow users to log into a system and start their basic shell. If you start X with **xdm**, however, users need only to type in their username and password at a friendly prompt, and they are dropped directly into the graphical environment. This is simple and easy to use, and is seen frequently in college campuses, cyber cafes, business environments, anywhere you have users not necessarily familiar with Unix to any great extent.

xdm can be configured with configuration files located in */usr/X11R6/lib/X11/xdm* on your Linux system. The file `xdm-config` is for configuring how the login screen appears to users, and `Xsetup_0` is used to tell xdm what programs should be launched when X is started. Some of the configuration a normal user would put in their `.xinitrc` file should go in here, if xdm is to be normally used.

Here is a sample `Xsetup_0` file to look at, which might help to configure your system. The `xfstt` program is the TrueType font server, and is discussed later in this document. Also, notice that we're using a shell script here, and it's trying to call **xv** to set the background to a nice picture (instead of the boring black and white background pattern), and if that call fails, **xsetroot** is called, to at least try to set the background to a nice blue color.

```
#!/bin/sh
xconsole -geometry 480x100-0-0 -daemon -notify -verbose -fn \
'-schumacher-clean-medium-r-*-*-10-*-*-*-*-*-*-*' -exitOnFail
/usr/X11R6/bin/xfstt &
/usr/X11R6/bin/xv -quit -root                                \
/usr/local/share/WindowMaker/Backgrounds/InDreams.jpg        \
|| xsetroot -solid darkblue
xset fp+ unix/:7100
```

Many distributions of Linux include this facility automatically. If you are able to look at the run levels for your Linux system, you can probably see that run level 3 is the normal startup (look under `/etc/rc.d`) unless you're booting into something like *xdm*. If you are, you're going to be starting in run level 5. This is something of a standard on Linux (and similar) systems. Programs such as *linuxconf* and its functional equivalents should be able to adjust this.

Also on many newer Linux systems are facilities such as *gdm* and *kdm*, which are GNOME and KDE aware equivalents of this program. Primarily, this just changes the look and feel to suit your desktop preferance, but often these other versions contain more features, such as remembering which desktop environment you last logged in to, the ability to shutdown and reboot from the console, and so forth.

3.Choosing a Window Manager

This is a discussion of the window managers available to the X Window user, with a comparison of some finer ones to choose from. In order to prevent this document from becoming overwhelmingly large (and proportionally difficult to maintain, as well) I am limiting the discussion to the more popular and well known window managers out there. I believe these are most representative for our purposes, and once you grasp the basic concepts of a few, you more or less will have the hang of them all.

Your choice of window manager can dramatically influence how pleasant your computing experience will be. You spend much of your time dealing with windows when you're in X, and you don't want something that's too obtrusive, one that will get in your way. Some window managers are extremely customizable, to the point that you can pretty much feel like it's a new operating system. If you don't like icons, get rid of them! If you miss that toolbar, you can build a better one, and it can be a pretty painless and rewarding process besides.

3.1 FVWM And Its Ancestors

The most common window managers nowadays have their roots in Robert Nation's FVWM window manager. These include FVWM, FVWM2, FVWM95, and many more. FVWM itself is derived partly from TWM, which comes with an X Window installation, but which we will not cover here. The syntax and usage of TWM is similar to FVWM, but it actually uses more memory, and feels clumsy and awkward to most users.

FVWM is the original and old standby favorite Linux window manager. Now that the 2.0 of versions of FVWM have become stable and in more general usage, this newer version is preferred, for the syntax is much more direct and simplified, and much more flexible as well. However, many people still have the original FVWM on their systems, as it has been a long time favorite and standard, and most of the examples in this text are equally applicable to this older version. And if the older one is working fine for you, there may be no need to upgrade, since it may break your configuration files to some degree, for some of the syntax has indeed changed, and is not altogether backward compatible. Such is the nature of progress.

Nobody seems quite sure what F in FVWM stands for (not even the author, from what I can tell!), but the VWM would seem to (correctly) indicate Virtual Window Manager, and indeed the FVWM series are virtual window managers, and that is one of their strengths. You can bind keys to any function, including the switching between the virtual desktops, and do this with ease.

FVWM2 is the new standard, including many improvements and features not found in the previous version. Unlike many newer window managers, it works just fine on 8−bit, 256 color displays, which I am using at the moment, and still can be improved with little icons and gradients, to look as much like the Win98 interface as you may or may not want. This version is also much more extensible in general, and allows the use of themes and dynamic menus. Technically, FVWM2 is still in beta, but it works just great, and I have not yet had a single problem with it.

Resources:

The official FVWM and FVWM2 site is http://www.hpc.uh.edu/fvwm/. The latest version should always be available at ftp://ftp.hpc.uh.edu/pub/fvwm/.

3.2 The Wide World of Window Systems

For users more familiar with other window environments on other operating systems, there are plenty of window managers to emulate the desktop you're accustomed to.

The icewm window manager is aimed at a more consistent look and feel, and can emulate many other windowing system standards, hot key bindings, and such. The OS/2 emulations are particularly nice. AmiWM is an Amiga Workbench type window manager, mlvwm is a MacOS emulator. A nice addition to these window managers is dfm, a desktop and file manager that pleasantly resembles the OS/2 Workplace Shell.

Resources:

The Window Managers website can be found at http://www.PLiG.org/xwinman/, and is a consistently good resource for finding out about the variety of window managers available, particularly for finding good window managers to emulate other windowing environments. Many more still are listed at http://www.PLiG.org/xwinman/others.html, provided by the same author as the previous link, and this is quite a formidable and complete list indeed. You can download Linux versions of most popular window managers at the metalab archive.

3.3 The X Graphical Interfaces

Open Look and Motif were early attempts to standardize X Window, and make it usable in more mainstream environments, around a greater variety of workstations. Both were somewhat successful at those attempts (in look and feel, if not politically) and can be used under a modern Linux system. Both require specific libraries, which can be used to compile a variety of applications specifically designed for one or the other environment, to give all programs a similar look and feel.

Starting with X11 Release 5, Sun Microsystem's **OpenWindows** graphical interface was available to X users. This includes two versions of the Open Look Window Manager, *olwm* and *olvwm* (with a virtual desktop). This was developed by Sun in hopes of making a standard windowing environment, and indeed it was standard with Solaris for many years. If your distribution of Linux did not come with this windowing system installed, you must remember to install the *XView* libraries to get it to work, and you will also have to put the */usr/openwin/...* directories in your search path.

If you want the actual **OSF/Motif Toolkit** for Linux, you'll have to pay, the programs and toolkit are not free. However, the *Hungry Programmers* have written **LessTif**, which allows you to compile Motif programs more or less as if you owned to toolkit. LessTif is a clone of the Motif toolkit. Currently LessTif is partially implemented with most of the API in place. Many programs already work under this free version of the toolkit, and it even comes with a window manager, derived from FVWM code, which you'd swear was the Motif Window Manager.

The most useful feature of this toolkit, however, is compiling programs dependent on having a Motif library on your system. The window manager is nothing spectacular, and mostly useful if you're migrating from the original Motif, and want to keep your configuration file. For all intents and purposes, you will find FVWM much more feature−filled and useful, and it looks and behaves almost identically, even recognizing the window hints supplied by programs built with the Motif toolkit.

Resources:

The Hungry Programmers LessTif can be found at http://www.lesstif.org/.

3.4 The X Desktop Environments

The second generation of Linux window managers was brought about by KDE, and soon joined by GNOME. There are some striking similarities, and some great differences, between these two, and I will attempt to cover them here. The most important thing to remember at this point is that neither of them is in any way a complete product. Both are at the start of a long development cycle, and not completely stable yet, and as such are not suited to mission−critical work at this time.

