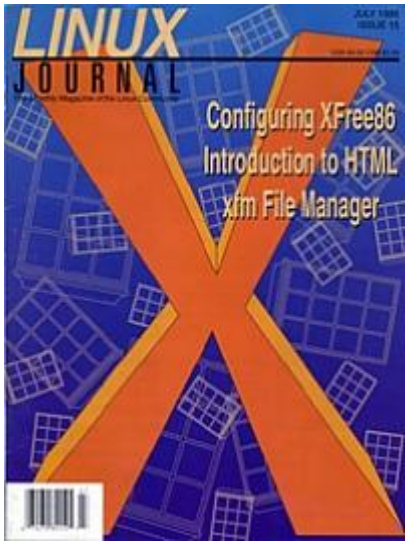
Linux Journal Issue #15/July 1995



Features

News & Articles

Reviews

Columns

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

The LINCKS GPD

Martin Sjolin

Issue #15, July 1995

“Computer Supported Cooperative Work” (CSCW) and “groupware” both refer to software that supports group projects. This article demonstrates how to build a small CSCW application using the free LINCKS CSCW.

In *Linux Journal* Issue 11, we described a typical installation of LINCKS, described the functionality of its components, and briefly introduced the main application interface, xlincks. Since LINCKS is distributed with an online tutorial, and a 70-page xlincks manual, we will assume that you know the basics about operating xlincks. In this second and last part, we describe how to create your own views, or general presentation descriptors (GPD), as we call them. For a reference manual description of GPDs, see chapter 7 of the *xlincks User's Manual*, and for a technical overview description see *Journal of Systems and Software*.

As a running example, we will show you how to use xlincks to create an address book, and the GPDs used to enter and display data. We start out by designing our data structures, then describe the LINCKS object model, and finally show how the address book is represented in the object model. We briefly describe the automatic search for GPDs used in xlincks and show our first GPD, followed by a description of the GPD's parts. We conclude by extending the newly-created GPDs to demonstrate a few additional features.

An Address Book

In our address book, the two obvious objects are the *person object* containing personal information, such as: name, birth date, e-mail address, and private phone number, and the *address object*. An address object can be shared by different persons and contains the common street address and common phone number.

The LINCKS Object Model

In LINCKS, each object consists of the following parts:

- **IMAGE** storing any type of information which can be PostScript, object code, a GIF, or a name (string). The size must be less than 232-1 bytes and the content is single-valued. (Currently, xlincks can only handle IMAGEs containing zero-terminated strings.)
- **ATTRIBUTE** describing or typing the object. The value of an attribute is a zero-terminated string shorter than 216-1 bytes. An attribute is named by two strings, a *group* tag and a *field* tag, where several attributes can share the same group tag, but the combination of group and field must be unique.
- **LINKS** containing links to other objects. A link field is *plural*, that is, for each object it can contain several links to other objects, as opposed to the attributes, which are single-valued.

In Figure 1, the window with three entries is the person object where the image contains the person's name, the second line specifies the birth date, and the fourth line is a link to an address object. The second, bigger window is the *node* view applied to the same object. (To apply another view, click on the item, then on the "Expand..." menu item, and then choose the node view. See the previous article in Issue 11, or read the xlincks manual, for more information.) The first entry in the *node* view is the image, followed by all the attributes and finally the link section beginning with the **SYSTEM:Parent** entry.

Notice that the system creates the two attributes **SYSTEM:Created** and **SYSTEM:Owner**--this pair is a good example of two attributes sharing the same group tag. In general we use all capital letters for the group part and lower case letters, possibly initial capital letters, for the field part.

Person GPD

GPDs are named by a string, which may contain white space. xlincks looks for a specific GPD first in your own **GPDmap**s and then in the list of system **GPDmap**s. This allows you to override the system GPDs with your own specialized GPDs. Now, if you expand the **Empty User 4's GPDmap** line, you will get a window with your first GPDmap as in Figure 2.

Replace the <<**table**>> with the name of the new GPD, **person** and then move out of the line with ctrl-n. A new placeholder <<**gpd**>> will appear which we fill in with a one line description of the new GPD, **view of person object**, as in Figure 3. Now, expand (meta-l meta-e) the description line to get the empty GPD template seen in the bottom of Figure 3.

First, you add a link from the GPDmap by first clicking on the line **Empty User 4's GPDmap** followed by a click on the line <<**gpdmap**>> in the empty GPD, then on the **Add Link** button—resulting in <<**gdpmap**>> being replaced with a link to **Empty User 4's GPDmap**.

Second, we need to define the logical structure by describing the logical parts in the person object. Earlier, we defined the person object to contain a **name** and **birth date**. Now, move to <<**direct**>> line in the **STRUCTURE** section and replace it with the name of our GPD, **person**. The name of the GPD used in GPDmap *must* occur at least once in the STRUCTURE part. After moving out of the line, we replace the newly created **empty: value** with the parts in our person object: **name** and "**birth date**"--since the part name includes a space we must use quotation marks (otherwise it would be three parts!).

Third, now that we've defined the structure of the person object, we need to define *where* to store the information. The **ACCESS** section of the GPD defines where we store or retrieve the parts of the logical structure. We would like to store the person's name in the **IMAGE** part and his/her birth date in an **ATTRIBUTE** called **ADDRESSBOOK:Date**.

To specify the name component, we move to the <<**direct**>> in the **ACCESS** section and replace it with the **name** (the name of structure part). Then we move out of the line and replace the **empty: value** with **IMAGE**. The resulting GPD can be seen in Figure 4. A warning: any omitted ACCESS specification defaults to **IMAGE**!

To define where we find the "**birth date**" in the **ACCESS** part we move the cursor to the **name** line in the **ACCESS** section and do an insert closest plural (*meta-l meta-i*), with the result as shown in Figure 5. Replacing the **direct** line with **birth date** (without quotation marks!) and the **empty: value** line with **ATTR ADDRESSBOOK Date** gives us Figure 6. Notice that we use the keyword **ATTR** followed by the group tag (ADDRESSBOOK) and the field tag to define the storage place for **birth date** in an attribute.

