6 Window Managers

Strictly speaking, this belongs in the previous section because a window manager is a commonly used `X` client. However, they are sufficiently important, and behave sufficiently differently that they merit a section to themselves.

6.1 Why have window managers?

Basic `X` provides the facilities for windows to be resized, popped up and down, moved, stacked, partially covered etc. However, it makes no attempt to provide a common interface for these operations. It is not feasible for each `X` client to provide its own hooks for these operations. There would be little or no consistency between any of the clients written.

Instead, a special `X` client called a "window manager" is set up which "decorates" each window with a border and a title bar. These are all provided by the window manager and have the same lay out and features. By this means, the window manager provides the common interface for resizing, iconifying, moving etc. and passes the instructions on to the clients at a lower level.

Some means of accessing `X`, such as Vista-eXceed, can provide their own window management as part of the software running on the `X` terminal. Under these circumstances, no alternative window manager can be started up.

6.2 What window managers provide

The near-infinite scope of `X` to be customised will bite us again here. All of the descriptions given here are for very basic default settings. There is no reason to believe that the system administrator of any system has not changed them. A description of how to customise one of the window managers (`twm`) will be given in section 6.3.

There are a number of window managers available, and each has its own distinctive style of decoration. Figure 4 illustrates the decoration provided by the window manager that comes with MIT's X11R5 software: `twm`. Some

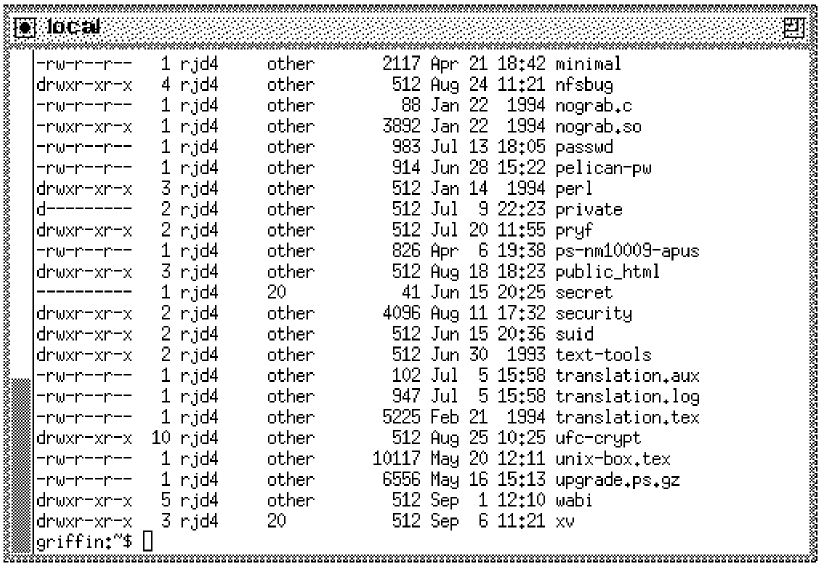

Figure 4: An `xterm` decorated by `twm`

others are illustrated in figure 5. Three of the most common are X11R5's Tab Window Manager (`twm`), Sun's OpenLook Window Manager (`olwm`) and the Open Software Foundation's Motif Window Manager (`mwm`).

Each of these sets of decoration provides facilities for resizing and iconifying the windows. For iconifying, simply click the left button on the relevant bit of the decoration. For `twm` use the button on the left-hand side of the title bar. For `olwm` use the only button on the title bar (marked with a downwards pointing arrow). For `mwm` use the button second from the right with a small dot in it.

To *de*-iconify an icon either single- or double-click the icon with the left button. The number of clicks is a function of the window manager, again.

To move a window, press the left hand button in the title bar and move the window to the desired location. Then release the button.