The KDE Desktop Environment

(A quote from the home page:) ``*KDE is a completely new desktop, incorporating a large suite of applications for Unix workstations. While KDE includes a window manager, file manager, panel, control center and many other components that one would expect to be part of a contemporary desktop environment, the true strength of this exceptional environment lies in the interoperability of its components."*

The KDE Desktop Environment is an attempt to make a desktop environment, not just a window manager. The tools of KDE work together so well, for instance, one might be fooled into thinking KDE was an entire operating system. All the tools to work in a windowing system are included, and many more have been ported to the KDE environment. KDE has achieved a surprising level of maturity already, but many are reluctant to install it on their desktop, because of the licensing stipulations of the QT toolkit, upon which KDE is based. This has changed a little lately, and the licence now qualifies as Open Source by definition, but is not the same as that of GNU software.

For those just looking to get down to business, KDE is often the way to go. This project has been around for some time now, and has let go of some flashiness for the ability to get lots done. In many ways you can become fooled into thinking you're using a product strikingly similar to the good parts of Windows. Which can be a good thing. But it can also be somewhat frustrating, and one longs after a while for something a little different from the paradigm upon which KDE is firmly based.

GNOME: The GNU Object Model Environment

(A quote from the home page:) ``*GNOME stands for GNU Network Object Model Environment. The GNOME project intends to build a complete, user−friendly desktop based entirely on free software. GNOME is part of the GNU project, and GNOME is part of the OpenSource(tm) movement. The desktop will consist of small utilities and larger applications which share a consistent look and feel. GNOME uses GTK+ as the GUI toolkit for all GNOME−compliant applications.''*

The GNOME project is an attempt to do much of the same work as KDE, but even a little more than that. GNOME is less tied to one window manager, for instance, and it is interoperable not just between applications, but computers and platforms, as it uses the Common Object Resource Broker Architecture (CORBA). Also, and to many most importantly, GNOME is based on the GTk+ toolkit, which is free and open source, unlike the underlying toolkit of KDE, thereby following in the philosophy of Linux itself.

Resources:

> The official KDE website is http://www.kde.org/. The official GNOME website is http://www.gnome.org/. More detailed information regarding the issues surrounding GNOME can be found at http://www.gnome.org/gnomefaq/FAQ.txt. The home page of CORBA is located at http://www.corba.org/ , and the GTk+ toolkit home is http://www.gtk.org/ .

3.5 The Flashy Window Managers

The latest generation of window manger is very very pretty indeed. Sporting every convenience you could think of, and emulating the most beautiful operating systems ever used on the most gorgeous workstations in the world, these are the window managers to run if you've got the memory and CPU cycles to burn.

Window Maker

(A quote from the home page:) ``*Window Maker is an X11 window manager designed to give additional integration support for GNUstep applications. It tries to emulate the elegant look and feel of the NEXTSTEP(tm) GUI. It is relatively fast, feature rich, and easy to configure and use.''*

A big strength of this window manager is that it supports the GNU desktop, meaning that it makes a great and very pretty front−end to GNOME. This is also one of the most easily configurable window managers, and can be configured from a graphical interface, and supports the OffiX drag and drop protocol, easy switching of desktop themes, and it's now available within the popular Red Hat distribution, so it's easy and painless to switch from FVWM when the mood finally strikes. As of the 0.50 release, Window Maker supports KDE compliance as well.

AfterStep

(A quote from the home page:) ``*AfterStep is a Window Manager for X which started by emulating the NEXTSTEP look and feel, but which has been significantly altered according to the requests of various users. Many adepts will tell you that NEXTSTEP is not only the most visually pleasant interface, but also one of the most functional and intuitive out there. AfterStep aims to incorporate the advantages of the NEXTSTEP interface, and add additional useful features. The developers of AfterStep have also worked very hard to ensure stability and a small program footprint."*

Enlightenment

Enlightenment is more than just a window manager, it is an extreme, detailed, and configurable environment, and is particularly attractive in that it allows irregular and completely customizable window shapes. It is open in design, and instead of dictating a policy it allows the user to define their own policy right down to the minute and infinitesimal details; from its functionality right on through to its looks.

If you are using GNOME, you will find that Enlightenment is the default window manager, and in fact, it must be installed for GNOME to function. It is also basically the de facto implementation of the GNOME features for integration, making it the most practical choice for a desktop in that situation. Many other window managers will work alright with GNOME, but you will find that Enlightement excels. Unfortunately, it is still in development, and runs slowly and imperfectly from time to time.

Resources:

> The official Window Maker website is http://www.windowmaker.org/. The official AfterStep website is http://www.afterstep.org/. Enlightenment can be found, somewhat predictably, at http://www.enlightenment.org/.

4.Working In X

In this section we will become familiar with running clients in X Window and the basic procedures involved in using the system. X is not an intuitive interface on its own, and without any window manager, it is easiest to use as a display for programs started from the command line. One of the most common uses for X is just to have several xterm windows open at the same time. Not exactly maxing out the graphics capabilities of the computer, but it's a nice feature to start with.

The flexibility and usefulness of the command line is so great, in fact, that you don't really need a window manager. It's nice, and you will want it very quickly, but the fact is that you can get by without one, just the command line and the mouse. If you give it a shot, you'll be surprised as to the exact distinctions between the clients and the window manager. If you start toying around with this, however, *remember* that the key combination Ctrl−Alt−Backspace gets you out of X Window in a pinch.

4.1 Command Line Options

Most X programs try to use the same basic names for command line options. All applications written using the MIT X Toolkit Intrinsics automatically accept the following options:

−display display

This option specifies the X server to use. See the section on Display Names for details.

−geometry geometry

The initial size and location of the window, in a format such as *width*x*height*+*hoffset*+*voffset* or +*hoffset*−*voffset*. Note that if you put in a negative horizontal or vertical offset, the window will be placed counting backward from the right or the bottom of the screen, respectively.

−font font

The font to use for displaying the text in your window.

−bg color

The color to use for the window background.

−fg color

The color to use for the window foreground.

−name resource−name

Useful for specifying the name under which the resources for this application will be found. This is useful to distinguish between invocations of the same application, for example, two **xterm**s can be named differently so that they may inherit different resources based upon those names in the resource database.

−title string

This is the title to be used for the window on your display, generally used by the window manager to put a descriptive title at the top of the window. Not to be confused with the **−name** option.

–iconic

Invoke window as an icon.

–xrm resource–string

This option specifies a resource name and value to override any defaults that may already be set. Also useful for setting X resources that do not have explicit command line options. For instance, the command line `xterm –xrm "xterm*background: blue"` is functionally identical to typing `xterm –bg blue`.

4.2 Display Names

Every X Server has a display name of the form:

hostname:displaynumber.screennumber

The *hostname* specifies the name of the machine to which the display is actually, physically connected. The *hostname* can be omitted, and if so, the server on the same machine will be chosen. In fact, if you are the only one using X on your computer, you will want to just leave this off of your display specifications. The *displaynumber* should probably be zero, this is used if the X Server is controlling more than one keyboard and monitor unit, for instance, a network of X terminals. The *screennumber* specifies which monitor in a multiple monitor setup should be used. Following this specification, you would open an **xterm** window on your local machine with the option **–display :0.0**. You can see that we have omitted the *hostname* from the option, so the current machine is assumed.

On Linux systems, your *DISPLAY* variable holds your display name, which on my system is **:0.0**. This is usually set by **xterm**, or one of the scripts that starts X Window for you, although you can set it yourself, or as discussed above, use the **–display** command line option when invoking your application.

If you have opened an **xterm** or **rxvt** window, and then opted to do super–user work via the `su` command, you will find that if you try to launch an X application you will have no display to launch it on. Silly it seems, because you are sitting right in front of your display, but the trick to getting this to work is to pass an option on the command line to the program such as `–display :0.0`, and it will work fine.

4.3 XTerm Versus Rxvt, or, Know Thy Terminal Emulator

Your choice of **terminal emulator** can affect your experience in X almost as much as your window manager, so a little discussion of your best options is due here. A terminal emulator is a program such as **xterm**, discussed above, which lets you emulate the simple console of Linux in X. You will rely heavily on the program you choose, so it pays to choose it wisely to begin with.

