Customizing the X Windows System

This document helps individuals learn how to customize their X environment. Before you try to customize your environment, you should be very familiar with working with X. If you need help getting started, read the document, Introduction to the X Windows System (UNIX 2).

Table of Contents

Customizing X

This section explains how to customize X. The other two main sections of this document cover twm customization and recovery from failed customizations.

The getxfiles Command

If you decide to abandon or shelve your customization, Information Technology—not X—provides the **getxfiles** command. This command lets you copy the current X default files into your directory. Since **getxfiles** is not an X command, it only exists on systems maintained by Information Technology.

getxfiles requires that you specify a window manager, one of **fvwm** or **4Dwm** (for SGI's only). **twm** is the Tab Window Manager discussed in this document, so most likely you will choose twm. **mwm** is another window manager (the Motif Window Manager). Generally, if you are using mwm, the names of your configuration files alter accordingly, i.e. **.twmrc** becomes **.mwmrc**, likewise with the **.fvwmrc** and **.4Dwmrc** files for fvwm and 4Dwm, respectively.

Example usage:

```
prompt% getxfiles twm
```

Normally, if you execute **getxfiles** it replaces your .Xresources and .Xsetup.twm files with the system default files. If any of these files already exist, it will ask for confirmation before replacement. Note that **getxfiles** by itself does not replace .twmrc; for that, you need to use one of the command options. **getxfiles** has several options; three of them appear below.

The -x Option

This option stands for "expert," and it replaces .Xresources, .Xsetup.twm, and .twmrc. **getxfiles** requires confirmation if a file exists. Thus, you can replace only certain files without losing customizations in other files. Example:

```
prompt% getxfiles -x twm
```

getxfiles has other options for copying different combinations of files. Type:

```
man getxfiles
```

for more information about the options.

X and twm

X supervises network communication, and often runs a client called a **window manager**, which handles window functions like moving, resizing, and iconifying. Machines managed by Information Technology default to the Tab Window Manager, or twm. This hierarchy allows X to concentrate on network communication. As a result, you can customize X without changing twm, and vice versa.

When you start X, both X and twm read start-up files that include information concerning startup windows, window locations, menus, and mouse button functions. By editing the appropriate file, you change these attributes.

X Start-up Files

The following start-up files provide information for X when you login. They open windows automatically, position them, set borders and backgrounds, and specify window titles. Because the filenames begin with a period (.), they normally do not appear when you list your directory with the UNIX command **ls**[1]; use **ls -a** (for "list all").

.Xauthority	Limits the ability of other compute servers and display servers to affect your display. You **should not** alter this file; it is designed to be read and written only by the computer.
.Xdefaults	Overwrites any existing X attributes like window size and border width. Information Technology maintains a .Xdefaults file for all users, which permits certain network-wide changes. If you create your own .Xdefaults file, you may fail to see such changes. Hence, you **should not** make your own file.
.Xresources	Like .Xdefaults except that it merges its X attributes with existing X attributes rather than overwriting them entirely. This property makes it more flexible than .Xdefaults. This is where you should customize resources for X applications.
.Xsetup.twm	A file containing X commands that open initial windows or icons. Customize this file to change your initial window setup.
.xinitrc	A file containing X commands; it also calls on .Xsetup.twm. Information Technology maintains a file for all users, so you **should not** make your own file. Doing so may break things for you down the line.

From the above list, you have two files, .Xresources and .Xsetup.twm, which are available for X customization. .Xresources specifies different attributes such as colors, font, and title for each kind of client (xterm, etc.), which X reads directly.

.Xsetup.twm starts clients automatically when you start an X session and gives information such as starting location and size for these clients. Unlike .Xresources, the login process reads .Xsetup.twm directly. As a result, each file has a different syntax.

1. For more information on this and other UNIX commands, consult the on-line manual pages and the document *Introduction to the UNIX Operating System*.

Customizing with Resources

This section introduces some X terminology, a special X syntax, and explains your default .Xresources file. Some simple exercises familiarize you with ways to customize X by using resources.The section, *Customizing with Commands*, explains how to customize X with commands and command options.

Resources

X maintains many parameters, each describing a different attribute or a group of attributes of your X configuration. X uses the term **resources** for these parameters. Resources might specify an icon's border color, a title's font, or a window's size. When a client starts, X applies the appropriate resources. *Your changes do not affect current clients, only the clients started after a your .Xresource file has been merged with the resources currently in use by X*.

Widgets

As one of its basic building blocks, X employs components called **widgets**. These provide standard tools that can be combined to define a client's appearance, its input and output methods, and other options. A client usually consists of several widgets, and a widget may contain widgets within itself. Some programs use no widgets; you must refer to that program's documentation to determine what resources are customizable.

Standard widgets used by many X programs include:

- an input-output area
- a dialog box (like the network notices displayed when you log in to an X terminal)
- a command button (like the aXe buttons)
- a scrollbar
- a menu

For example, consider the xterm program itself. It has at least three widgets: a Main Options menu, a VT102 window, and a Tektronix 4010 window. The VT102 window has four widgets: an input-output area, a scrollbar, a VT Options menu, and a VT Fonts menu. The Tektronix 4010 window has two widgets: an input-output area and a Tek Options menu.

Resource Specification Syntax

When you declare a resource parameter, you make a **resource specification**. All resource specifications use the same syntactic structure, shown and explained below.

appname[/.]subname[*/.]subname[*/.]...: value*

appname	The name of a client, such as xterm. If you do not include *appname* then all applications to which *subname(s)* apply will be affected.
[/.] (binding)*	Either an asterisk (for a loose binding), or a period (for a tight binding). Bindings are explained below.

subname	If it is the last item, *subname* is a variable or a widget. If it is part of a chain of *subnames*, *subname* is a widget.
value	The new resource value. Depending on the variable, *value* may be a color, a number, a geometry, a name, etc.

Widget Hierarchy

The left-hand side of the colon (:) lists the **widget hierarchy**. For example,

```
xterm*vt100.geometry: 80x48+0+0
```

is a three-level widget hierarchy: the xterm widget, the vt100 widget, and the geometry widget. The xterm resides at the highest level of the hierarchy. vt100 is one of the widgets under xterm and geometry is a widget under vt100.