To resize a window the instructions vary more widely depending on the window manager:

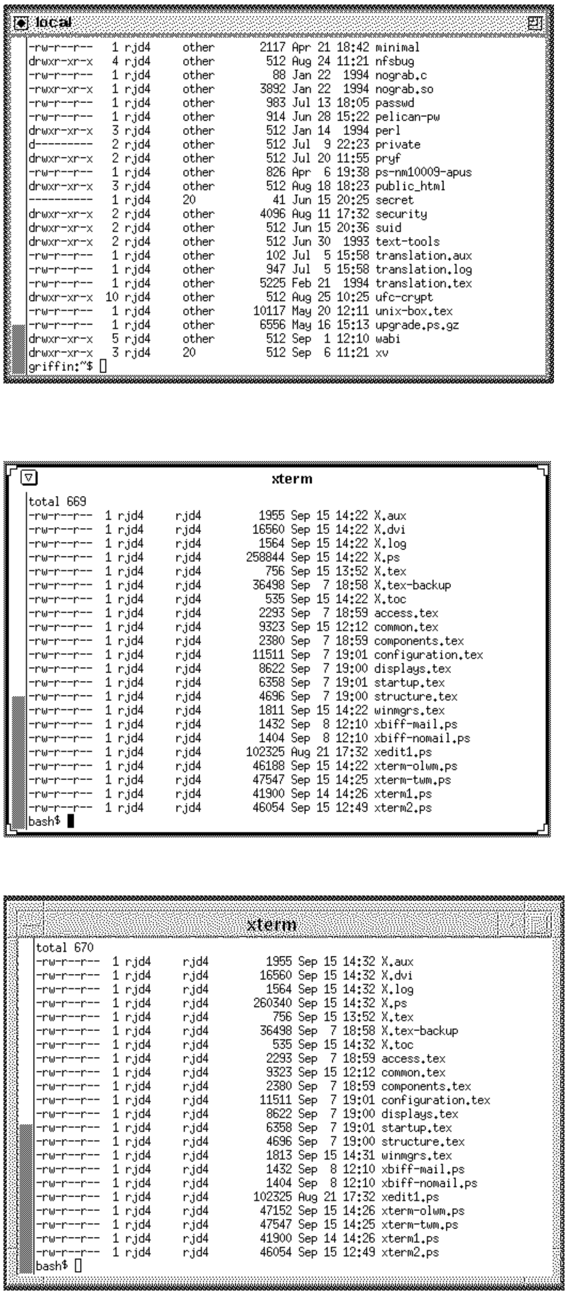

Figure 5: A collection of decorations:(a) `twm` (b) `olwm` (c) `mwm`

`twm` Press any button in the button at the right hand side of the title bar. With the button held pressed down, slide the pointer to an edge of the window. As the pointer touches the edge of the window it sticks to it and, as the pointer moves further out or back into the original window's space, will move the boundary with it. Release the button to define the new position.

`olwm` Press the left button on any of the "corner pieces" (designed to look like the corner pieces from photo albums) and, with the button held down, move the pointer to the position you want that corner to have.

`mwm` Press the left button on the edge or corner piece of the border. Keep the button pressed and move the pointer to the desired position.

The window managers can also provide another common interface. Each of them allows a menu to be pooped up from the title bar. Again, the exact mechanics depend on the window manager being used and how it is configured:

`twm` Nothing by default.

`olwm` Click the right button on the title bar's button. Don't hold the button down. This will reveal a menu that will persist when the pointer is moved away. Move the pointer over the desired menu item and click the left button.

`mwm` Click the left button on the left hand button on the title bar. This will cause a menu to appear and to persist after the button has been released. Move the pointer over the desired menu item and click the left button.

These menus provide a common interface to a wider variety of functions. The most important are entries to delete the window (often marked "quit"). These send an `X` request for the window to stop being displayed. Some window managers provide a more powerful option (often called "kill" or "destroy") which foribly remove access to the `X` server by that client.

6.3 How to customize twm

Here is an example of how to customize one of the window managers: **twm**. A user's configuration is given in a file called `.twmrc` in the user's home directory.