Last, if we would like to try our new GPD, we need to create a place to store the object, say in our LINCKS home directory:

- 1) Click on the line "Linux on the Road"
- 2) Do an insert closest plural (meta-l meta-i)
- 3) Expand the item (using meta-l meta-e or ctrl-left-click)
- 4) Fill in the GPD as person
- 5) Expand the same line (person) which should result in a two-line window saying item and empty: birth date as seen in Figure 7.

Address GPD

We will not cover the creation of the address GPD in that much detail, but the basic steps are:

- 1) Select the GPDmap (use the same that we used for the person GPD)
- 2) Move to the person line. and do an insert plural.
- 3) Name the GPD address and give it the descriptive line view of address object.
- 4) Expand on the view of address object.
- 5) And complete it according to Figure 8.

Now, we need to create an address object and we follow the same steps as for the initial person object. We add the object to our home directory with the result as in Figure 9.

Connecting the Address GPD & the Person GPD

We have now defined templates for storing and retrieving address and person objects, but we need to connect one person to one or more specific addresses. We would like to link one person object to one or more address objects under the link name of **ADDRESSBOOK Address** (group and field tag). We will modify the person object to include a link to an address object. First, we need to add a new entity to the logical structure. Let us call it **address**. Second, we need to specify where to find the new part, extending the person GPD as in Figure 10.

Notice that we have introduced the **ATTR CONSTANT** which defines a constant attribute, in this case a sequence of dashes. Moreover, in the GPD's **EXPAND** section we have defined that any expansion of the logical part **address** (which is found by following the link **ADDRESSBOOK:Address**) should be viewed using the address GPD. After storing the changed person GPD, we expand again the **Martin Sjölin** entry in our home directory and we see the added dashes as well as the placeholder <<**address**>> (Figure 11).

Adding a link to address **+46 13 148155** to the placeholder and after expanding that phone number, we get Figure 12.

To make the person GPD look more like the standard folder or our home directory, we can put a border around the address entry (for example, the phone number) and push the left margin 10 pixels to the right by adding an entry to the **FORMAT** section under the logical part address as:

```
address
   borderWidth=1;leftMargin=10;width=400
```

where we use **width** to specify the widget's width in pixels.

Combining the person GPD and address GPD

The last example shows how to create a combined person and address GPD, let us call it the **person and address** GPD. As before, select the GPDmap, add a new GPD, name it **person and address** with a one line description, as seen in Figure 13. This GPD contains several noteworthy features. First, we use the font specification in the **FORMAT** section. Any valid X11 font in our system can be specified after the equal sign. Sometimes quotation marks (") are needed around the font name. Also, the font and marginal specification is valid for *all* entries which follow below (in logical structure), as seen in Figure 13. We have applied the **person and address** view on the person object **Martin Sjölin** in the home directory.

Second, in the **ACCESS** section, **--- Indirect References ---** we have re-used the ACCESS specification in the **person** GPD by using the same logical name on the same logical part in the **address** GPD and then adding a link to the **person** GPD. The system follows the link to the other GPD and looks for the ACCESS specification under the name name and **birth date**. Thus, we only have the ACCESS specification in one GPD instead of copied into several GPDs (we avoid magic numbers!).

Third, what about the **addresslink** structure part? We have defined addresslink to point to the **address object** in the ACCESS section which results in an address object. Then, in the STRUCTURE section, we have used the indirect feature and re-used the **address** GPD (as seen by the descriptive name on the line below **address**).

Now, if we use the **--- Indirect References ---** in ACCESS, EXPAND, or FORMAT sections, we are *only* re-using the declaration for that specific logical part, that is, we are using that ACCESS, EXPAND or FORMAT declaration in the other GPD. But, when using the indirection in the STRUCTURE part, we use the STRUCTURE, ACCESS, EXPAND and FORMAT declarations in the GPD that is pointed to (in this case the **address** GPD) and no longer use any declarations in the GPD that is pointed from (the **person and address**).

For example, if we add a FORMAT specification for the **address** part in the **person and address** GPD, it will not be used since any FORMAT declaration must be included in the (**address**) GPD pointed to—try it out yourself. The mechanism is similar to a function call.

This introduction, along with the material in the xlincks manual, should help get you started. If not, you will have to bug the author enough to finish the real

GPD tutorial, hopefully before the next public release of LINCKS (hopefully released by the time you read this).

LINCKS RESOURCES

> Martin Sjölin is about to complete an MSc in computer science at the Department of Computer and Information Science, University of Linköping, Sweden. He is working in the fields of hypertext/hypermedia, document handling, CSCW, and information filtering/ sharing. He is responsible for support and development of LINCKS, whenever he is not browsing the net (WWW, mailing lists, Usenet). Beside computers, he enjoys cooking, backpacking, skiing, wind surfing, canoeing, and reading, whenever he is not hacking on LINCKS or Linux for the MacIntosh. (marsj@ida.liu.se)

Bibliography

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

xfm 1.3

Robert Dalrymple

Issue #15, July 1995

xfm converted this file manager skeptic. After reading this review, you too may be swayed.

Using a file manager is a matter of personal taste. Some of the hard core Unix users prefer to type their Unix commands (probably not even using a GUI), while others click away at a file manager, which takes over some of the mundane file operations. As one of those people in the first category, I had avoided the use of a file manager until I saw the utility and the flexibility of xfm, which is one of the file managers supplied with the Slackware distribution of Linux (but you probably haven't configured it yet).

Not only does xfm provide the usual services of a file manager—allowing the insertion, deletion, copying and movement of files and directories, but, in addition, its Applications window allows you to start programs by clicking their icons à la Windows.