Bindings

Bindings effectively “walk down” a client’s widget hierarchy. A period (.) produces a **tight binding**, which gives an explicit “walking path.” An asterisk (*) creates a **loose binding** so that if the path from client to variable is uncertain, X searches for every valid path. Consider the following bindings which activate the window dimensions:

```
xterm.vt100.geometry: 80x48+0+0

xterm*geometry: 80x48+0+0

xterm*vt100.geometry: 80x48+0+0
```

The first resource specification is a tight binding. It also does not work because the actual xterm contains another widget between xterm and vt100.

The second specification, a loose binding, works because X walks down all the paths that begin with xterm and end with geometry. It yields unexpected results, though, as ALL xterm windows, even normally small popup windows, will have this geometry. In the last specification, X walks down every path beginning with and ending with vt100.geometry, so this specification works. It also demonstrates that you can combine tight and loose bindings in a longer binding.

Observe that you can omit *appname* altogether to specify every case of a resource. For example, the resource specification

```
*scrollbar: on
```

tells X that every client that permits a scrollbar should activate the scrollbar.

Generally, you should use loose bindings because they maintain compatibility. If later versions of the X Window System change a client’s widget hierarchy, your customization will probably perform correctly. Loose bindings are flexible; tight bindings are brittle.

Classes and Instances

Related resource variables constitute a **class**; each variable itself forms an **instance**. An example[1] can clarify this concept. An xterm contains a class called Foreground, and Foreground consists of several

instances: foreground color, pointer color, and text cursor color. Thus, to make these three instances all be the color blue, you could have the resource specifications:

```
xterm*foreground: blue
xterm*cursorColor: blue
xterm*pointerColor: blue
```

or you could specify just the class:

```
xterm*Foreground: blue
```

Notice that class names begin with a capital letter, whereas instance names begin with a lowercase letter (subsequent letters may change case).

Precedence

The two resource characteristics—tight/loose bindings and classes/instances—carry the following precedence:

1. More-specific (tighter) bindings take precedence over less-specific (looser) ones.
2. Instances take precedence over classes.

This precedence means you can make a general resource specification, then preempt it for a few special cases. As an example of (1), suppose you want all windows except your clock (xclock) to have a title. You could use the declaration:

```
*showTitle: on
xclock*showTitle: off
```

For an example of (2), suppose that for windows, you want a red text cursor, but a blue border and pointer. You could declare:

```
xterm*Foreground:blue
xterm*cursorColor: red
```

Recall that Foreground is a class, but cursorColor is an instance, so when X determines the cursor-color, cursorColor takes precedence over Foreground.

Values

Most resource specifications take a value that you would naturally associate with the resource. For example, cursorColor takes a color value, and borderWidth takes an integer value. The geometry resource requires a value in the form of *geometry* (more later on this option).

Some resources, like scrollbar, are booleans, and most of them take “on” or “off” as their value. However, in some cases these resources take “True” or “False,” which can create confusion if you do not realize this inconsistency. If “on” and “off” do not work, try “True” and “False.”

1. From *X Window Systems User’s Guide, Volume 3*. O’Reilly, Quercia, and Lamb.

Permitting Error Messages

If you use an invalid widget hierarchy in a resource specification, X fails to complete the specification, but does not print an error message. When testing new customizations, adding the resource specification

```
*StringConversionWarnings: on
```

lets you see any error messages. You can also specify a client, like

```
axe*StringConversionWarnings: on
```

if you only want to see error messages for that client (in this example). Similarly, to permit error messages for all but a few clients, you can add the lines

```
*StringConversionWarnings: on
xterm*StringConversionWarnings: off
axe*StringConversionWarnings: off
```

These specifications allow error messages for all clients except xterm and axe.

The .Xresources File

Recall that X reads .Xresources directly, and this file contains only resource specifications. The system copy of this file is kept in the directory **/usr/lib/X11/xinit** as **Xresources** in most of the networks administered by Information Technology.

In .Xresources an exclamation point (!) indicates a comment. Following the exclamation point, X ignores any text on the same line, but it allows you to put explanatory notes into the file. You can add as many comments as you like.

Customizing On-The-Fly: The xrdb Command

This command gives you access to the X Resource Database—also called the RESOURCE_MANAGER—which contains all of your current X resource specifications. You can make changes to your X configuration immediately (without having to login and/or start a new X session). This feature allows you to test changes to see their effects, but remember that your new resource specifications only affect subsequent clients.

Some of the more useful options follow; for information on other options, consult the manual pages or enter

```
prompt% xrdb -help
```

-backup *suffix* — Use this option with the **-edit** *filename* option. It saves the contents of the file given with **-edit** *filename* in a backup file called *filename.suffix*, and then saves the current resource specifications in the file *filename*.

-edit *filename* — Saves the current resource specifications in the file given by *filename*. You may want to use the **-backup** *suffix* option with this option.

-load *filename*	Loads new resource specifications from the file given by *filename*. Note that this option erases all current resource specifications, so you may want to use **-merge** instead.
-merge *filename*	Merges new resource specifications from the file *filename*, but does not erase the current specifications.
-query	Lists all the current specifications, in alphabetical order; this option is very useful. You may need a scrollbar if you have many specifications that fill up your xterm.
-remove	Removes (erases) all resource specifications. Unfortunately, you cannot remove just one resource specification.

Because of the way the **-remove** operation works (it erases all the resource specifications), you cannot remove just one specification. Instead you must save the current specifications with **-edit**, edit the file and delete the unwanted resource line or lines, and then use the **-load** option.

For example, to remove the specification “xclock*mode: analog”, enter

```
prompt% xrdb -edit currentX
```

Then edit the file currentX and delete the line containing xclock*mode. Finally, enter

```
prompt% xrdb -load currentX
```

to load the altered file of resource specifications.

If you do not give a filename, xrdb assumes you want to type in resource specifications directly from the keyboard. Enter each resource specification, followed by RETURN. When you have entered the last specification, type CTRL-D to confirm them. To abort the operation entirely, type CTRL-C.

If you do not specify an option, xrdb assumes that you want to use the **-load** option. If you only want to add one or more resource specifications, enter

```
prompt% xrdb -merge
```

and then enter the resource specifications you want to add. The default to **-load** can cause problems if you forget to use **-merge** because xrdb will erase all but the new resource specifications! If you accidentally forget **-merge**, type CTRL-U instead of RETURN to abort the command.

If you just enter

```
prompt% xrdb
```

then xrdb assumes you want to erase all current specifications and make new ones from the keyboard. If you accidentally enter just “xrdb”, you should just type CTRL-C to abort.