This is an (annotated) `.twmrc` file containing most of the commonly used features. The **#** character causes the remainder of the line to act as a comment.

```
# Fonts
TitleFont "-*-helvetica-bold-r-*-*-120-*-*-*-92-*-*"
MenuFont  "-*-helvetica-bold-r-*-*-20-*-*-*-p-*-*-*"
IconFont  "-*-helvetica-bold-r-*-*-14-140-*-*-*-*-*-*"
```

See section 9.6 for a description of how to interpret these `X` logical font descriptions.

```
# General characteristics
NoGrabServer
BorderWidth     2
RandomPlacement
```

`BorderWidth` determines how wide a border each window is given. It is measured in pixels. `RandomPlacement` causes each window to pop up at a random location unless it specifies (in its geometry) exactly where it wants to be placed. A common alternative is `UserInteraction` which causes the skeleton of the window to pop up under the pointer and to follow the pointer requiring the user to click with the left button to place the window.

```
# Colour scheme
Monochrome
{
    BorderColor          "black"
    TitleForeground      "black"
```

```
    TitleBackground      "white"
}

Color
{
    BorderColor         "slategrey"
    TitleBackground     "maroon"
    TitleForeground     "gray85"
}
```

This sets the colours for various components of the window decorations in the obvious manner. See section 9.5 for details of what colours are available.

```
# Special behaviour for certain clients

NoTitle
{
  "xclock"
  "xbiff"
}
```

The `NoTitle` option causes certain clients not to be given title bars because they would just get in the way and are unnecessary.

```
# Buttons

DefaultFunction f.menu "win_ops"

#Button = KEY : CONTEXT      : FUNCTION

Button1 =     : root         : f.menu "hosts"
Button2 =     : root         : f.menu "win_ops"
Button3 =     : root         : f.menu "tools"

Button1 =     : title|frame  : f.raiselower
Button1 = c   : title|frame  : f.iconify
```

```
Button2 =     : title|frame  : f.move
Button2 = c   : title|frame  : f.resize
Button3 =     : title|frame  : f.menu "frame_menu"

Button2 = m   : window       : f.move

Button1 =     : icon         : f.iconify
Button2 =     : icon         : f.move
```

This is the first complicated part of the configuration file.

First of all a default action is defined by `DefaultFunction`. If none of the subsequent instructions match what the user actually does, this is the instruction to be followed. The `twm` instruction `f.menu` means to pop up a menu, and the following string `"win_ops"` specifies which particular menu. The definition of a menu follows in the same file.

Three things are required to specify a button click: which button, what modifier keys were pressed on the keyboard at the same time, and where the pointer was pointing. The buttons of a mouse are typically numbered from left to right and are simply called "`Button1`", "`Button2`" and "`Button3`". The typical keyboard used with `X` has three "modifier" keys on the keyboard: "shift", "control" and "meta". These are indicated by the letters `s`, `c` and `m`. A choice of modifier keys can be offered with the syntax "`c|m`" meaning either the control key or the meta key. The location where the button was clicked is called the "context" and can be one of `root`, `title`, `frame`, `window` or `icon`. Again, a choice can be offered using the vertical bar character.

Once the button click has been identified, a `twm` function can be assigned to it. As has already been stated, `f.menu` pops up a menu specified by the next string. A list of these "`f.`" functions will be given on page 35 together with descriptions of what they do. Many of them are fairly obvious from their names.

```
# Menus

menu "frame_menu"
{
```

```
"Frame Menu" f.title
"Redisplay"  f.winrefresh
"Move"       f.move
"Resize"     f.resize
"Iconify"    f.iconify
"Zoom"       f.fullzoom
"Front"      f.raise
"Back"       f.lower
""           f.nop
"Quit"       f.delete
"Kill"       f.destroy
}
```