Figure 1 shows a sample File Manager window, with some of its many pixmap icons (that make it look a lot better than Xfilemanager, the other manager that comes with Slackware). The two yellow folder icons denote directories. The directory folder on the upper left with the arrow heads can be double clicked to go up a directory. The other directory folder (nextdir) shows there is one directory below the present one. Double clicking on it opens that directory. The other icons are for different types of files (there are about 35 different file icons to distinguish between files ending in such things as ps, gz, tar, gif, etc). Double clicking a regular file—the one in white—brings it up in your editor, while other actions are associated with the other file types: double-clicking a PostScript (PS) file brings it up in ghostview, and clicking a gif file displays it. Pushing the right mouse button on an file icon brings up a menu of edit, move, delete, information, and permissions, each or which then causes a dialog box to open for (or with) more information. These menu options are all self-explanatory.

The three the buttons, **File**, **Folder**, and **View**, at the top of the File Manager window have drag down menus. For the File button there are numerous options, including select all and delete, a combination that would empty a directory. Under Folder are the options make a new directory or go directly to a new directory, so that you don't have to click through the entire directory tree to go from top to bottom, chasing folder icons. Home is a convenient option as well. View changes the file display in the File Manager—files can be listed by name (much like an ls -l listing) rather than with icons. A portion of the directory tree can be shown instead or you may elect to show the hidden files.

To move files between directories, it is convenient to open several File Manager windows—one for each directory you are using—by clicking the right button on a folder—this time we get a new menu allowing open, move, delete, information, and permissions, so we chose open. Or we can drag a directory folder out of the File Manager onto the root window and release. The moving process is simply done by selecting a file with the left mouse button and then dragging the file from one directory window to the other. To copy files, you do the same using the middle mouse button. (Note: files must be copied, not moved, between different file systems, otherwise you get a cross-device error.)

The Applications Manager that comes along with the File Manager window can be configured with the applications that you use frequently—and it's not just one windowfull, you change to different configurations by using the toolkit approach. More about this later.

Some of the icons in my Applications window (see Figure 2) do the following: Double clicking the Xterm icon brings up a color_xterm with a scroll bar and tcsh, while dragging an executable program from the File Manager window and dropping it on the Xterm icon executes that program. Every icon can have two actions—one for double clicking, one for drag and drop. Dropping files onto the trash can moves them to a directory called .trash for later disposal. Dropping PostScript files on the PSPrinter icon prints them out.

Setup

Type **xfm.install** (a program included with xfm) will add the directories .xfm and .trash in your root directory and place several configuration files in .xfm. You may then modify these files to optimize the program for yourself. These configuration files include xfmrc, a file that associates file types to icons and actions, and xfm-apps that lists the applications and icons that appear in the Application Manager and the associated double click and drag and drop actions. xfmdev is a listing of devices (disk drives/CD) and the mount and umount commands (see “Linux Tips”, *Linux Journal*, April 1995 page 41). Finally there is a file for each of the additional “tool boxes” that you wish to add.

The man page that comes with xfm is very well written and explicit on its use. Be sure to read it before doing any changes.

Configuring the File Manager

Many of the double-click/drag—drop operations are pre-configured in xfmrc based on the following format:

```
file_name:icon:push-action:drop-action
```

Xfm recognizes file names three ways: by the file name itself (e.g., core or Makefile), by its extension (***.c** means c source files), or by its prefix (**README***). The most common pattern used for the **file_name** is ***.suffix**, which, to have it recognize C files, is *.c, with the * permitting any file name ending in .c to be recognized. The **push-action** (activated by double clicking the icon) is what you most likely will change in the xfmrc file. For example, I changed the **push-action** on ***.gif** files to xv rather than xpaint. I also added:

```
*.au:xfm_au.xpm:cat $* >/dev/audio:
*.dat:xfm_data.xpm::
```

The first plays au files (audio files) when they are double clicked (read the Sound HOWTO, if you haven't configured sound), while the second merely associates the xfm_data icon to data files that I have created for executable programs.

Drop actions are those that occur when a file is dragged onto the icon. I don't have any drop actions defined for my File Manager window.

Configuring the Applications Manager

The programs and icons that constitute the Applications manager are dictated by the xfm-apps file. The entries in that file have a somewhat different form:

```
name:directory:filename:icon:push-action:drop-action
```

name is the title that appears in the Applications window, **icon** is the name of the icon to use, **push-action** is what to do when the icon is double-clicked, and **drop** is what happens when a file is dropped on the icon. A simple example is

```
PSPrinter:::printer.xpm::exec lpr $*
```

Double clicking this printer icon does nothing, while dropping a PostScript file on it from the file manager window results in the file being printed. The remaining entries in this file look like:

```
Xterm:::xterm.xpm:exec color_xterm -sl 600 -sb -fn
*     7x14 -j -ls -e tcsh:$*
```

```
Xclipboard:::clipboard.xbm:exec xclipboard:
Scilab:::math4.xpm:exec scilab:
Graphics:.xfm/xfm-graphics:xfm_appmgr.xpm:LOAD:
System:.xfm/xfm-system:xfm_apps.xpm:LOAD:
Calendar:::calendar.xpm:exec xcalendar:
CD:/cdrom::cdrom.xpm:OPEN: (*This line should not be broken)
```

Toolbox, Graphics and System are icons that change the Applications window to another containing a different set of application icons. For example, I made a simple graphics tool box with the following entry in xfm-apps:

```
Graphics:.xfm/xfm-graphics:xfm_apps.xpm:LOAD:
```

The LOAD command loads the .xfm/xfm-graphics file below:

```
Apps:.xfm/xfm-apps:xfm_apps.xpm:LOAD:
XFig:::draw.xpm:exec xfig: exec xfig -P -e ps -startf 16 $*
XPaint:::xpaint.xpm:exec xpaint:exec xpaint $*
XV:::xv.xpm:exec xv:exec xv $*
mpeg:::movie.xpm::exec mpeg_play -loop $*
```

The Apps line gets me back to the previous window: xfm-apps.