Practice Customizations

Suppose you want your xterm windows to open with a smaller font. First use **xlsfonts** to check the available fonts, then change the appropriate resource specification

```
xterm*font: 6x13
```

to select a new font (6x13, for example).

Finding X Resources Names

X provides a program that can be used to determine the X resource hierarchy in X applications that use the Athena widget set (which includes most applications normally used). This program is called **editres**. You can run the program by typing

```
editres &
```

To query the resource hierarchy for a particular X application, start that application and choose “Get Tree” from the Commands Menu in **editres**. The cursor will change to a crosshair (+). Left click once on the application you want the resource hierarchy for.

The bottom part of editres will have a left to right graphical depiction of the X resource hierarchy for the selected application. If this information is larger than area it is displayed in, you’ll need to increase the window size using your window manager - editres does not have scrollbars.

Next, select an interesting looking widget name somewhere on the right edge of the bottom screen with the left mouse button. It should highlight. From the Commands Menu you can now choose “Show Resource Box.” This will display all associated resource names. Unlike editing your .Xresources, you can update the X resources of running applications with editres using the resource box.

Customizing with Commands

In this section you will learn about X commands and how to use them to customize your X display.

X Commands

UNIX reads X commands, which are then executed by your host just like any other user command. While most X commands start new clients, some commands control aspects like display access (xhost) and the screen saver (xset).

For an example of an X command, recall that inside an xterm you enter the user command

```
prompt% ls
```

to list the contents of a directory. You can also enter the X command

```
prompt% axe &
```

to start a new aXe window. The ampersand (&) tells UNIX to execute the command in the background[1] and continue reading input from the xterm.

Just as user commands permit options (specified with a dash), X commands have their own command options. Continuing the example, entering

```
prompt% ls -l
```

1. Background processes are discussed in the document *Introduction to the UNIX Operating System.*

lists a directory in long format. The table below describes several X commands, some more useful than others. Since X commands are read by UNIX, you can enter them directly in an xterm. That way, you can try several options with one command to find the combination you like best.

X Command	Description
maze	Creates and solves a maze
oclock	Analog clock
puzzle	Numbered-tile puzzle
xbiff	Mailbox indicator
xcalc	Desktop calculator
xclock	Analog or digital clock
aXe	A text editor
xeyes	Eyes watching the mouse
xfd	Font displayer
xload	Graph displaying a computer's workload
xlogo	Draws 'X' logo
xman	Read manual pages
xterm	xterm terminal emulator

Some of the X commands have special options (besides the usual ones in Subsection 1). For example, entering xclock displays an analog clock, but entering xclock -digital & produces a digital one. You can find out more about these X commands and their individual options by consulting the manual pages.

X Command Options

When starting up a new X client, often command line options may be specified. Some of the options require an argument. Depending on the option type, the argument may be a number, color, font, name, title, etc.

One command can use several options at once. For example,

xterm -g 80x30+250+280 -fn 6x12 -title "XTERM WINDOW"

starts an xterm and includes three command options.

Option (abbr.) arg	Description
-background (-bg) *color*	background color[a] (usually white)
-bordercolor -bd) *color*	border color[a](usually black)
-borderwidth (-bw) *number*	window border width in number of pixels[b]
-display (-d) *display*	display server to send client output

Option (abbr.) arg	Description
-font (-fn) *string*[c]	font for text windows like xterm
-foreground (-fg) *color*	foreground color[a](usually black)
-geometry (-g) *geometry*	initial size and location of each window
-iconic (-i)	start client as icon
-name (-n) *string*[c]	name for resources
-reverse (-rv)	reverse foreground and background colors
+reverse (+rv)	do not reverse foreground and background colors
-title (-t) *string*[c]	title for title bar; for an xterm, use -T
-xrm *string*[c]	*string* is a resource parameter

a. When using a monochrome screen, the only available colors are black and white. Using a different color may produce a white-on-white or black-on-black window, which makes the window appear entirely white or entirely black.

b. A pixel is a picture element on the screen. It is the smallest displayable element, and looks like a small dot when displayed in a contrasting color to the pixels around it. Monitor resolution is generally measured in pixels (e.g., 1280x1024).

c. *string* does not require quotes unless it contains blankspace. For example, -title North and -title “North Site”.

The -font Option

X has many available fonts; to list them, enter

```
prompt% xlsfonts
```

You may need a scrollbar since the many font names will fill up your window. To view a particular font, if *fontname* represents the name of the font you want to view, enter

```
prompt% xfd -fn fontname &
```

To find a font that has a desired appearance, there is a font display utility called **xfontsel**. You can invoke it by typing:

```
prompt% xfontsel &
```

The -geometry Option

The **-geometry** *geometry* option specifies a window’s initial size and location. *geometry* has four parts: *width*, *height*, *x offset*, and *y offset*. For icons or graphics windows, *width* and *height* give the window size in pixels. For text windows, the size is given in characters and lines.

The offsets specify the starting position of the window, relative to the edges of the display. Both offsets require a plus (+) or minus (-) sign. The plus sign refers to an offset from the top corner or left edge, and the minus sign refers to an offset from the bottom corner or right edge.

You do not have to give a complete *geometry* argument. In this case, X consults twm for the rest of the geometry. If it cannot find one, it lets you finish the geometry (by sizing or positioning the window with the mouse). You can give just a size (*width* and *height*) or just a location (*x offset* and *y offset*). However, if you give *width*, you must give *height*, and similarly the offsets must be complete.

The easiest way to get a windows geometry exactly how you want it is to use your window manager to size and position the window, then type:

```
prompt% xwininfo &
```

in an xterm. When the cursor changes to a crosshair, left click on the window for which you want the geometry information. The last piece of information listed is the geometry of the window.

The -title Option

The **-title** option lets you specify the client's title bar string. That way you can distinguish between different instances of the same kind of client.

Do not confuse the **-title** option with the **-name** option. **-title** simply declares the title in the title bar; it does not determine which resource specifications X uses. **-name** tells X to use a certain set of resource specifications. See the section, *Customizing with New Resource Names*, for a discussion of the -name option.

The -xrm Option

The **-xrm** option lets you make a resource specification rather than putting that specification in a start-up file. In other words, you can preempt the normal specification when you enter a command. This preemption only takes place for the command issued with the **-xrm** option. You can use **-xrm** several times with one command, but you must use it for each different resource specification. For an example, entering

```
prompt% axe -xrm “axe*iconX:30” -xrm “axe*iconY:50”
```

gives aXe a specific icon geometry.

xterm Command Options

The xterm command has many command options which affect its initial appearance; many of them come from the VT Options Menu. For example, the **-sb** option starts the scrollbar, and **-vb** turns on the visual bell. The remainder of this subsection discusses a few xterm option peculiarities.