If you have a slow video card, the first thing you should consider doing is switching to **rxvt**. Using **xterm** is a good starting point, mainly because it comes standard with all X distributions, but it's not always the best choice. It contains a lot of legacy code, and is quite bloated for a simple terminal emulator. This is partly due to the fact that it's not just a plain text terminal emulator, but also emulates graphics modes that you simply will never use. Because of this, you may wish to switch to **rxvt**, and also because **xterm** can be extremely slow. I'm not sure why it's so slow, but if you have a slow video card you'll know what I mean, and you'll be amazed at the difference when you dump **xterm**.

Some other perks of **rxvt** include pixmap backgrounds, and a much nicer scrollbar. If you specify the `−pixmap` option on the command line (and have support compiled in to your version!) you will get any `.xpm`−type picture in the background. A very cool feature, and it surprisingly doesn't slow down your output at all, it still redraws faster than your **xterm** window. Give it a shot, I haven't seen it around lately but I think it can be found on metalab.

5.X Startup

We will presume for the following examples that we have picked a fairly stable window manager, such as **FVWM2**, to try out some sample configurations. I would suggest giving that a shot for the purpose of learning these topics, as most of what you will learn here and in the following sections will apply to any window manager out there, but the topics seem most easily picked up using FVWM2.

5.1 A Sample Starting Configuration

Our first step is to write ourselves an initialization file for X itself. This file can be either a system−wide file, in which case it would likely be placed in */var/X11R6/lib/xinit/xinitrc*, or it can be overridden on a per−user basis by placing the file `.xinitrc` in your home directory. Generally, it is expected that there will be a basic, default file in the system−wide location, possibly enforced if necessary for security reasons, but otherwise users will probably wish to configure the file themselves.

First let's create a file in your home directory called `.xinitrc`. Open up your ``favorite'' text editor, and paste the following, or something like it, in that file:

```
 #!/bin/sh

 # if your backspace and delete are reversed, try this:
 xmodmap −e "keysym BackSpace=Delete" −e "keysym
Delete−BackSpace"

 xsetroot −solid darkslateblue

 # start some basic applications
 xclock −geometry 96x96+2+2 −bg grey40 −fg black −hl white &
 xload −geometry 120x96+2+147 −bg grey40 −fg white −hl darkred −update 4
&
 xterm −sb −ls −geom 80x25−2+2 −title "shell" &
 xterm −sb −ls −geom 80x25−2−2 &

 # start the window manager
 /usr/X11R6/bin/fvwm2
```

There are plenty of things to learn from this example. First of all, this file will be a shell script, as indicated by the first line. The **xsetroot** command on the second line turns the background of our desktop to a pleasant blue color, not a bad idea if we're going to be staring at that color predominantly all day.

The third and fourth lines are some programs that I like to leave running while I'm fast at work. You'll notice that some of the options make for a nicer setup, for example, specifying the colors and geometry (location on screen). I'll give you some tips for figuring this stuff out in a bit. The fifth and sixth lines follow similarly, opening up two handy **xterm** windows for us, which we will no doubt be needing soon.

The last line is very important−−it is this line that starts up your window manager! Notice that the only commands we did not run as *background processes* (by putting the **&** at the end) were **xsetroot**, **xmodmap**, **fvwm2**. With these first two it doesn't matter, as the programs exit immediately. But all the rest of the programs have to be in the background, otherwise when you closed one, it would kill your X Window session. That would not be very pleasant, nor very expected. As shown above, when you close **fvwm2**, you exit X.

5.2 A More Intelligent Startup

We can add lots to our primitive example of a startup file. For instance, this is a good way to warn yourself when you may have carelessly stared X as the root user. Red Hat users seem to do this often, for many of the configuration programs which must be run as root, must also be run in X. You can avoid this by issuing an **su** command to become root during your normal X user session, and then calling the program you need to run as root with the option **−display :0.0** discussed above.

```
# change background color for root
if [ "$USER" = "root" ];
then
     xsetroot −solid darkred
else xsetroot −solid darkslateblue
fi
```

This will check to see if you are the user named root, and if you are, it will set the background to a harsh red, rather than the usual friendly blue, to warn you. This next bit of code, also intended for your `.xinitrc` file, will merge in your user−specific and system−wide resources, first checking to be sure the files exist.

```
userresources=$HOME/.Xresources
usermodmap=$HOME/.Xmodmap
sysresources=/usr/X11R6/lib/X11/xinit/.Xresources
sysmodmap=/usr/X11R6/lib/X11/xinit/.Xmodmap
# Merge in defaults and keymaps
if [ −f $sysresources ]; then
   xrdb −merge $sysresources; fi
if [ −f $sysmodmap ]; then
```

```
   xmodmap $sysmodmap; fi
if [ -f $userresources ]; then
   xrdb -merge $userresources; fi
if [ -f $usermodmap ]; then
   xmodmap $usermodmap; fi
```

5.3 Getting The Windows Where You Want Them

Placing everything on your screen by guesswork with the *−geometry* option can get very tedious indeed. Particularly when you consider that you can specify the *−geometry* option to pretty much any program you can run in X. This allows a great precision in tuning the interface, but that's a heck of a lot of options to set, indeed.

This brings up one very nice feature of the FVWM window managers. By default, when you move a window around the screen, you see the geometry specification appear on the screen. Go ahead, try moving a window around with the left button. Now try resizing it. As you can see, you can get some primitive specifications this way. However, even this method can be a little difficult, and it would be nice to have all the details about your window in one concise list.

It is at this point that we will introduce the program **xwininfo**. To use this program, go to an **xterm** window and type in that program name. It will ask you to click on another window that you want information about, and after you click it will dump out useful information that it knows about that window. This is useful for plugging information about windows once you have them set up how you want them on your screen − run this program, then click on the window, then put in those parameters in your startup file, and your window system will henceforth be frozen in a pristine state of immaculate precision.

6.Configuring the Window Manager

Now we take a look at the files to configure your window manager. These vary from manager to manager, but for our examples we will focus on the per−user files, and we are looking at FVWM2 in this example, so you will modify the file *.fvwm2rc* in your home directory.

6.1 Basic FVWM2 Configuration

Configuring your window manager resource file in earlier versions of FVWM was a rather arduous process, because the order of the items in the file needed to be very particular, but that has relaxed quite a bit in this version. To my knowledge, the only crucial part of the ordering is the bindings for the menus, but we'll cover that a bit later. Your FVWM2 installation should have come with an example resource file, and you should be able to find that in */var/X11R6/lib/fvwm2/system.fvwm2rc*. This is the default system−wide configuration file. A good idea would be to copy that file into your home directory and call it *.fvwm2rc*. From that point on, when FVWM2 starts up it will read the file in your home directory, and ignore the system−wide configuration file.

Now that you've got a working, personalized copy of the window manager resource file in your home directory, open it in your favorite text editor, and take a look at it. If you're in X at the moment, you can see the correlation between the file and what you see on your desktop. Very helpful indeed. Let's try changing something and see if we can make it look a little better. Probably the first entry in your file (that is not followed by the # comment character) is the `WindowFont` entry, followed by a very bizarre font name. If you want to figure out how to set up the fonts and colors to their fullest, skip ahead to the next section on X Fonts and Colors.

6.2 Advanced FVWM2 Configuration

Configuring FVWM2 can get incredibly subtle and complex. Take this code snippet from the **fvwm2gnome** setup for the *.fvwm2rc* file:

```
# import config files

Echo −Styles−
Read .fvwm2gnome/config/styles/app.styles
Read .fvwm2gnome/config/styles/window.styles

Echo −Buttons & Keys−
Read .fvwm2gnome/config/buttons.config
```

This is an example of a way to read in other configurations. In this fashion you can modularize your *.fvwm2rc* −− not a bad idea, considering how jumbled one huge configuration file can become. This can make it easier to change and to debug, and it also makes it easier to configure so−called *themes*, which we will cover in a moment.