In the other applications file, System, I have in .xfm/system

```
Apps:.xfm/xfm-apps:xfm_apps.xpm:LOAD:
Xsysinfo::::xsysinfo:
TOP::::exec xterm -e top:
who:::view.xpm:exec who|xless:
lpq::::exec lpq |xless:
```

By piping the last two commands to xless, an xwindow is created that displays the results of the Unix commands.

Dialog boxes allow the input of command line parameters, such as:

```
LaTeX::::(latex %Latex_file\:%;beep):(latex $*;beep)
grep:::grep.xpm::grep '%Regular expression\:%' $*
```

The percent signs delimit the comment for the dialog box (with the **** escaping the **:**). The beep is to tell me when the operation is done; it is defined as **echo -n '^G'**.

Changing the Defaults

If you prefer a different default editor than emacs, then you need to add an entry to the .Xdefaults file in your root directory. The xfm man page lists resources that you can change. I changed two by adding **Xfm*defaultEditor: textedit** and **Xfm*updateInterval: 3000**. (Be sure to run xrdb .Xdefaults to get the changes implemented if you want them before you next start X.) The update interval change was to refresh the directories more often than the default 10000 milliseconds. You can also change the default directory paths to the pixmap files; this may be important if your pixmaps are not in the default

location (add **Xfm*pixmapPath: *your_path*** Alternatively, to make changes for all users, root can change options in the Xfm file in /usr/X386/lib/X11/app-defaults.

About Xfm

Xfm was created by Simon Marlow, who maintained it up to version 1.2. Albert Graef produced the present versions, fixing some bugs and adding the pixmaps. (He also graciously reviewed this article, improving it). As you read this, version 1.3.2 should be available, with such features as the recognition of "magic" file types in addition to those now specified in xfmrc, and better management of applications groups, such as installing applications groups within the applications manager, cut/copy/paste between applications files, and a view option for the File Manager (in addition to edit).

Getting xfm-1.3

Robert Dalrymple (rad@coastal.udel.edu) works at the University of Delaware. He uses Linux both at home and work. www.coastal.udel.edu)

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

HTML: A Gentle Introduction

Eric Kasten

Issue #15, July 1995

In the May issue of *Linux Journal*, Eric explained how to set up and install CERN's World Wide Web server software. This month, he tells us how to use HTML to create hypertext documents for viewing by Web surfers.

HyperText Markup Language (HTML) is a simple language for representing document format styles and links to other documents or media types, such as images or sound recordings. HTML can be used to create documents which contain styles such as underlined or bold-faced text. It can also be used to mix text, images, and sounds into a single document, the individual elements of which may be located on geographically distant systems around the world.

HTML is designed to create documents for the World Wide Web and it helps determine what is displayed when you are browsing documents with your favorite WWW browser. You can use it to create your home, or welcome page or to create a research document, article, or book. This document can then be viewed locally or made accessible for viewing by other Web surfers through the use of a WWW server, such as NCSA's or CERN's httpd daemon. This article introduces you to HTML basics so you may get started creating your own HTML documents.

Document Tags

You can use your favorite text editor to create an HTML document. The document will be composed of text, which will be displayed directly to the user, and markup tags, which are used to modify the appearance of the text or to incorporate images or sounds as part of the document. Tags are also used when referencing other documents or different locations within a document. Document references are called *hypertext links* or simply “links”.

A tag for indicating the start of a particular format is represented as a tag name enclosed in a pair of angle brackets. To indicate the termination of a format, the

tag name is prefixed with a **/**. For instance, **<I>Italics</I>** would display the word "Italics" in italic format. Let's examine a simple HTML document.

```
<HEAD>
<TITLE>Sample Document</TITLE>
</HEAD>
<BODY>
<H1>A Sample HTML Document</H1>
Here is some <B>Bold text</B>,
and here is some <I>Italic text</I>.
</BODY>
```

This makes use of a few basic tags. The text between the **<HEAD>** and **</HEAD>** tags is the document header. The header contains a **<TITLE>** tag which indicates the start of the document title tag and is terminated by a **</TITLE>**. The title usually isn't displayed as part of the document text by most browsers, but instead is displayed in a special location. For instance, Mosaic displays the title in the title box at the top of the browser window.

After the header is the body of the document, which is contained between the **<BODY>** and **</BODY>** tags. Inside the body the **<H1>** represents the start of a first level document heading. There are six levels of headings. Each increase in level results in a decrease in the prominence with which a heading is displayed. For instance, you might want to use an **H1** heading for displaying the document title in the document text, and then use **H2** for subheadings.

This is probably a good time to mention that tags are case-insensitive. Thus **<TITLE>** and **<title>** are the same tag; however, I will continue to capitalize document tags for clarity.

Physical Format Style Tags

Physical format styles are used to indicate the specific physical appearance with which to display text. The following is a list of physical format tags:

<B>*text***</B>**
: Displays *text* in bold face.

<I>*text***</I>**
: Displays *text* in italics.

<U>*text***</U>**
: Displays *text* underlined.

<TT>*text***</TT>**
: Displays *text* using a typewriter font.

The problem with physical formats is that there is no guarantee that a particular browser will display the text as expected. A user may modify the fonts that a browser uses, or the browser may not even have the specified font style available. For instance, if a text mode browser is used to display a document, it is unlikely that italic text can be displayed at all. To avoid the ambiguity associated with the display of physical formats, you may use logical format tags. In fact, it is usually recommended that you use logical format tags, in preference to physical format tags, wherever you can.

Logical Format Style Tags

I've already introduced you to one logical format. The H1 through H6 heading styles are logical formats for headings. If physical heading styles existed in HTML, you would have to specify a particular font and font size for each heading (for example, 30-point Courier). Following is a list of some of the more common logical format styles:

<CITE>*text***</CITE>**
: Used when *text* is the title of a creative work such as a book.