This is the first of the menus being defined by the configuration file. The syntax is fairly straightforwards. The `menu` command takes a string as the name of the menu. This will be the name by which `f.menu` can call it. It then takes a list of pairs. The first entry in the pair is the string to display in the menu and the second is the function to run if that item in the menu is selected. The only point of note is the antepenultimate entry which has the empty string as the name of the option and the "`f.nop`" function as the action. This performs no operation and is used to leave a blank space in the menu.

```
menu "win_ops"
{
"Window Operations"   f.title
"Refresh Screen"      f.refresh
"Refresh Window"      f.winrefresh
"Focus on Window"     f.focus
"Focus on Root"       f.unfocus
"Zoom Window"         f.zoom
"FullZoom Window"     f.fullzoom
"Source .twmrc"       f.twmrc
"Source .Xresources"  !"xrdb -load .Xresources&"
""                    f.nop
"Delete window"       f.delete
"Destroy Window"      f.destroy
""                    f.nop
```

```
"Diagnostics"        !"printenv|xless&"
"Resources"          !"xrdb -query|xless&"
""                   f.nop
"Quit X11"           f.quit
}
```

This is very similar in structure to the previous menu. The only new features are the commands starting with an exclamation mark. The syntax `!"`*command*`"` means to start up a subshell and run the command in it. The `xrdb` command is explained in section 9.3. The `printenv` command displays the window manager's environment variables. The `xless` command is a simple command that will pop up a window displaying the text being fed to it. *N.B. All the commands are backgrounded.*

```
menu "hosts"
{
"Hosts"   f.title
"Local"   !"xterm -name local -ls&"
"Apus"    !"xon apus xterm -name apus -ls &"
"Bootes"  !"xon bootes xterm -name bootes -ls &"
"Grus"    !"xon grus xterm -name grus -ls &"
"Ursa"    !"xon ursa xterm -name ursa -ls &"
}

menu "tools"
{
"Tools"         f.title
"Emacs"         !"emacs&"
"Xbiff"         !"xbiff&"
"Xcal"          !"xcal&"
"Xcalc"         !"xcalc&"
"Xclock"        !"xclock&"
"Xfontsel"      !"xfontsel&"
"Xman"          !"xman&"
"Xmh"           !"xmh&"
"Xrn"           !"xrn&"
}
```

These two menus are fairly obvious. Note the allocation of a name to the `xterm` depending on the host accessed. This will cause the window manager to use this name in the title bar rather than the default "xterm". It actually has a far more significant effect described in the section on resources (section 9.1).

Finally, we will describe the `f.`*functions* in some detail:

`f.delete` This sends a "polite request" to a client asking it to remove a window.

`f.destroy` This instructs the `X` server to close the display connection corresponding to the client that owns the window. This should only be used as a last resort. If this doesn't work then you are paddleless up the proverbial creek.

`f.focus` This sets the keyboard focus to the relevant window independent of the position of the pointer. If repeated, it behaves as `f.unfocus`.

`f.fullzoom` This resizes the selected window to occupy the full display. If the window has already been zoomed then it is restored to its pre-zoom size. See also `f.zoom` for what you probably want.

`f.iconify` This iconizes a window or deiconizes an icon.

`f.lower` This drops a window to the bottom of the window stack. See also `f.raise` and `f.raiselower`.

`f.menu` *name* This command invokes the named menu (defined elsewhere in the `.twmrc` file by the `menu` command. Menus may have submenus by calling `f.menu` from a menu.

`f.move` This drags an outline of the window until the button is released. To cancel a move, press one of the other buttons before releasing the button that started the move.

`f.nop` This does nothing.

`f.quit` This causes `twm` to exit. If `twm` is the last line in the `X` script (see section 7.3) then the `X` server resets as well.

`f.raise` This raises a window to the top of the window stack. See also `f.lower` and `f.raiselower`.

`f.raiselower` If the window is at the top of the window stack it will drop to the bottom. If it is not at the top is will pop to the top.

`f.refresh` This causes all windows to be redrawn by their clients. See also `f.winrefresh`.

`f.resize` This invokes the resizing feature. A skeleton window is drawn, crossing the edge of the outline changes its shape as described above. To cancel a resize, press one of the other buttons before releasing the button that started the resize.

`f.title` This simply provbides titles for menus.

`f.unfocus` This restores the focus policy to being pointer-driven after the policy was changed by `f.focus`.

`f.winrefresh` This causes a window to be redrawn by its client.

`f.zoom` This causes the window to grow vertically to fill the entire height of the display. The window's width is not altered.