6.3 FVWM2 Configuration Shortcuts

A really neat way to configure your script for FVWM2 (and FVWM, and **bash**, and a whole lot of other programs besides!) is to use the **Dotfile Generator**, by Jesper K. Pedersen, available from http://www.imada.ou.dk/~blackie/dotfile/. You will need to have a recent version of **Tcl/Tk** installed (available with most Linux distributions). This program starts with a configuration preset, or takes the one you presently have, and allows you to tweak it via many structured menus of options.

Be warned, however, for I advise learning how to set up your configuration file yourself! Not only is this the *Unix Way* (for what that's worth) but it's much easier to make small and incremental changes to your `.fvwm2rc` file that way, and get things exactly how you like them. Also, being a **Tcl/Tk** program, it's a little slow. Nevertheless, if you want a painless way to start with a decent configuration, this is definitely worth a shot.

6.4 FVWM2 Themes

Among the many new possibilities of FVWM2 are something commonly called *themes*. This basically means that you can switch between a basic look for all of your windows on your desktop, on the fly. Note that if you've made the distinction between the functions of the window manager and the functions of the applications themselves, you will realize that a window manager theme is not going to affect the look and feel of the applications themselves. Integrated toolkits such as KDE and GNOME do have this capability, however, and the two might be used together very effectively.

In order to generate a theme, you must roll up your sleeves and modify your *.fvwm2rc* file a little bit. Here's something I added near the beginning of my file, try putting it right after your `Style` definitions:

```
# Blue Theme
DestroyDecor Blue
AddToDecor Blue
+ WindowFont -b&h-lucida-bold-r-*-*-*-140-*-*-*-*-*-*
+ TitleStyle ActiveDown (Solid DarkSteelBlue)\
    ActiveUp (Solid SteelBlue) Inactive (Solid Grey)
+ HilightColor white blue
+ ButtonStyle 1 -- UseBorderStyle
Style Blue UseDecor Blue, BorderWidth 5, HandleWidth 5,\
    MWMborder, MWMbuttons

# Function to change all windows to a new style.
DestroyFunc ChangeStyle
AddToFunc ChangeStyle
+ "I" Style $0 $1
+ "I" Recapture
```

That's a lot to swallow, I know. Basically we're first defining a theme called ``` ``Blue ```,'' and you can use that first half as a model to design other themes. Themes can describe many more features than that, in fact, and can be quite remarkably different from one another. Then we're defining a function to change all the windows to a new style. Notice in both sections above that we destroy the object before creating it. This is a good idea since you may well be restarting FVWM2 a lot to try out your different styles, and this makes it work a bit more smoothly.

The function call is needed as a generic interface to call the definitions of the styles we have defined. Now we will make the menu items to call them.

```
DestroyMenu "Themes"
AddToMenu "Themes"
+ "Choose a theme..." Title
+ "" Nop
+ "Blue" ChangeStyle "*" "UseStyle Blue"
+ "Mwm" ChangeStyle "*" "UseStyle Mwm"
+ "Flat" ChangeStyle "*" "UseStyle Flat"
```

We're being a little terse with the menu definition here, but there should be a lot more in the sample file on your system. What we're doing is calling the `ChangeStyle` function that we defined above to change the style for all the windows on the screen to one of the presets we defined above that. Notice, again, our good practice in destroying the menu before creating it. Now if you restart FVWM2 (you should also have a menu option for that, hopefully!) you will see a new `Themes` menu selection, and you should be able to try out the different themes.

For more examples of FVWM2 Themes, visit http://www.vis.colostate.edu/~scriven/Linux/fvwm/index.html.

7.Fonts and Colors

There are a lot of tricks to the fonts and colors used in X. They are not quite as simple as in some other systems, for instance, the font is not just a one–word name. You specify these resources quite explicitly, and it seems rather complex at first, but with a little explanation you'll be a whiz in no time.

7.1 Fonts Demystified

The *X Logical Font Description* (``XLFD'') is the full name for a font. It consists of the following fields:

- `fndry` – font foundry, the company or individual which made the font
- `fmly` – font family, the popular nickname of the font
- `wght` – font weight (bold, medium, etc.)
- `slant` – font slant (italics, oblique, roman (normal), etc.)
- `sWdth` – font width (normal, condensed, extended, etc.)
- `adstyl` – additional style (sans serif, serif, etc.)
- `pxlsz` – pixel size, the number of pixels vertically in a character
- `ptSz` – approximate point size of the text (similar to pxlsz)
- `resx` – horizontal resolution, in dpi
- `resy` – vertical resolution, in dpi
- `spc` – spacing, only useful, apparently, in the Schumacher fonts
- `avgWidth` – average character width of the font
- `rgstry` – the recognized registry that lists the font
- `encdng` – nationality encoding

In light of this chaos, the program **xfontsel** (the default X Window font selection program) will come in enormously useful to you. Try launching it right now. You will see a strange nothing helpful in the main window, but try holding the left button down on the `fndry` button. If all your fonts are in order, you will see a menu of selections such as `adobe` and `b&h` and `bitstream` and so forth. Select one such as `b&h` and you will notice that the font in the lower window changes to something intelligible. This is generally the way you will select fonts with this program, starting from the left, which is the most general selection, and moving toward the right, to the more specific options. Selecting an option toward the rightmost end will not make much sense before the foundry, for instance, is selected, because the options are generally ordered by their dependence on each other.

When you go to select from the `fmly` selection, you will see most of the options greyed out, and only three remaining. That means that these three are the only families of font made by this foundry. Some families appear under more than one foundry, for instance, both *Adobe* and *Bitstream* make a variation of the Courier font. Now you can select the `wght`, and so forth. After you get far enough you will have narrowed it down to the font that you want. You don't necessarily have to fill in all the options to choose a single font, there's not *that* many fonts on your system! The options that you do not select will be represented by a `*` indicating that any option will do in that spot.

When you are happy with your font selection, you can hit the select button, and your selection will be placed in the X clipboard, ready to be pasted into your document or whatever you are working on. For instance, go to your **xterm** window and type in something like `xterm -font` followed by an opening quotation mark. Then point to that spot on your screen, and click your middle mouse button (or click both the left and right, if you're middle−button impaired). This will paste the selection from the clipboard, which should be the font you just selected. Then enter the closing quote, and hit `Enter`. For instance, a nice big **xterm** with a Courier font specified would look like this: `xterm -font "-adobe-courier-medium-r-*-*-14-*-*-*-*-*-*-*"` A fresh **xterm** should the pop up using the font that you selected.

The utility **xfd** is very helpful for examining a font. If you launch it with a command line such as `xfd -fn fixed`, it will show you the character set for the font, much like the keycaps utility on a Macintosh. Note that you can also limit the number of fonts that you want **xfontsel** to display with the command line option *−pattern*, followed by a quoted font specification, as discussed above.

7.2 Font Aliases and Configuration

Sometimes it gets tiresome to remember all of the long font names, and very impractical too. Luckily, it is not necessary to type in a hundred keystrokes or so just to get the font name you want, for X provides something called *font aliases*.

If you look in the directory */usr/X11R6/lib/fonts/misc/fonts.alias*, you will find shortcut names for many of the fonts. For example, `8x16` is listed as a shortcut for `-sony-fixed-medium-r-normal--16-120-100-100-c-80-iso8859-1`, and anywhere you enter `8x16` as an X font resource or at a command line for a font name, the long version of the font will be substituted. The *75dpi* and *100dpi* directories have similar aliases to take advantage of, on most systems the Lucida Sans font has a nice shortcut.

If you add fonts or aliases to your system, you will have to issue a few commands (probably as root). If you add fonts, you will have to issue the following two commands (these are examples, you will have to put in the correct directory, or perhaps leave it off altogether, to have your fonts re−read correctly.

```
mkfontdir /usr/lib/X11/fonts/misc
xset fp rehash
```

If you change the alias file for a font, you may only have to issue the last command above, but it may be a good idea to issue them both, to be sure. With the **xset** command you can explicitly issue a font path you wish the server to use, you can delete a specific directory from your font path, see the man page for more information.