<CODE>*text***</CODE>**
: Used when *text* is a piece of computer code.

<EM>*text***</EM>**
: Display *text* with emphasis.

<SAMP>*text***</SAMP>**
: Used when *text* is a piece of sample output.

<STRONG>*text***</STRONG>**
: Display *text* with strong emphasis.

Of particular note, it is better stylistic practice to use the **EM** logical style in place of the *italics* physical style, and to use the **STRONG** logical style in place of the **bold** physical style.

Page Breaks, Rules, and Preformated Text

Carriage returns and white space are usually ignored in HTML. This is done because different browsers have different display capabilities. Where one browser might be able to display 100 character lines, another might not. This presents a problem when it is necessary to have a forced break, such as between paragraphs. Special tags were created so that an author could force a

break in a document. Following is a list of tags which can be used to force a break in a document:

<P>
Force a paragraph break.

**
**
Force a line break.

<HR>
Force a line break with a horizontal rule.

As a general note, it is best to use break commands, as well as other HTML directives in ways that keep your document device independent. The browser your readers use might format documents with different line lengths. If you use a **BR** directive at each line break found in a paper document you were converting into HTML, a browser might insert additional line breaks as well. This could result in a choppy-looking document, probably with alternating long and short lines.

Notice that these break tags don't have an associated closing tag. For instance the **
** doesn't have an associated **</BR>**.

If you do want to display something exactly as it is typed, HTML provides a tag for preserving the format of preformated text. The **<PRE>***text***</PRE>** directive instructs a browser to display *text* exactly as it is typed in the document. Once again, keep in mind that characteristics of some browsers may make it difficult for them to display the text in the exact format you expect.

Lists and Definitions

HTML provides for bulleted and numbered lists. Following is an example of HTML code which will generate a bulleted or unnumbered list of items:

```
<UL>
<LI>Apples
<LI>Oranges
<LI>Pears
</UL>
```

The **<UL>** tag indicates the start of an unnumbered list. Each item in the list is preceded by an **<LI>** tag to indicate the start of a new item in the list. Note that the **<LI>** tags don't have closing tags, and that when displayed, each list item will be separated by a carriage return.

To change this list to a numbered list, just change the **<UL>** and **</UL>** tags into **<OL>** and **</OL>** tags. A numbered list will display each item in the list preceded by a number instead of a bullet.

Another type of list is a definition or description list. Following is a snippet of HTML code demonstrating how to code a definition list:

```
<DL>
<DT>Apple
<DD>A red colored fruit
<DT>Orange
<DD>An orange colored fruit
</DL>
```

A **<DL>** tag begins a definition list, and a **</DL>** tag ends the list. The **<DT>** indicates a term to be defined, while a **<DD>** indicates a term definition. When the browser formats the list, each term and definition will be on a line by itself: the term is usually left justified with definition indented directly beneath it.

An Example Document

Let's look at an example document which contains many of the text markups which I just explained. The HTML source for the example document is listed below, and the formatted document, as displayed by Mosaic, as I have configured it, is shown in Figure 1, below.

Figure 1.

```
<TITLE>Example Document 1</TITLE>
</HEAD>
<BODY>
<H1>Example Document 1</H1>
<HR>
<H2>A Few Physical Styles</H2>
<I>This is in italics</I><BR>
<B>This in in Bold face</B><BR>
<U>This is underlined</U><BR>
<H2>A Couple Logical Styles</H2>
<EM>This is text is displayed with
        emphasis</EM><BR>
<STRONG>This text has strong emphasis</STRONG><P>
<H2>An Unnumbered List</H2>
<UL>
<LI>Apples can be red
<LI>Oranges can be orange
</UL><P>
<H2>A Definition List</H2>
<DL>
<DT>Term One
<DD>This is a short definition.
<DT>Term Two
<DD>This is a much longer definition, which
demonstrates what happens when a definition is
carried over to more than one line.
</DL>
</BODY>
```

Note a few interesting things about how the browser displayed this document. Text marked up to be underlined is not displayed as underlined. This

demonstrates one of the dangers of physical styles—some browsers may not support, or may not display a physical style as expected. Also, note that italic text looks like emphasized text and bold text looks like strong text. Notice the use of BR and P break tags, and the display of a multiline definition.

URLs

Uniform Resource Locators, or URLs, are designed to provide a standard format by which to point to a file. The file may exist on any network-accessible machine the browser has access to, and files may be accessed using a variety of protocols. The general form for a URL is: ***protocol://host.domain*[:*port*]*/path/filename***

The *protocol* indicates how the browser should communicate with the host it is requesting a file from. Probably the most common protocols are **http, file, gopher**, and **ftp**. The **http** protocol indicates that the browser should contact the server using the hypertext transport protocol, which is used by servers designed to serve HTML documents. The **file** protocol is used to retrieve a file from a local directory. Many browsers also support an **ftp** protocol for retrieving non-local files using anonymous ftp. The gopher protocol is used to retrieve documents from a gopher server.

The *host.domain* is the host and domain name of the remote server to contact in order to retrieve a document. If the document is on the local system, you can create a partial URL that does not specify the *host.domain*. To do this, you would omit the *//host.domain* from the URL. Following the *host.domain* is the optional (as indicated by the "[ ]" characters, which should not be entered) port to connect to in order to retrieve a document. This option is often omitted since most remote services will be provided at a well-known port on the remote system. For instance, the http protocol is commonly found on port 80, while gopher is found on port 70. When omitting the port, omit the ":" character as well.

The *path* is used to indicate the directory location of the desired document. The *filename* specification indicates the name of the file on the server where the document is stored.