For a brief description of xterm command options, enter

```
prompt% xterm -help
```

For a complete discussion of the xterm command options and resources, consult the on-line manual pages (**man xterm**).

Abbreviation Constraints

A few command options warrant mention, primarily because the regular abbreviations do not work. They appear below.

-n *string*

You specify the icon string with this option. It saves you from using a long **-xrm** option like

```
prompt% xterm...-xrm "*iconName:string
```

This option is *not* an abbreviation for **-name** which tells X to use a resource name.

-name *string*

This option instructs X to use the resources under the resource name *string*. See page 19 for more information on **-name**. You can not use the abbreviation **-n** for this command option.

-title *string*

This option functions just as expected, indicating the string for the xterm title bar. You should notice that you can not abbreviate **-title** to -t. Instead use the **-T** abbreviation for **-title**. The **-t** option tells X to start the xterm in Tektronix 4010 mode.

The -e Option

This powerful option lets you start an xterm and, when the new xterm starts, immediately execute a command. Normally you do not employ this option, but the subsection *twm Functions* in this document demonstrates how it produces menu-selectable clients.

Most importantly, *the -e option, if used, must be the last option in the xterm command options*. That is, you cannot put another xterm command option like -name or -i after the -e.

The syntax of this option appears below:

```
prompt% xterm...-e command [arguments]
```

command can be any user or X command, and if that *command* requires any of its own options or arguments, they appear where [*arguments*] appears.

The xset Command

This command lets you set some X preferences like bell volume, mouse speed, auto-repeat, and key click. Only a few of the options appear below; consult the manual pages for more information or enter

```
prompt% xset
```

Option	Abb.	Parameters	Description
bell	b	[on/off]	Controls bell: turns it on/off or sets volume (%), pitch (in hertz), and duration (in milliseconds). With no parameters, the system defaults apply.
key click	c	[on/off] [c [volume]]	Turns key click on/off or gives volume (%). With no parameters, the system defaults apply.
mouse	m	[[acceleration][threshold]]	Determines mouse *acceleration* and *threshold*. When you quickly move the mouse more than *threshold* pixels (dots), it accelerates to *acceleration* times as fast as normal. That means you can move the cursor accurately and slowly, or jump across the display with a flick of the wrist. To restore normal mouse operation, use default.
query	q		Lists current X preferences.
auto-repeat	r	[on/off]	Turns auto-repeat on or off.
screen saver	s	[[length] [period]]	Tells X how long (in seconds) to wait before activating the screen saver. Other parameters include blank or noblank to clear the screen or show a c/changing 'X' logo, and on or off.

The xsetroot Command

With the **xsetroot** command, you can change aspects of the root window, such as the background pattern and root window cursor. Like the xset command, you can get a list of options by entering

```
prompt% xsetroot -help
```

A few of its options appear below. The section, *Special Topics*, explains how to make bitmaps, pick cursors, and make your own cursors.

def — Resets the root window to its default settings (grey background and 'X' cursor).

cursor *cursorname maskname*

Specifies the root window cursor as a *cursorname* and *maskname*.

cursor cursor_name *cursorname*

Specifies the root window cursor as one of the standard cursors.

bitmap *bitmapname*	Uses the bitmap in the file *bitmapname* as the background pattern.
gray	Uses the normal gray background pattern.

NOTE: Most X servers on Sun hardware only support 256 nultaneous colors. f your background image uses 240, here won't be any left for other pplications. These applications may complain or fail to run if this is the case.

Using xv to put an Image in the Background

If you want to use an image for your background, the **xv** command, available on IT systems, is capable of this task. The syntax is:

```
prompt% xv -root -quit filename
```

xv supports most common Unix image formats. For help or more information, type:

```
prompt% man xv
```

or

```
prompt% xv-help
```

Notes About Color

When working at a color display server, you can choose from a wide variety of colors when specifying the background, border, and foreground colors. Enter

```
prompt% more /usr/lib/X11/rgb.txt
```

for a list of the available colors. The output includes three columns of numbers indicating the mix of red, green, and blue; you only need to read the color names. Due to color generation methods, your display server may not be able to produce every color on its screen at the same time. Normally, Information Technology X displays can handle up to 256 colors.

Notice that the .Xresources and .Xsetup.twm files do not contain any color specifications. You can give color resource specifications and command options if you like. However, twm has its own color features, and sometimes they create conflicts with X. For this reason, your color choices may not work as expected. You may enjoy more success if you modify your .twmrc[1] file or use the command option for your color choices.

The .Xsetup.twm File

Within this file you tell X to open windows and specify attributes like their sizes, locations, and border widths. The contents of your default file appear below, followed by a brief explanation.

Note that in UNIX, the hash sign (#) indicates a comment. Unlike the .Xresources file, an exclamation point does *not* signal a comment.

The system copy of this file is kept in the directory /usr/lib/X11/xinit as Xsetup.twm in most of the networks administered by Information Technology.

If a pound sign (#) precedes a command, this makes the rest of the line into a comment. Skipping a command this way is called commenting out the command.

1. See the section, *Customizing twm*.

Remember that the ampersand (&) tells UNIX to execute a command in the background and continue reading additional commands. When customizing you should add the ampersand after each X command you change or add, except if the last process is changed, in which case the ampersand should not be added. It is the final process that executes in the foreground during your login session; when this process terminates, your X session ends. If it is put in the background, your X session will immediately terminate!

The two commands explained below are commands that you should not change. They are vital to starting X and twm successfully.

twm &	starts twm. twm first looks for the file .twmrc (discussed in the section, *Customizing twm*) in your directory. If twm cannot find .twmrc it reads the default system .twmrc file.
$SHELL	starts your window properly.

You may modify the other commands and add ones of your own. Observe that these commands are all X commands, and most of them have an ampersand at the end. A few commands are described briefly.

xhost	allows any specified computer to send its output to your display server.
xset	sets the bell and key click to the system defaults, then sets the screen saver to wait ten minutes (600 seconds) before activating.
xclock	places an analog clock in the upper-right hand corner of the display.
xload	displays a load meter (indicating how busy your host is).
xbiff	puts the mailbox indicator next to the clock.
xterm	starts the large window on the left-hand side of the display.