Another common problem is that some distributions (notably Red Hat 5.2, at the moment) come with the fonts configured in the wrong order. If you take a look at your */etc/XF86Config* (it may be somewhere else in some distributions, unfortunately, and I'm not sure where it is in Red Hat at the moment, so maybe **locate** it...) Take a look at this file if your fonts are ugly in X (as in, very difficult to read even at large sizes). You will see a bunch of entries that look something like this:

```
        FontPath   "/usr/X11R6/lib/X11/fonts/misc/"
        FontPath   "/usr/X11R6/lib/X11/fonts/75dpi/:unscaled"
        FontPath   "/usr/X11R6/lib/X11/fonts/100dpi/:unscaled"
        FontPath   "/usr/X11R6/lib/X11/fonts/Type1/"
        FontPath   "/usr/X11R6/lib/X11/fonts/Speedo/"
```

The important thing to note here is that we have the unscaled bitmapped fonts (the directories misc, 75dpi, 100dpi) before the scaled fonts (such as Type1 and Speedo). Bitmapped fonts are preferred for the X Server, because scaled fonts don't look very pretty for common use, they're better for things like The Gimp or Netscape, perhaps. You should also do a sanity check to see that these directories all do, in fact exist, and if you have modified any lately, you might need to (as root) go in and issue an `mkfontdir *` command in the **fonts** directory.

Note that if you are running a more recent distribution, such as one based on Red Hat 6.0 or later, you won't have to worry about this, as the *xttfs* True Type font server is the default, and does not use the path system to find your fonts.

7.3 Using Type 1 Fonts in X

The font types X generally accepts are of limited use, considering that about the only place you'll find fonts of that kind used are in the X Window System itself, and unfortunately most media junkies and fontaholics work in operating systems that prefer other formats. **Type 1 fonts**, most commonly used in conjunction with PostScript document formats, can be found for free on the internet with considerable ease. Try ftp://ftp.cdrom.com/pub/os2/fonts/ for starters.

To make use of these fonts is not that difficult, and graphics programs like **GIMP** will take advantage of them tremendously, and best of all, the Linux X servers understand Type 1 fonts ``out of the box.'' To use them, first unpack the archive using the appropriate commands, and put the fonts with a `.pfb` extension in */usr/X11R6/lib/X11/fonts/type1/* on your system. Then, add mention of those fonts in the `fonts.scale` file in that directory, using the format for the other fonts, already there. Then save the file.

Now, you should be able to `mkfontdir` to tell the X server about the updates, and then run `xset fp rehash` to re–read the font path. If this doesn't work for you, you might have to restart X to see the changes.

If you find yourself using Type 1 fonts a lot, and for things other than just X (GhostScript, for instance, can handle this font format as well), you'll want to check out the **type1inst** utility by James Macnicol. This program can configure Type 1 fonts for GhostScript and X, and it can also generate sample sheets for your fonts, and does most of the work of configuration for you. This program can usually be found at ftp://metalab.unc.edu/pub/Linux/X11/xutils/.

7.4 Using TrueType Fonts in X

If you have been using an operating system such as Windows or the MacOS, you may have lots of fonts sitting on your computer already that are **TrueType** fonts. TrueType fonts are considered best on smaller, low−resolution displays, such as your computer monitor, and attempt to provide nice features like shading and antialiasing, making your display look smoother. It is also really easy to find cheap TrueType fonts in bulk, and CD's featuring upwards of 500 fonts in the TrueType format are common in computer stores.

X does not understand TrueType fonts, and has no innate capability (at present) to do so, and so it needs a separate program to do the *font rendering* for it. There is a FreeType library to do this, but if you just want to use them on your computer, all you will need is a program called **xfstt**, the X font server for TrueType. This program should be available at ftp://metalab.unc.edu/pub/Linux/X11/fonts/.

Installation is fairly straightforward. Extract the archive, and `make` the file, and then `make install`. You may get a few errors that don't make sense after the second command; ignore them. What you need to do next is create a writable directory called */usr/ttfonts* and put some of your `.ttf` fonts there, just to make sure it's going to work. Then synchronize the font server with the command `xfstt --sync`.

If you got no errors there, run the font server in the background with the command `xfstt &` and tell your X11 server about the font service with the command `xset fp+ unix/:7100`. If you got no errors, you should be able to run Netscape, the GIMP, even xfontsel, and have those fonts available to you. Your biggest problem may be finding the font you want, but that's not really a bad thing.

It has been brought to my attention that more recent versions of xset require a slightly modified version of this command, because of changes in the code of xset. If you are running a distribution based on Red Hat 6.0 or later, or something equivalent, you will first want to try `xset fp+ unix/:7101`.

If everything seems to be working fine, at this point you will want to configure your system so that the fonts will be working when you start X. If you are just starting X from the command line, this is easy. All you have to do is add the two commands from the previous paragraph to your `.xinitrc` file, in that order. When you next start X, it should work just fine. If you are starting X via **xdm**, you will need to add `/usr/X11R6/bin/xfstt &` to `/usr/X11R6/lib/X11/xdm/Xsetup_0`. Got that? Then add `xset fp+ unix/:7100` as well.

Bear in mind that the installation of a TrueType font server can be tricky business. Remember that the order that you issue the above commands is important. The font server must be started before X is started, otherwise you will run into problems. Be sure to read the FAQ file that came with the distribution, and the `xfstt` man page too. Many newer Linux versions come with this already working, so see if that's not already the case before worrying too much about it.

7.5 Colors

Let's go back to our terminal window and try some more things. Try opening an **xterm** with a command line like the following:

```
        xterm -fg darkslateblue -bg red3 &
```

While that window may not be pretty, and you may not do much of your best work in it, it demonstrates one interesting theme of X, the names of the colors. While not very precise, this is a nice way to remember the colors more easily than remembering a series of hexadecimal numbers. Note that color names are never case−sensitive.

If you're interested in the gory details, or want to see the samples, or even want to replace those silly color names with your own for some strange reason, you can find the file listing all the colors with their hexadecimal representation in */usr/X11R6/lib/X11/rgb.txt* on your system. There are also some extremely useful utilities with names like **xcolorsel** and such. They can be found in the usual locations, on http://metalab.unc.edu/pub/Linux/ and elsewhere.

A more precise way of specifying colors is through the numerical definition. This consists of a color space name and a set of values in the following syntax:

<colorspace−name>:<value>/.../<value>

An RGB Device, which you will most likely be using, is specified with the prefix "rgb:" and has the following syntax:

rgb:<red>/<green>/<blue> , where <color> is a 1 to 4 digit hexadecimal number.

As an example, you might represent the color *red* with `rgb:ffff/0/0`. For backward compatibility, you can also specify red with a syntax such as `#ff0000` or `#ffff00000000`, and you will likely be seeing that (older) syntax a lot.

8.The X Resources

We will at this point consider some finer modifications to your system. Configuring the window manager only gets you so far. The window manager lets you customize everything that happens *outside* the windows, the frames, the borders, the buttons, the desktop... but what about making the *inside* of the windows pretty? The only way to do this is by modifying the *X Resources* for your system.

X Resources provide a mechanism for storing default values for program resources and tailoring your windowing environment to your favored look and feel. Resources are specified as text strings that are read in from various places when an application is run. Program components are named in a hierarchical fashion, with each object in the hierarchy identified by a class as well as an instance name. At the top level of the hierarchy is the class and instance name of the application itself. By convention, the class name of the application is the same as the program name, but with the first letter capitalized (e.g. Bitmap or Emacs) although some programs that begin with the letter ``X'' also capitalize the second letter for historical reasons (e.g. XTerm).

8.1 X Resources: The Basics

Let's try a few examples to ease you into the world of resources. Start by opening an xterm window. Most likely you have one open already, can open one with a menu selection, or may wish to backtrack a bit in this document to get your bearings! Type something like this into the xterm window:

```
xterm -background blue &
```

This command should have resulted in another xterm window popping up on your screen, but this time with a blue background. ``Wow,'' you say, ``that's amazing!'' Yes, indeed it is.