As an example, if you wanted to view the document foo.html, which you know is located in directory /docs/ on the server remote.host.name, you could use the URL:

```
http://remote.host.name/docs/foo.html
```

Your browser would then display the foo.html document. It is worth noting that many browsers will use the file extension to help determine how to display a

document. For instance, .html is commonly used for html documents and .text is used for text documents. For this reason, it is usually a good idea to append a standard extension to your documents to help ensure that they will be displayed properly. You may want to refer to the documentation on your browser to determine what file extensions are supported, although many browsers are now referring to the mailcap file to help determine the interpretation of a particular extension.

References

Why do we care about URLs? URLs are used to make references to image, sound and other media files that you may want to include or have a hypertext link to. URLs are also used to create links to other documents. Let's look at an example of how to include an inline image in a document.

```
<IMG SRC="http://remote.host.name/gifs/foo.gif">
```

The **IMG** tag has an attribute, a specification which indicates a particular characteristic about a tag. In this case, the SRC attribute indicates the image the tag refers to. Assuming that your browser is capable of displaying inline GIFs, foo.gif would be displayed along with the text of the document containing the **IMG** reference. Remember to use proper file extensions for image references. A browser will use the extension to determine how to properly display an image. For instance, **.gif** is used for GIF images while **.xbm** is used for X bitmaps.

To create a hypertext link to another HTML document you might use the following HTML directive:

```
<A HREF="http://remote.host.name/docs/foo.html">Foodocument</A>
```

This is an example of an *anchor*. An anchor is specified using an **A** tag, which typically has an **HREF** attribute. Unlike the **IMG** tag, an anchor has to be terminated with a **</A>**. In this example, “Foo document” will be displayed in the documents as a hypertext link, or anchor, to foo.html. Many browsers will display this anchor using colored text or some other indication that the text represents a link. When the reader selects such an anchor, the browser will attempt to retrieve the document specified by the **HREF** attribute.

Hypertext links may also point to images which are viewed using an external viewer or to sounds which are played when the link is selected. An image can be used as a link instead of using text. The following code demonstrates how to do this:

```
<A HREF="http://remote.host.name/docs/foo.html"></A>
<IMG SRC="http://remote.host.name/gifs/foo.gif"></A>
```

In this example an **IMG** tag is used as the anchor for foo.html. This effectively makes the inline foo.gif an anchor a user selects. Anchors which are images present another problem. How is a text mode browser going to present an image, and if it can't present the image, how is a user going to select the link? Fortunately, HTML provides an additional attribute for the **IMG** tag. The **ALT** attribute can be used to specify a text string as an alternate to displaying the image. This can be done as follows:

```
<IMG SRC="http://remote.host.name/gifs/foo.gif" ALT="FOO">
```

The ALT attribute is also useful to provide a method a text browser can use to display an inline logo, or some other image that isn't necessarily a hypertext link.

A Second Example Document

Now let's look at a second example document which contains inline images and anchors. The HTML source for the second example document is presented below. Figure 2 shows how Mosaic might display this document.

```
<TITLE>Example Document 2</TITLE>
</HEAD>
<BODY>
<H1>The Second Example Document</H1>
<HR>
<IMG SRC="http://remote.host.name/gifs/foo.gif"> An inline image
<HR>
 This is an <A HREF="http://remote.host.name/docs/foo.html">
anchor</A> to the foo.html document.
<HR>
<A HREF="http://remote.host.name/docs/foo.html">
<IMG SRC="http://remote.host.name/gifs/foo.gif" ALT="FOO!!!"></A>
This image is also an anchor to the foo.html
document.
</BODY>
```

Notice how the anchor, *anchor*, is embedded in the text of the document, as are the inline images. Further, the document text is aligned at the bottom of the image. This can be changed by using the **ALIGN** attribute of the **IMG** tag. Here is an example which aligns the text with the center of the image:

```
<IMG ALIGN=middle SRC="http://remote.host.name/gifs/foo.gif">
```

top and **bottom** are also valid options for the **ALIGN** attribute.

Online HTML References

Further Things To Explore

This article is only designed to give you an introduction to HTML. HTML includes other logical styles which were not covered, and provides methods for displaying special characters. HTML can be used to construct a form which

users fill out and through which they can interact with a server. In short, there are many more avenues to explore than I have space to introduce you to in this article. Enjoy making your own WWW waves with HTML!

Eric Kasten has been a systems programmer since 1989. Presently he is pursuing his Masters in computer science at Michigan State University, where his research focuses on networking and distributed systems. Well-thought-out comments and questions may be directed to him at tigger@petroglyph.cl.msu.edu. You may also visit his home page at petroglyph.cl.msu.edu/~tigger.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Setting Up X11

Greg Lehey

Issue #15, July 1995

Would you like to take advantage of all the graphic features of the X Window System, but the complexities of hardware configuration leave you scratching your head? Greg explains how to get the best performance from your monitor without making it go up in smoke.

One of the best things about Linux is XFree86, the free windowing system built on top of X11R6. The current version, 3.1.1, runs on just about any hardware, so that there is very little reason why you shouldn't be running it.

Are you running XFree86? That's a different question. If you answer no, then this article may be for you.

Another great thing about Linux is that it runs on PCs, which have arguably the highest performance/price ratio of any computer family. (Yes, I know, people argue this point constantly.)

Unfortunately, there are hardly two PCs with the same hardware configuration; in the course of the PC's evolution, hundreds of different display boards have appeared, each with their own peculiar quirks. How come they can sell? Most people use DOS or a system derived from DOS, and the boards have the necessary driver software for DOS on the board. If you're running a real operating system, you don't want to use these drivers: they run in 16 bit mode, and they're slow. That's why XFree86 supplies its own drivers and incorporates them in the X server.

XFree86 drivers are a “good news, bad news” situation. First the bad news: setup can be more difficult. With Microsoft Windows, you install the board and the software that comes with it, and it works (well, you get a recognizable picture on the screen). With XFree86, things might not be as easy.

The good news is that the drivers are much faster and much more flexible. In particular, you can configure XFree86 to your exact combination of display board and monitor, if you know how, even though you need to tune it manually.