Customizing On-The-Fly: Direct Entry

Since UNIX reads X commands, you can enter them from an xterm and see the results immediately. Just enter the command and any options. When you discover a desirable command-option combination, you can add that line to your .Xsetup.twm file. Do not forget to use the ampersand (&).

Customizing with New Resource Names

One important concept involves using one kind of client but giving it different resource names. This trick lets you use the same kind of client (several windows, for example) while giving each of them different attributes. Your console provides the best example.

The Console as an Example

While your xterm and console both function like an xterm terminal emulator, they have markedly different appearances. Compared to your xterm, the console has a different geometry, a smaller font, and an automatic scrollbar. While these attributes could be specified as a long set of command options, instead they form a new set of resources, separate from the xterm resources. These new resources fall under the name of console.

The file .xinitrc executes the command

```
xterm -name console
```

This command tells X to start an xterm terminal emulator, but the -name option tells X to use the resource named console instead of the default resource named xterm.

The different resource specifications under console and xterm give the windows their different appearance. The new resource name (console) creates a set of resource specifications completely independent of the standard client resource (xterm).

The -name Command Option

By using the -name command option, you can start the same kind of client but give each window of the client a totally different appearance.

1. Pick the client which should have multiple resource names.
2. Identify the resource specifications to be changed, under the normal name.
3. Choose your own resource name, myResource, for example.
4. Using your new resource name, add the new resource specifications to the X Resource database with xrdb.
5. Start a client with the -name command option and check its performance.
6. If satisfied, save your changes in your .Xresources file.

X normally uses the resource name for a window's title, but recall that you can specify a title with the -title option. -title only declares the string that X puts in the title bar; it does not determine the resource name.

Customizable File Summary

The three files (.Xresources, .Xsetup.twm, and .twmrc) which you can customize appear in the table below. Hopefully this table clarifies the use of each file.

File:	.Xresources	.Xsetup.twm	.twmrc
Read by:	X	UNIX	twm
Contents:	X resource specifications	X and user (UNIX) commands	menus, window functions, mouse and key bindings
Syntax:	widget hierarchy bindings	commands and command options	variables and lists

File:	.Xresources	.Xsetup.twm	.twmrc
Comments:	exclamation point (!)	hash sign (#)	hash sign (#)

Customizing twm

This section explains how to customize aspects of twm. twm handles menus, window operations (moving, resizing, etc.), and mouse button functions.

The twm Start-up File:

twm reads a start-up file called .twmrc (for “twm Resource Configuration”), which specifies menu contents and mouse button functions. .twmrc requires a syntax different from both UNIX and X.

The system default .twmrc can be found under /usr/lib/X11/twm/system.twmrc. As you read the rest of the material, this listing will prove a handy reference.

Note that in.twmrc, a hash sign (#) indicates a comment line.

Testing Changes: Sourcing

This section explains how to test your twm customization. Before you make any changes, you should backup your existing file, just in case your changes do not work. For example, enter

```
prompt% cp .twmrc twmrc.old
```

before editing your .twmrc file.

When you are ready to test your customized .twmrc, activate the twm Window Manager Menu and select Source .twmrc. This instruction tells twm to read your .twmrc file, and your changes become effective. You can perform this operation as many times as you like, which enables you to customize twm incrementally. Just follow this procedure:

1. Backup your current, error-free .twmrc.
2. Make a few changes to your .twmrc.
3. Select Source .twmrc from the twm Menu.
4. Check your changes for proper results.
5. Make any necessary corrections.
6. Repeat this procedure until your twm customization is satisfactory.

Restoring the Default .twmrc

If you run into trouble at any time during your experimentation with .twmrc customization, you can restore the system default .twmrc until you have figured out what is wrong with your own version. To restore, copy the system default .twmrc using the following syntax:

```
getxfiles -R -S -w
```

twm Instructions

File Structure

twm employs several types of instructions, and generally the .twmrc file consists of three sections:

1. variables
2. bindings (mouse buttons, title bar buttons)
3. menus

You are not required to divide .twmrc into three explicit sections, and the default .twmrc is not divided as such. However, sectioning .twmrc to some extent can make customization and debugging easier.

Variables and Arguments

twm maintains a large set of predefined variables, which you manipulate to customize twm. Certain variables simply turn an option on or off, and they are called *toggle variables*. Other variables require an argument, usually a number or a string, and some variables take more than one argument.

A variable and its argument(s) are separated by *whitespace*. Whitespace consists of spaces, TAB characters, and RETURN characters.

Variable names are case-insensitive, meaning that twm does not distinguish between capital and lowercase letters. The capitalization simply makes them easier for you to read. For example, DontMoveOff, Dontmoveoff, and dONtmOVEofF refer to the same variable.

Strings

Enclosed in double quotes (“), strings *are* case-sensitive. Hence, capitalization can distinguish otherwise identical strings. Thus, “X applications” and “X Applications” are different strings.

A string usually corresponds to a window or client name. A string matches a window if the string matches the client name, resource name, or class name. An example may clarify this concept.

Consider the twm variable Icons, which takes a name and associates it with a particular icon.

```
Icons {“xterm” “xterm.icon”
“console” “xlogo32”
```

“XTerm” “plaid”}

Here, the three names are a client name (xterm), a resource name (console), and a class name (XTerm). If you start a terminal emulator with

```
prompt% xterm &
```

then twm uses the client name, xterm, to pick the icon. Thus, twm selects the xterm.icon in this case. However, you can indicate a different resource name with the command option. If you enter

```
prompt% xterm -name console
```

then twm starts an xterm client but uses the console resources, and thus twm picks the xlogo32 icon. Finally, if you enter

```
prompt% xterm -name fickle
```

then twm starts an xterm client and looks for a resource named fickle. If twm cannot find a resource, twm looks in the corresponding class. Since fickle does not exist, twm uses the class name for the xterm client, and this class name is XTerm. Therefore, twm chooses the plaid icon.

Lists

When a variable permits several arguments, these arguments form a **list**. Braces, {}, enclose a list, which may include variables and strings. Items within a list are separated by spaces, TABs, or RETURN characters.

This feature lets you format your lists for easy (or difficult) reading.

For example, twm treats the three lists below identically.