We'll need a little more background for our next example. Type exit in that xterm (which will close that window) and go back to the original window that you typed that command in. Try this: hold the Ctrl key and then hold the left mouse button. Now try that with the middle mouse button. Now the right. All xterm windows give you options to try while they are running. With the right button, for instance, you can change the font of the window. Notice the color of these menus. Now try this example:

```
xterm -xrm 'xterm*fontMenu*background: green' &
```

This time, it just looks like a normal xterm window. But if you try holding Ctrl and the right mouse button in that window, you will see that this menu (the ``fontMenu'' mentioned in the above example) is green. What just happened? If you look at the man page for **xterm**, you will see many command–line options, such as *−background* or *−font*, that you can set when you launch any given window in X, listed after the options just for **xterm** windows. These particular options are called *X Toolkit Options*, and they apply to just about anything in X.

8.2 Inside The X Resource Database With *editres*

To really get inside the resources of X, we need to run **editres**. Go ahead and launch it (and a sample program to play with) now:

```
xclock &
editres &
```

Probably you will see a simple clock, and the rather innocent looking, mostly−blank window of **editres**. The main window is showing us the resource tree of any given window on our desktop, and upon program launch, there is none. So let's give it one. Hold down the left button on the Commands menu, and select Get Tree. In the top of the window will appear `Click the mouse pointer on any toolkit client.` This message is more than it seems, and it is a hint to us that not all X applications are toolkit clients (though most of the basic ones are, and the program will usually tell us if it is not in the manual page). The mouse cursor will turn to a crosshair, and wait for you to click on another window.

For this example, let's first click on the **xclock**. You will see a couple things appear in the **editres** window. These are the configurable branches that the toolkit sees. Click on the bottom right one (clock). It should invert. Now select ``Show Resource Box'' from the ``Commands'' menu. You will see another window pop up, with the heading ".xclock.clock.unknown". Bingo!

>From here you can toy with the configuration options of the main **xclock** window. First, click on ``Set Save File'' at the bottom, and put in a filename such as */home/yourname/resources*, to indicate that you're testing some of the resources here. Now click on ``foreground'' and put in ``blue'' next to the ``Enter Resource Value:'' prompt. Then hit ``Apply'' at the bottom. The minute notches around the clock should turn blue. Go to ``background'' and enter ``navajowhite'' (or whatever color you prefer). Then apply that. As you can see, we can configure the whole window just fine this way. But if you were to try launching another **xclock**, it would appear just as this one appeared before any changes. We need to save these changes.

Click ``Save'' and then ``Popdown Resource Box'' (a fancy name indeed for the ``Close'' function!). If you view the file you just created (*cat /home/yourname/resources* perhaps) you will see all those resources spelled out, in correct X Toolkit format, for your convenience. But that does us little good, for if you try launching another **xclock**, it will still look Plain Jane. So here is the last and final step in loading in your resources:

```
xrdb −merge /home/yourname/resources
```

This command *merges* the resources we just wrote into the *X Resource Database* (**xrdb**) for your session of X Window. That means that for all future invocations of the X applications we modified, our changes will take place, and remain binding. So if you run another **xclock**, it will look as nice as you have just now set it up, every time you run it. Mind you, **xrdb** is a complex program, and you may want to have a look at the man page before moving on, or playing around with it some more. If you added the modification to `.xinitrc` listed earlier in this document, to load the resources automatically on X startup, you should only have to worry about the **xrdb** command when you make changes during your X session.

8.3 The Anatomy of X Resources

As you can see we have stumbled across a plethora of configuration options here. This method of configuring X, as has been stated, offers nearly limitless possibilities, and an equivalent amount of confusion. To get some picture of the scope of the resources in just the X Toolkit Intrinsics alone, run the program **viewres**, and play around with it a bit. This program graphically displays the tree of resources in the *Xt Library*.

If you read the manual page for **X**, you will find a rather obscure definition of the exact syntax for defining resources it understands. We can simplify this quite a bit and break it down into this essential syntax definition:

<program><binding><widget><binding><widget><...><resource>:<value>

That doesn't really seem that simple, actually. Well, let's define some things about what has just been said, and it will all start to make sense after all. You can peek ahead to the examples in the next section as you read along, if you wish.

The *program* in this definition is the invocation of an application in the resource database. This would be `XTerm` for an invocation of **xterm**, `emacs` for an invocation of the **emacs** text editor, or a user−defined name that was given when the applications was launched with the **−name** command line option. In this way you can define separate resources for **xterm**s which will be used in different ways. Which is pretty cool, really.

The *binding* can be one of two characters to separate the widgets and such. If you use a `.` (period), you get a *tight binding*, which means that one widget is directly above or below the other in the widget hierarchy. This also has the highest precedence of the specification methods. If you use a `*` (asterisk), you get a so−called *loose binding*, and you will skip any number of widgets in the hierarchy, and it will attempt to match the next possible widget defined.

The *widget* entries are items in the widget tree, in order of most−specific to least−specific, that they appear in the widget tree, visible with **editres**. Any single widget entry can also be replaced with a `?` (question mark) to skip a single widget definition, and match any possible widget item.

The *resource* item must be specified, and cannot be replaced with the `?` character. This is the most−specific item in the hierarchy, and usually contains items like the actual color to define, actual font to define, and so forth. In fact, everything else before the resource in this definition can be left out and replaced with a single asterisk, but the actual resource to define must be present. If you just put an asterisk and the most−specific resource name, such as `*background: blue`, X will attempt to define that resource globally, for all its clients, if possible.

Following the colon is the *value* entry. This entry defines what the resource will be set to, such as a font name or color value. The value can be specified (depending on context) as a boolean, numeric, or text data type. The value entry, also, cannot be omitted in a valid resource definition.

8.4 Making Your Changes Last With *.Xdefaults*

There is a magic file you can put in your home directory called *.Xdefaults*. If you copy the lines in the resources file from the last example into the *.Xdefaults* in your home directory, you will never have to configure **xclock** again! While this might not be the finest example of its utility, it makes its point. This file can be crammed full of every option you prefer for every type of program you run in X, and if you take proper care of it, you can still easily go back and make slight changes when you need to. But making lots of changes, and hunting down lots of subtle resources using **editres** can be an extremely tiring and painstaking procedure. Indeed, sometimes that's too much work, and most of these resources are already waiting for you, neat and orderly, grouped by program, on your system.

In the directory */var/X11R6/lib/app−defaults* you will find a great many files, all named after an X Toolkit program. If you examine these files you will find that they contain a great many configuration options for each one, and I do mean a great many! You would not want all of these options from all of these files in your *.Xdefaults* file, that would be quite tiresome to deal with. These are the defaults, and it is from these that you can decide what you would like to see changed for your particular configuration.