This may sound simple, or it may sound complex—and it is. Some people set up X in a few minutes, but others run into problems which make grown-ups cry. In this article, we'll look at:

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

I mean the last point seriously. Conventional wisdom says that you can't damage hardware with a programming mistake, but in this case, you can, and people do from time to time. When you've read the section on how monitors work, you'll understand, but please don't start tuning until you understand the dangers involved.

How TVs and Monitors Work

You don't have to be a computer expert to see the similarity between monitors and TVs; current monitor technology is derived from TV technology, and many display boards have modes which can use TVs instead of monitors. Those of us who were on the microcomputer scene 15 to 20 years ago will remember the joy of getting a computer display on a portable TV, a “glass tty” running at 300 or 1200 bps.

TVs and monitors display the picture by scanning lines across the screen. As in a book, the first line starts at the top left of the screen and goes to the top right. Each successive line starts slightly below the previous line. This continues until the screen is full. The lines don't have to be full; the picture is formed by altering the intensity of the electron beam as it scans the lines.

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

The electron beam can move at only a finite speed. When the electron beam reaches the right hand side of the screen, it needs to be deflected back again. This part of the scan is called the *horizontal retrace*, and it is not used for displaying picture data. The actual time that the hardware requires for the

retrace varies, but it is in the order of 5% to 10% of the total line scan time. Similarly, when the vertical deflection reaches the bottom of the screen, it performs a *vertical retrace*, which is also not used for display.

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

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

In a nutshell, that is how horizontal deflection works. Vertical deflection works in almost the same way, just slower, with one minor exception. This basic display mechanism was developed for TVs in the 1930s, when the term “high-tech” hadn't even been invented, and even today we're stuck with the low data rates that they decided upon in those days. Depending on the country, TVs display only 25 or 30 frames (pages of display) per second. This causes an unpleasant flicker in the display. This flicker is avoided with a trick called *interlacing*. Instead of displaying the frame in one vertical scan, the odd and even lines are displayed in two alternating half frames, which increases the apparent frame frequency to 50 or 60 Hz.

How Monitors Differ from TVs

So how do we apply this to computer displays? Let's look at the US standard NTSC system—the international PAL and SECAM systems are almost identical except for the number of lines and a minor difference in the vertical frequency. NTSC specifies 525 lines, but that includes the vertical blanking time, and in fact only about 480 lines are visible. The aspect ratio of a normal TV is 4:3, in other words the screen is one-third wider than it is high, so if we want square pixels, (one with the same height and width, which makes graphics software much simpler) we need one-third more pixels per line. This means that we can display

640 pixels per line on 480 lines. (Do these numbers look familiar? Now you know why.)

This resolution is normally abbreviated to “640*480”. PAL and SECAM have lower vertical frequencies, which allow a nominal 625 lines, of which about 580 are displayed. Either way, these values have two huge disadvantages: first, they are interlaced displays, and second, the resolution is the minimum acceptable for modern graphics displays. Older PC display hardware, such as the CGA and some EGA modes, was capable of generating these signal frequencies, but VGAs can no longer do it. This is a pity, in some ways; I'd like to have an X display on my TV in the lounge room, but my last EGA adaptor died a couple of years ago.

The first problem with these standards is interlace. It works reasonably for TVs, but it's a pain for computer displays—there's still more flicker than with a real 50 Hz or 60 Hz display. VGAs can still be run in interlace mode, but you shouldn't even think about doing so unless you're into masochism. The second problem is the resolution; nowadays, 1024*768 is a normal enough resolution, but I'm writing this on a display with 1280*1024, and many modern boards display 1600*1200. On the other hand, even 60 Hz refresh rate is barely adequate. Read any marketing literature and you'll discover that 72 Hz is the point at which flicker suddenly disappears. To get high-resolution, high-refresh-rate displays, you need some very high internal frequencies—we'll see how high further down.

How to Fry Your Monitor

Now we know that a monitor is just a glorified TV. TVs were designed to use the same circuitry for the horizontal deflection as well as generation of the high voltage required for the CRT. This simplified the hardware design at the expense of versatility loss. The flyback transformer which generates the high voltage is part of a resonant circuit which can only operate at one frequency. Run a flyback transformer off its intended frequency and it will run much less efficiently and use more power. This additional power will be dissipated in the flyback transformer and horizontal output transistor as heat. The result can be the failure of the flyback transformer, the transistor, or both.

You don't have to roll your own X configuration to burn out the monitor. Ten years ago, the standard display boards were CGAs (Color Graphics Adapter) and HDAs (Hercules Display Adapter) and they had different horizontal deflection frequencies and thus required different monitors. If you connected an HDA (18.43 kHz horizontal deflection frequency) to a CGA monitor (15.75 kHz, the NTSC horizontal deflection frequency), you could expect smoke signals within a few minutes.

In modern PC monitors the horizontal deflection and high voltage generation circuits have been separated. Called multi-sync monitors, they are capable of running at a range of deflection frequencies. Running at any of the frequencies in the specified range will work, but it is still possible to damage the monitor by running it out of the specified range. Note that just beacuse a monitor displays the image correctly doesn't mean that it is running in spec. I have a rather elderly Eizo 9500 (called Nanao in the US) which has three frequencies: (exactly) 31.5 kHz, 48 to 50 kHz, or 64 to 78 kHz. In fact, it will display at any frequency between 48 and 78 kHz, but if it were run at 57 kHz for any length of time, I would be in for a hefty repair bill. The moral of the story: **Don't ever run your monitor out of spec**. If your display is screwed up, there's a good chance that the frequencies are out. **Turn off the monitor**.