{“xclock” “xbiff” “dclock”}

{“xlock”
“xbiff”
“dclock”}

{ “xclock”
“xbiff” “dclock”}

Sometimes the items in a list are grouped in pairs or triplets. For example, the variable Icons lists names and their corresponding icons:

Icons {“xterm” “xterm.icon”
“Elm” “letters”
“aXe” “keyboard16”}

Bindings

The second section of the .twmrc file produces title bar buttons and mouse button functions. Function keys can also be bound to functions. Do not confuse twm bindings with resource specification

bindings. twm bindings link buttons and keys to twm functions; resource specification bindings trace a path down an X widget hierarchy.

See the section, *twm Functions*, for information on twm functions, which are not discussed here but are shown to demonstrate proper binding syntax.

Title Bar Buttons

You can add title bar buttons to all windows with this binding. The syntax for title bar buttons appears below.

Left/RightTitleButton " bitmapname " = function

Left/RightTitleButton indicates where the title bar button should appear. *bitmapname* tells twm the name of the button's picture, and *function* is a twm function. See the section, *Special Topics*, to read how you can select and create bitmaps. Title bar buttons appear in the order they are declared, from left-to-right for LeftTitleButton and right-to-left for RightTitleButton.

Mouse Buttons

Mouse buttons take a special syntax, which indicates the mouse button, any special keys (modifiers), the cursor location (context), and the corresponding twm function, respectively.

Button = modifiers : context : function

Button1 corresponds to LEFT, Button2 to MIDDLE, and Button3 to RIGHT; twm also accepts Button4 and Button5, but most hardware (mice and X mouse drivers) does not have these buttons.

The modifiers are c, m, and s (CTRL, META, and SHIFT, respectively). You can indicate a combination of modifiers with a vertical bar (|). context specifies the cursor location, such as window, title, icon, root, frame, iconmgr, or all. If several contexts are valid, use a vertical bar (|) to delimit them.

Function Keys

Function keys take the same syntax as mouse buttons, except that the key is surrounded by double quotes (")

Key = modifiers : context : function

Menus

The third part of .twmrc defines menus. A menu begins with a header, followed by a list of items and functions.

```
Menu "menuname
{
 string1 function1
 string2 function2
```

```
    string3 function3
    etc.
}
```

Some Useful Variables

This subsection describes several useful twm variables. You can consult the manual pages for more information.

DontMoveOff	When employed, this variable prevents windows from being placed off the display. If you use a program which requires a very large window, it may not fit properly unless you let part of it hang off the edge of the display
NoTitle	Suppresses title bars for windows in the display. For example, you might suppress the title bar on xclock. MakeTitle may be used with this option to force titlebars to be put on specific windows.
RandomPlacement	When this variable is activated, twm positions new windows randomly, rather than making you place them. If a window is started with the -geometry option, then that geometry applies.
DefaultFunction *function*	This variable tells twm which function to execute when you press a key or mouse button which does not have a function bound to it. Usually *function* is something like f.beep or f.nop (the terminal beeps or does nothing).

twm Functions

Up to this point, you have seen various twm functions in examples and exercises, but you have not received a formal introduction to them. This subsection describes several twm functions, but you should consult the manual pages on twm for a complete list.

f.beep	Rings the bell (beeps).
f.circledown	Lowers the highest window which overlaps another.
f.circleup	Raises the lowest window which is overlapped by other windows.
f.deiconify	Turns an icon into a window.
f.destroy	Kills a client. Not a recommended means of stopping a client; try to exit the client normally.
f.focus	Directs all input to the selected window, regardless of the cursor position. You can move the cursor off the window but still type to it. Use f.unfocus to stop this function.

f.forcemove	Lets you move a window and allows the window to go off the display. This function can be useful if you have large windows.
f.iconify	(De)iconifies a window or icon.
f.identify	Displays a small box listing information like title (although twm calls the title “Name”), resource name, and class name. Also displays the window geometry (not icon geometry).
f.lower	Lowers a window or icon (sends it to the bottom of the stack of overlapping windows).
f.menu	Activates a menu or submenu named by *string*. A submenu can have its own submenu.
f.move	Moves a window.
f.nop	No operation. Usually used to add a blank line to a menu.
f.quit	Stops twm.
f.raise	Raises a window or icon (brings it to the top of the stack of overlapping windows).
f.refresh	Redraws the display.
f.resize	Resizes a window or icon.
f.showiconmgr	Forces twm to display the icon manager window or windows.
f.title	Adds a centered title, set off by horizontal lines, to a menu.
f.unfocus	Turns off focused window and lets the cursor position highlight windows. See f.focus.
f.winrefresh	Refreshes a single window.

The f.function Function

twm lets you create a new window function by combining existing built-in functions. Calling a new function uses a syntax like f.menu:

f.function *string*

where *string* gives the name of the new function. Usually new functions appear after menus, at the very end of .twmrc. To define a new function, use the form:

Function “*string*” {*function(s)*}

MoveDelta and the f.deltastop Function

The MoveDelta variable and the f.deltastop function let you create an “or” function. Depending on whether you click or drag, you determine which built-in function takes place. Normally f.deltastop appears between two built-in functions in a custom-made function.

MoveDelta *number*

Function “*string*” {*function1* f.deltastop *function2*}

When performing a custom function, twm executes *function1* unless you drag the cursor more than *number* pixels (dots). If you do, then twm executes *function2*.

f.deltastop may not always behave as you expect because some built-in functions act as soon as you click, while others require dragging. In addition, the order of the built-in functions may affect the behavior of the new function. Generally, you need to test your custom function, especially if it uses f.deltastop, to ensure that it performs correctly.

The f.exec Function

Often abbreviated by an exclamation point (!), the f.exec function executes an X or UNIX command. In your default .twmrc this function enables almost every item in the X Applications menu. Without this option, that would not be possible.

You can use this function when you want to add another X client or UNIX command to those accessible from your twm menus.

Special Topics

The Monochrome and Color Variables

These variables prepare your display depending on your monitor type. You can have both variables in .twmrc and twm consults the proper one for display characteristics. These variables take a list of other variables which specify different colors for borders, backgrounds, foregrounds, and menus. The .twmrc file may only have one Monochrome and one Color list.

For example, consider the following Monochrome variable:

Monochrome

{

MenuTitleForeground “white”

MenuTitleBackground “black”

BorderTileBackground “white”

BorderTileForeground “white”

BorderColor “black” “console” “white”

“aXe” “white”

```
IconBorderColor “white” {“aXe” “black”}
TitleForeground “black” {“console” “white”}
TitleBackground “white” {“console” “black”}
}
```

First of all, these specifications apply to a monochrome monitor. A similar set of specifications could fall inside a separate Color variable. Inside the list, the first two variables make menu titles appear in reverse-video.