The following are some samples from my *.Xdefaults* file. Notice a few things we have not yet mentioned about the resource definition files. If a line begins with `!` (exclamation point), it is considered a comment, and the rest of the line is ignored. If the line begins with `#include filename`, that line is an *include directive*, and at that point in the resources another file will be merged, when it is loaded. This can help keep your resource files from becoming too bloated. And here are some examples:

```
! Default resources for me@localhost xterms
! start with the generic, move to the specific...
*Dialog*Text*font:     -b&h-lucida-medium-r-*-*-12-*-*-*-*-*-*-*
*dialog*value*background: white
*Dialog*Label*font:    -adobe-helvetica-bold-r-*-*-12-*-*-*-*-*-*-*
*MenuButton*font:      -adobe-helvetica-medium-r-*-*-12-*-*-*-*-*-*-*
*MenuButton*background:        grey80
*MenuButton*foreground:        black
*Label.font:           -adobe-helvetica-medium-r-*-*-10-*-*-*-*-*-*-*
*Label*shadowWidth:    1
*SmeBSB.font:          -adobe-helvetica-bold-r-*-*-12-*-*-*-*-*-*-*
*SimpleMenu*font:      -adobe-helvetica-medium-r-*-*-10-*-*-*-*-*-*-*
*OptionMenu*font:      -adobe-helvetica-medium-r-*-*-10-*-*-*-*-*-*-*
*Command.font:         -linotype-helvetica-bold-r-narrow-*-12-*-*-*-*-*-*-*
*commandBox*font:      -b&h-lucida-bold-r-*-*-12-*-*-*-*-*-*-*
*Toggle.font:          -adobe-helvetica-bold-o-*-*-12-*-*-*-*-*-*-*
*Form.background:      grey70
*TransientShell*Dialog.background: grey70
*Scrollbar.Foreground: grey80
*Scrollbar.Background: grey50
*Scrollbar*cursorName: top_left_arrow
*Scrollbar*width:      16
*shapeStyle:           Rectangle
*XlwMenu.shadowThickness: 1
*shadowWidth:          1

! xterm stuff
xterm*scrollbar.background: grey40
xterm*foreground: grey90
xterm*background: grey25
xterm*cursorColor: white
xterm*visualbell: on
! rxvt stuff (a quicker, better xterm)
rxvt*color12: steelblue
rxvt*color15: white
rxvt*color9: rgb:ff/7f/5f
rxvt*foreground: grey90
rxvt*background: grey10
rxvt*cursorColor: white
rxvt*font: lucidasanstypewriter-12
rxvt*loginShell: false
rxvt*saveLines: 1024
rxvt*title: shell
rxvt*geometry: 80x25
! Make Xman just a little bit more sane
xman*topBox: false
```

```
xman*background: lightsteelblue
xman*foreground: black
! xcalc is too bland by default...
xcalc*Command.font: -adobe-helvetica-bold-r-*-*-10-*-*-*-*-*-*-*
xcalc*customization: -color
! Disallow the <blink> tag in Netscape
Netscape*blinkingEnabled: False

! Merge other resources (example)
# include $HOME/.otherXresources
```

One word of warning with regard to X resources, for KDE users. There's a setting in the control panel which will cause all your X Resources to be overwritten in your applications. This can be a really nice feature, but it can also be an annoyance if you have painstakingly configured your resources. Open the control panel and find the ``Fonts etc.'' section. There's an option to ``Apply style to non−KDE apps''. If you turn this on, all your programs will adopt the look and feel configured by KDE (you may need to restart X to notice this). If you leave this option turned off, your resources will remain as you have configured them.

8.5 Your Own User Resource Directory

You can also create a directory of resource files, just like the system−wide *app−defaults* directory mentioned above, with one file per program. Just create the directory (for our example we'll use app−defaults under your home directory,) and then set the environment variable `XAPPLRESDIR` to point to it. A good place to set this variable would be in the beginning of your `.xinitrc` file, for example, put in the line `export XAPPLRESDIR=$HOME/app-defaults` (if your files are going to be in an *app−defaults* directory under your home directory).

Now, whenever you start an X program, this directory will be searched for a file with the same name as the resource name of the program, just like the system−wide directory. This is the *client name* that you used in `.Xdefaults` files.

For example, a file called *XTerm* could contain the line `*background: gold`, and all your **xterm**s would, by default, come up with a gold background. This is a nice alternative to a single `.Xdefaults` file, and makes it more clear when trying to decide which settings to configure later on, and to find the ones for a certain program. There are still uses for the `.Xdefaults`, though. It's useful for setting resources not bound to a single program, like modifications that you would make to turn all of a certain kind of button blue, regardless of the application.

9.Clients and Application Tips

We have covered a few clients in X, and this section will cover some more. We will limit our discussion here to the most basic and important core items, those which come with X or you are likely to find yourself using with X. If you have installed an integrated desktop environment such as KDE or GNOME, you will have many others to choose from which likely perform many of the same functions. However, it is important to know about and understand many of the fundamental programs in X, because they can be very useful for working with your environment and such. Also X offers many new options that even your normal console applications can take advantage of.

9.1 Screen Savers for X

A common feature of many operating systems is the ability to blank the screen after a specified amount of time, and optionally display some sort of nifty graphics demo thing, a screen saver. There are a couple ways to do that in X, too.

The most basic way to use this feature is by putting a command in your `.xinitrc` startup file such as `xset dpms 2400 3600 4800`. The **xset** program can configure the screen saving features of the X server, not to be confused with the screen blanking that the kernel does when you are at the text console. With the `dpms` option, X can use the power saving features of your monitor as well. The first option configures how many seconds before the screen blanks, the second option is how many seconds before the power saving feature starts, and the third option is for the "off" mode. Turning on an option implicitly enables the feature, setting a feature to zero explicitly disables it.

Many Linux distributions come with **xlock** preinstalled, or as an option. This is a pretty basic and fairly nice screen saver. If you run it with the `-nolock` option, you can see some of the modes that it offers, and if you leave that option off, it will ask you for a password when you move the mouse or press a key, as a security feature. Note that this is no real security, for at a Linux console a user could restart the computer or just drop out of X with a combination of keystrokes. The last method can be disabled, however, and if you are using **xdm**, it will offer as much security as your login, so it may be that only rebooting will let someone in.

A newer and better program is **xscreensaver** by Jamie Zawinski. This program offers a great many niceties, for instance, it can run its processes at a nicer priority level, lessening the load to the system while it's running, and it automatically detects when the screen has been powered down by **xset** and doesn't waste processor time. Also, all of the graphics routines it calls are modular demos, and you can add routines without upgrading the whole package, and it can also call other programs, such as `xearth` or `xdaliclock`, as modules.

The latest version of **xscreensaver** can be found at http://www.jwz.org/xscreensaver/. Once you get it installed and ready to go, here are some nice additions you might wish to add to your `.Xdefaults` file:

```
!!! some XScreenSaver sample defaults
! Time out after 3 minutes, cycle mode after each 2
xscreensaver.timeout: 3
xscreensaver.cycle: 2
! Run very low priority, and fade between modes
xscreensaver.nice: 12
xscreensaver.fadeSeconds: 2
```

9.2 Emacs and XEmacs

If you are a fan of the text editor **EMACS**, or just someone who uses it a lot, you will find your work even easier in X Windows. If you have not tried XEmacs, you may want to get it for use in X. There are features in XEmacs that are nice even if you are not in X, for instance, your text can be colored to match the markup style you are editing automatically. You should give the following modification to your `.emacs` file a shot, and read the info pages for more options. Also look for an option to edit faces in the menus.

```
(global-font-lock-mode t)
(setq font-lock-maximum-decoration t)
```

9.3 Some Useful Programs and Tricks

appres

The appres program prints the resources seen by an application (or sub−hierarchy of an application) with the specified class and instance names. It can be used to determine which resources a particular program will load. Useful for debugging your X defaults and such.

rclock

Many distributions come with this nice replacement for xclock, which saves memory, alerts you when your mail comes, and can pop up reminder messages and launch programs. The **Battery−Powered Mini−HOWTO** contains instructions on patching this utility to show how much battery is left in your laptop, too.

rxvt

A nice replacement for xterm − uses less memory, works faster, lets you put in a background pixmap, and lets you switch fonts with keyboard hotkeys, rather than menus.

xcpustate

Displays CPU state (idle, nice, system, kernel) statistics, as well as Ethernet information.

xearth

Display the earth on your root window, many options for display available. **Xscreensaver** can use this as a screensaver module, for maximum fun.

xfig

A vector drawing program, particularly useful for charts and documentation. Quite useful but hard to get the hang of at first.

xfontsel

Font selection utility for X Window. Try the command `xterm -fn `xfontsel -print` &` to pick a font and then open the xterm window using that font.

xload

Monitor your memory usage with a moving graph or the lights on your keyboard! If you use Window Maker, look for **wmmon** to do the same, but prettier.

xmag

A magnifying glass for X, with a couple other useful features.

xman

Manual page browser for X. If the little box it starts with gets annoying, launch it with the *-notopbox* option.

xmodmap

Edit and display the keyboard modifier map and keymap table that are used by client applications to convert event keycodes into keysyms, usually run from user's startup script. An example was given earlier in this document, see the man pages for more info.

xpaint

Basic bitmap painting program, for any real work you should grab GIMP.

xset

User preference utility for X. You can change all sorts of stuff with this. For instance, `xset s 600` sets the screen to blank after ten minutes.

xsetroot

Change the color of your desktop. If you have a color selector program like **xcolorsel** installed, try a command like `xsetroot -solid `xcolorsel`` to pick a color and set your desktop to that color.

xwininfo

You can run this program and click on any window for lots of useful information about it.