The two BorderTile... variables determine the colors used in unhighlighted window borders. twm uses a tile pattern of two colors, and because both colors are white, unhighlighted windows have a white border rather than a gray one.

Next, BorderColor tells twm to display a black border for highlighted windows, with the exceptions of console and aXe windows, which highlight with a white border. Since highlighted and unhighlighted console and aXe windows employ white borders, they do not appear to highlight.

The next variable gives icons a white border, except the aXe icon, which displays a black border. Finally, the last two variables specify a normal black-on-white title bar, but the *console* gets a reverse-video title bar.

Icon Region

twm lets you declare an area of the display for icons with the IconRegion variable. This variable has the following form:

```
IconRegion “ geometry vgrav hgrav cellwidth cellheight
```

geometry is a regular geometry string. twm divides the icon region into a rectangular grid, and each cell within the grid has dimensions of *cellwidth* by *cellheight* pixels.

vgrav and *hgrav* give the vertical and horizontal gravities, respectively. *vgrav* is either North or South, which fill the icon region from top-to-bottom (North) or bottom-to-top (South). Likewise, *hgrav* is East or West; East fills the region from right-to-left, and West from left-to-right.

You can specify more than one IconRegion. In this situation, twm uses the first icon region until it becomes filled, then proceeds to the next one, and so on.

The bitmap Program

The bitmap Program lets you draw your own icons, cursors, and cursor masks, each of which is represented by a bitmap. To start bitmap

```
prompt% bitmap filename WIDTHxHEIGHT &
```

where *filename* is the name of the bitmap file (if it already exists, bitmap will read that file). *WIDTHxHEIGHT* give the size of the bitmap in pixels (screen dots). If you do not specify the size, bitmap uses 16x16.

The following example creates a file called piano that is 80 pixels wide and 65 pixels high. If piano already exists, bitmap reads that file and uses its dimensions.

```
prompt% bitmap piano 80x65 &
```

bitmap is an exceptionally easy-to-use program and has excellent documentation in the manual pages. For these reasons, you should try bitmap on your own or read the manual pages for more information.

X and twm include several standard bitmaps; for a list of standard bitmaps, enter

```
prompt% ls /usr/include/X11/bitmaps
```

/usr/bin/X11/bitmaps is a directory containing the standard bitmaps. You can view its bitmaps, use them for icons, cursors, and masks. They can also be useful as a starting point for creating your own bitmaps.

Choosing Icons

twm lets you pick icons for different applications, and the bitmap program, explained in the previous section, lets you create new icons. The toggle variable ForceIcons overrides any default icons so that if a client requests an icon, you can specify a different one.

IconDirectory tells twm in which directory to search for icon bitmaps. By default twm looks in the directory /usr/include/X11/bitmaps, but if you create a directory of your own icons, you can provide a different IconDirectory variable.

Finally, Icons lists pairs of windows and icons. A window may be expressed as a client name (the usual method), a resource name (if you use the X command option), or a class name.

As one example, the default .twmrc contains the lines

```
ForceIcons
IconDirectory “/usr/include/X11/bitmaps”
Icons
{
 “xterm” “xterm.icon”
 “Elm” “letters”
 “gnuemacs” “keyboard16”
 “Emacs” “keyboard16”
}
```

The first line makes the subsequent Icons variable override any default icons. The next line gives the normal icon directory. Finally, the Icons variable includes five windows and their icons; note that the last three windows use the same icon. Also, Elm and Emacs are both class names.

You can also give a specific directory path for a particular icon. To do so, just include the entire path with the icon name. You can use the tilde character (~) to indicate your home directory, or you can use ~*userid* to indicate someone else's home directory.

In addition, twm supports three default icons: :xlogo or :iconify (they are the same icon), :resize, and :question. They are always available.

For example, consider this .twmrc segment:

```
ForceIcons
IconDirectory "~fred/pictures"
Icons
{
 "aXe" "horse"
 "xterm" "xterm.icon"
 "xman" "~/my.icon"
 "gnuemacs" ":resize"
}
```

Now twm looks for icons in another person's (fred's) home directory. Fred has a directory called pictures, and it contains an icon called horse, so twm finds horse for the icon.

For the xterm icon, twm searches pictures and does not find anything. So twm then looks in its default directory /usr/bin/X11/bitmaps and finds xterm.icon.

Next twm looks in your home directory for my.icon and uses it for xman. Lastly, twm uses its standard icon :resize for the gnuemacs icon.

Picking Cursors

Along with icons, you can specify the cursors you want twm to use in different parts of the display. The cursor definitions should appear before any bindings, menus, or functions. For example, the cursor defaults are

```
Cursors
{
 Frame "top left arrow"
 Title "top left arrow"
 Icon "top left arrow"
 IconMgr "top left arrow"
 Move "fleur"
 Resize "fleur"
 Menu "sb left arrow"
```

```
Button “hand2”
Wait “watch”
Select “dot”
Destroy “pirate”
}
```

You can find a complete list of standard cursor names by entering

```
prompt% cd /usr/include/X11
prompt% more cursorfont.h
```

This action lists the available cursors and produces output that looks like:

```
/* $XConsortium: cursorfont.h,v 1.2 88/09/06 16:44:27
jim Exp $ */
#define XC_num_glyphs 154
#define XC_X_cursor 0
#define XC_arrow 2
#define XC_based_arrow_down 4
#define XC_based_arrow_up 6
```

To specify a cursor you only need the cursorname, without the XC or the ending number. Also, in your specification, the underlines (_) should be replaced by spaces. For a display of the standard cursors, enter

```
prompt% xfd -center -fn cursor &
```

Creating Cursors

You can create your own cursors with bitmap but you must also make a cursor mask. The cursor and mask must have the same *WIDTH* and *HEIGHT*. To get an idea for what cursors and their masks look like, enter

```
prompt% xfd -center -fn cursor &
```

This command displays the standard cursors and their respective masks. Notice how the masks are usually solid black and form a silhouette of the cursor.

Since most cursors are larger than one pixel, and the actual pointer size is one pixel, the server needs to know which of the pixels in the cursor bitmap to assign this role to. This point is called the **hot spot**. Bitmap includes a “Set Hot Spot” command.

twm looks for cursors in the directory /usr/include/X11 unless you specify a different directory. twm does not support a CursorDirectory variable, so you must include the directory for each cursor and mask. When you specify a bitmap cursor, it requires a bitmap and a mask. Then the Cursor list includes a third argument which names the mask. For example,

```
Cursors
{
 Frame “top left arrow”
 Title “~/cursors/me” “~/cursors/me.mask”
 Icon “gobbler”
 IconMgr “~/cursors/fish” “~/cursors/fish.mask”
```

The Icon Manager Feature

If you find that you have too many icons floating around your display, or if you get too lazy to place your icons, twm provides a feature called the **icon manager**. The icon manager maintains a list of all your icons in a single location, which keeps your display uncluttered. Do not confuse the icon manager with the IconRegion previously discussed.

The icon manager lists all your clients, with a mark beside those that are icons. By clicking on a client in the icon manager, you (de)iconify that client. When the icon manager is running, twm does not display icons; this act of “disappearing” a window is called **unmapping**. An unmapped client continues to run, just like a iconified client, except that it has no icon.

The icon manager uses the following variables, as well as a few color specifications that can be found in the manual pages:

DontIconifyByUnmapping {}

Normally when the icon manager is running, twm does not display icons. With this variable, windows in the display become icons rather than disappearing by unmapping.

IconifyByUnmapping The icon is optional with this variable, which instructs twm to convert windows into icons by unmapping. Unmapping effectively causes a window to disappear, although the client keeps running. Using this variable without the icon manager can create a problem because you can render a client inaccessible.

IconManagerDontShow Clients do not appear in the icon manager list. Normally you use this variable to prevent rarely-iconified clients (like xclock) from occupying space in the icon manager. With value **no**, only clients given with IconManagerShow appear in the icon manager. Thus, the icon manager could run without actually handling any clients!

IconManagerFont *string* Specifies the icon manager’s font.

IconManagerGeometry *“geometry” columns*

Determines the geometry of the icon manager window. *columns* gives the number of clients in each row of the window. If *columns* is not given, it defaults to 1.

IconManagers {*icon-manager-list*}

This variable lets you create different icon manager windows for different kinds of clients. Each line in *icon-manager-list* takes the form:

"windowname" "geometry" columns

windowname identifies the windows for the icon manager, *geometry* gives the icon manager's geometry, and *columns* determines how many columns of clients appear. Unlike IconManagerGeometry, columns must be specified.

IconManagerShow Specifies client which must be displayed in the icon manager. Make certain that you use this variable if you use IconManagerDontShow without options.

NoIconManagers Prevents all icon managers from running.

ShowIconManager Tells twm to show the icon manager window or windows. You can also add the function f.showiconmgr function to a menu.

SortIconManager Instructs the icon manager to sort its clients alphabetically rather then adding them as they appear.

For More Information

This chapter covers basic twm customization, but many other twm variables (of varying usefulness) are available. A complete list and description of twm variables and functions appear in the manual pages. To learn more about twm, consult the on-line manual pages. Enter:

```
prompt% man twm
```

Recovering from Failed Customizations

Quite possibly you will make mistakes during your customization. In some cases you can simply edit the appropriate files, but in other cases you may not even be able to login! This chapter tells you how to recover from such frightening circumstances.

You need to use UNIX commands for copying, moving, editing, and deleting files. For information on these commands, consult the document, *Introduction to the UNIX Operating System*.

Tips Before Customizing

To make customization easier and less painful, the following information provides some foresight into customization. Realize that your changes may not work at first, and take the proper precautions.

Test On-The-Fly

You can try your X customizations immediately with the xrdb command and by entering X commands inside an xterm. With twm, you can use the Source .twmrc item from the twm Menu. That way you can fine-tune your customization before making it permanent. See previous sections in this document for more information on xrdb, X commands, or twm.

Save Your Previous Configuration

As proper insurance, before you save any new changes, copy your current X and twm files as backups (under different names than the normal start-up files). This prudence establishes a reference point from which to work, and you can abandon new changes if you do not like them.

Work in Steps

If you plan to make many modifications, do them in smaller steps. You can concentrate more carefully on each step and test the changes after each step. If an error occurs, you have a good idea of its location. When you successfully complete a step, make a backup before continuing.

Alter One File at a Time

Only modify one start-up file at a time. That way if your customization fails, you know which file to edit.

Consult Other Individuals

Most successful customization results from experience. If you are new to customization, ask for assistance from friends or Information Technology staff members. Keep in mind, however, that customization takes a low priority in terms of consulting issue importance.

Troubleshooting

Usually recovery involves identifying an error, correcting it, and testing the correction. A few possible explanations for errors follow.

- Check your syntax. Recall that the start-up files require a different syntax.
- Make sure you use the proper comment character. Make sure your files contain the proper information.
- If you suspect errors, use the line "*StringConversionWarnings: on" in your .Xresources file to instruct X to report error messages.
- When customizing twm, replace the line twm entry with the line twm -v entry in your .twmrc to allow twm to generate error messages. Change the line back when your customization is working.
- If some resource specifications in your .Xresources file do not seem to work properly, try using a class instead of an instance, and vice versa. For example, if "xterm*scrollBar: on" does not work, try the line "xterm*ScrollBar: on".
- If you think an error lies in your .Xsetup.twm, check your use of the ampersand (&) after most commands in your .Xsetup.twm.

Logging In without X

If your customization has significant errors, you may not even be able to login to recover it. Usually the error lies in .Xsetup.twm. You need to login without invoking this file, then start a basic X session. The following procedure accomplishes this task from a dedicated display server.

At a Silicon Graphics workstation:

1. At the X Start-up Display enter your username.
2. Type your password, but instead of completing the entry with RETURN, press the "F1" key.

At a Sun workstation

1. Before typing your name and password, select Failsafe Session from the menu items Options, Session.
2. Enter your userid and password as normal.

These steps let you login without reading any of your start-up files. You start the X session without any defaults and without a window manager. The mouse buttons do not work as usual because twm is not running. You have a single xterm window, although it has no title or title bar. This functions like your console window.

Next enter these commands to start a minimal X session.

1. `prompt%` xrdb -load /usr/lib/X/xinit/Xresources

   Loads the system default file.
2. `prompt%` twm -f /usr/lib/X/twm/system.twmrc &

   Starts twm.

Now you have begun a X session with the system default configuration. You can edit or replace the necessary start-up files. After making these changes, you can logout, then login to try them.

Problems or Question

If you have any questions or problems, contact the Consulting Center at 713-348-4983 or stop by 103 Mudd Lab. You can also e-mail **problem@rice.edu** or submit the problem at http://problem.rice.edu via the World Wide Web.