The Intellimouse

You can use the Intellimouse in X with a great many applications. There is an excellent resource page located at http://www.inria.fr/koala/colas/mouse−wheel−scroll/.

10.Advanced X Usage

There's a lot to learn about X, a lot of info buried in the inscrutible land of man pages, a lot of things most folks don't bother reading. Some if it is rather important, and it's easy to make some mistakes, and get completely stuck. It can be much harder to build a program in X, or check the security of X, or many other things, because X and all the things that run on top of it are so vast. This section is an attempt to cover some of the more advanced and confusing situations that may arise.

10.1 Libraries and Compiling X Applications

Sooner or later you will have to deal with compiling applications of your own. Later, if you just installed a nice distribution of linux, and are happy with what you've got, sooner, if you're the kind of person who likes to tinker and install. Remember, this is a privilege, not a right, so have fun with it!

First, a few pointers on compiling programs with X. Many newer applications, GNU applictions in particular, come with a script in the root directory called **configure**. This assumes of course that you've extracted the file and are in the directory. This program should be run as `./configure`, and will automatically detect many things about your system. Afterwards, simply running `make` and perhaps becoming root and running `make install` will get the program up on your system.

You may have to do a little more tinkering if you do not have a **configure** script available. Many X programs require you to run a program to make the **Makefile**, called `xmkmf`. If you don't see a **Makefile** in your directory, this will sometimes work, and will generate a suitable configuration for you, and you'll be ready to `make` from there.

Sometimes you will have run the **configure** script, and have been warned that you lack a library that would be helpful or necessary to properly use that application. For instance, I recently installed the **xscreensaver** application, and found that it would support several 3D modes if the **Mesa** library was installed. If you run into this situation and want that library installed, the first thing you should do is check the CD or installation media for your Linux distribution to see if you have the library on there. That may save you lots of trouble trying to compile the library.

Once you get a library compiled and installed and ready to use, you can go back to the directory you were installing your X program from, remove the `config.cache` file, and run the **configure** script again. Hopefully it will find the library you have installed. Sometimes it does not, and you have to tinker to get it working. This happens occasionally with libraries you download off the net, that do not adequately support the Linux File System Standard (most do, however).

10.2 Basic X Security

It has often been said that X has a very simple security model: All or Nothing. This is not much of an exaggeration at all. X can be configured to use somewhat sophisticated security, via encryption, but that is beyond the scope of this HOWTO (for the present). It is assumed that the user is not using any encryption for this discussion.

First of all, you should try and follow some simple rules when you're compiling programs for X (or for any reason, really). Try not to become root any more than necessary. Configure your programs as a normal user with the `./configure && make` (or just `make` if there's no configure script), and then issue the single command to install it as root by typing `su -c "make install"`, and then typing in the root password at the prompt. That way you won't have to be logged in as root, you can just issue a single command, and you're back to your normal user prompt. This is a smart, safe way to install software.

The next thing to think about is running X software as root. Realize that X is more or less inherently insecure, and if your primary concern for a particular Linux box is security, you really don't want to install X at all! Having said that, some folks want to run nice fancy configuration programs or package management tools in X. I **do not** recommend starting X as root. It's just not a good idea. There's much, much better ways to do these things!

If you want to run an X application as root, just log in under your normal user account, and launch it from there. As I mentioned above, you don't want to be logged in as root any more than absolutely necessary. The root user has the ability to do just about anything on the system, including about a million ways to destroy it completely. Simply go to your **xterm** or such application and type in a command such as `su -c "glint -display :0.0"` to launch the window on your desktop. Now you have a single window running as root, and not the whole desktop. You might also want to consider using the complete pathname to the program you're running as root, for extra security.

Tomasz Motylewski also offers the following tip for **ssh** and **ssh−agent** lovers. If you put the following in your */usr/X11R6/lib/xinit/Xclients* file:

```
eval "exec ssh-agent fvwm${FVWMVER} ${FVWMOPTIONS}" >
"$HOME"/.FVWM${FVWMVER}-errors 2>&
```

That is, replace your standard `fvwm${FVWMVER} ${FVWMOPTIONS}` command with the preceeding. That is to start your window manager as a child of **ssh−agent**. Then you should try `ssh-add < /dev/null` from an **xterm** session (if you have your identity RSA key in the file *˜/.ssh/identity*). If this works for you then you should find that you are authenticated in all your X windows.

It should be noted that this is a suggestion from the reader, and the author has no experience with **ssh** at the present, so proceed at your own risk! More recent Linux distributions allow for setting up **ssh** automatically, via `setup` or something similar, which means that you may not have to worry too much about setting it up, just understanding how it works.

10.3 More About X Authority

The mysteries of user authentication in X are a prime example of the security problems many users encounter with X. Tomasz Motylewski relates the following story, which sums up the situation quite well.

``...in the default XFree86 configuration X session started by ``startx'' accepts commands from everyone connecting from localhost. If you start X as user1 and you have user2 remotely logged in, user2 has full access to your keyboard and desktop (once my friend played a joke on me and put in his cron commands dumping my X desktop image and sending to him every hour). I did not notice it for 2 weeks!''

Well that's not exactly a good thing, but unfortunately many readers have this exact problem. Tomasz goes on to point out that if you use **xdm** to log in (as detailed earlier in this HOWTO), rather than just running **startx**, this won't be a problem because access is controlled by the **xauth** program. By issuing the **xhost** command, you can see who is allowed access to your X session. Any user from the hosts that are listed in the output of the **xhost** command is authorized full access to your screen and keyboard.

If you want to start your X server in a secure fashion from the command line, you will need to make the following modifications to your configuration. First, modify **startx** (probably at */usr/X11R6/bin/startx*) by changing the last line `xinit $clientargs -- $serverargs` to read `exec xinit $clientargs -- $serverargs -auth ~/.Xauthority`.

Then you will also need to edit your **.xinitrc** or your system−wide */usr/X11R6/lib/xinit/xinitrc* (whichever you are actually using, probably the one in your home directory), by adding this at the beginning of the file:

```
 # if Xserver has been run with "-auth ~/.Xauthority" option
this will
 # prevent other users on your machine to connect to your X server
 # unless you allow it explicitly using xhost +host or give them
 # your ~/.Xauthority file.
 xauth generate $DISPLAY . trusted
```

Always be sure and run **xhost** to check the security that you have configured, to make sure everything is working correctly.

11.Bibliography and Other Resources

There is lots of information on X Window to be found, on your computer right now, on the Internet, and in some excellent books available at your favorite bookseller. Give some of the following resources a try, they have been extremely helpful to the author, and can be extremely helpful to you, too.

- Check the Linux Documentation Project website for a lot more documentation on Linux, X, and related items.
- http://www.x11.org/is sort of a clearinghouse for all things X.
- http://www.themes.org/is a headquarters for themes for various window managers.
- The X Consortium's web site is http://www.x.org/... or perhaps it's moved to http://www.opengroup.com/.
- XFree86 can be found at http://www.xfree86.org/.
- The O'Reilly series on X Window! Visit http://www.ora.com/for the definitive books on X.
- Much more information on using TrueType with X is at http://www.freetype.org/.
- The man pages for **X**, **xterm**, **XFree86**, and for any other clients you find yourself using often, are very useful and quite information−packed, and highly recommended. As the oft−repeated saying goes, RTFM.
- There is a **Remote X Apps** MINI−HOWTO that is very helpful in figuring out how to run local and remote clients with X.
- Don't forget to visit http://www.gnome.org/ and http://www.kde.org/ for the latest on unified desktop environments in Linux, which are becoming more and more commonplace all the time.