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

Another difference between TVs and monitors is the kind of signal they take. A real TV includes a receiver, of course, so you have an antenna connection, but modern TVs also have connections for inputs from VCRs, which are usually an audio signal and a video signal. The video signal consists of five important parts: the *red* signal, the *green* signal, the *blue signal*, and the horizontal and vertical sync pulses. This kind of signal is called *composite video*. By contrast, most modern monitors separate these signals onto separate signal lines, and older boards, such as the EGA, even use several lines per colour. Unfortunately, there is no complete agreement about how these signals should work; the polarity of the sync pulses varies from one board to the next, and some boards cheat and supply the sync pulses on the green signal line. This is mainly of historical interest, but occasionally you'll come across a real bargain 20" monitor which only has 3 signal connections, and you may not be able to get it to work—this could be one of the reasons.

The CRT Controller

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

The first part of the display controller creates the framework we're looking for: the horizontal and vertical sync pulses, blanking, and picture information, which

is represented as a series of points or *dots*. To count, we need a pulse source, which also determines the duration of individual dots, so it is normally called a *dot clock*. For reasons lost in history, CRT controllers start counting at the top left of the display, and not at the vertical sync pulse, which is the real beginning of the display. To define a line to the horizontal deflection, we need to set four CRTC registers to tell it:

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

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

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

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

on the display) is defined as the value of the VT register +256 if bit 0 of the overflow register is set, or +512 if bit 5 of the overflow register is set.

The XF86Config Mode Line

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

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

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

- A label for the resolution line. This must be enclosed in quotation marks, and is used to refer to the line from other parts of the XF86Config file. It is the only optional field—further down we'll look at what happens if it isn't present. Traditionally, the label represents the resolution of the display mode, but it doesn't have to. In this example, the resolution really is 640*480, but the a at the end of the label is a clue that it's an alternative value.
- The clock frequency. This may be a label, like the mode line label, but sometimes it really does have to match the clock frequency in MHz.
- The Horizontal Display End, which goes into the HDE register. This value and all that follow are specified in dots. The server mangles them as the display board requires and puts them in the corresponding CRTC register.
- The Start Horizontal Retrace (SHR) value.
- The End Horizontal Retrace (EHR) value.
- The Horizontal Total (HT) value.
- The Vertical Display End (VDE) value. This value and the three following are specified in lines.
- The Start Vertical Retrace (SVR) value.
- The End Vertical Retrace (EVR) value.
- The Vertical Total (VT) value.

As I noted, not every mode line needs a label. It's possible to have a number of mode lines for the same resolution, each with a different dot clock frequency. In this case, only the first line of a group needs the label.

This is pretty dry stuff. To make it easier to understand, let's look at how we would set a typical VGA display with 640*480 pixels. Sure, you can find values for this setup in any release of XFree86, but that doesn't mean that they're the

optimum for *your system*. We want a no-flicker display, which we'll take to mean a vertical frequency of at least 72 Hz, and of course we don't want interlace. Our multiscan monitor can any handle horizontal frequency between 15 and 40 kHz, so let's head for the highest and see what happens.

First, we need to create our lines. They contain 640 pixels, two porches, and a sync pulse. The only value we really know for sure is the number of pixels. How long should the porches and the sync pulses be? If you have a good monitor with good documentation, it should tell you, but most monitor manufacturers don't seem to believe in good documentation. When they do document the values, they vary significantly from monitor to monitor, and even from mode to mode. They're not as critical as they look. For example, here are some typical values from my NEC 5D handbook—Horizontal sync pulse: 1 to 4 µs, front porch 0.18 to 2.1 µs, back porch 1.25 to 3.56 µs.

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

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

The next question is: Do we have a dot clock of this frequency? Maybe. This should be in your display board documentation, but I'll take a bet that it's not. Never mind, the XFree86 server is clever enough to figure this out for itself—we'll see how the next article. At the moment, let's assume that you do have a dot clock of 33 MHz. You now need to calculate four register values to define the horizontal lines:

- The first value is the Horizontal Display End, the number of pixels on a line. That's easy enough—it's 640.

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

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

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

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

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

Now we can calculate our vertical frequency, which is the horizontal frequency divided by the vertical total, or 39.66/493 kHz, which is 80.4 Hz—not bad at all. By comparison, if you use the standard entry in *XF86config*, you will get a horizontal frequency of 31.5 kHz and a vertical frequency of only 60 Hz.

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

which work. This doesn't mean that they are optimal values—we'll look at that in the next article.

Greg Lehey is a partner in LEMIS-Lehey Micro-computer Systems. He resides in Germany.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Interview: Orest Zborowski

Phil Hughes

Issue #15, July 1995

On April 25, over a sushi lunch, Phil Hughes interviewed Orest Zborowski, the person who wrote X-Windows to Linux.

Phil: *I would like to get a little background on you.*

Orest: I grew up in Connecticut and then got a degree from Duke in North Carolina. I had a dual major in Electrical Engineering and Computer Science. Then I went to work at Kodak in Rochester, New York for eight years working on publishing and printing software. Now I work at Microsoft.

Phil: *Did you work on Unix systems at Kodak?*

Orest: Yes, we started working with old Sun systems, possibly Sun 1s and then Sun 2s, Sun 3s and Sun 4s. Thus most of my Unix experience was with the BSD flavor of Unix. We initially worked with SunView then migrated to working with X11R4 and Open Look.

Phil: *When did you get involved with Linux? And why?*

Orest: Version .12 in 1992, I think. It seems like so long ago. I had played with Minix at home but didn't see it as being more than something that would run the ls command. I saw no chance to run X on it. Linux, however, had real memory management and looked like a possible platform for X and I wanted to give a port of X to Linux a try.

Phil: *Did you consider one of the BSDs as an alternative to Linux?*

Orest: Yes, I did, but at the time BSD required a dedicated machine rather than a partition on an otherwise MS- DOS disk; it offered no math co-processor emulation and just seemed less open than Linux.

Phil: *So your first step was to get Linux up and running, and the next step for you was to port X to Linux?*

Orest: Right. I saw Linux as a great platform for running X at home and considered it a good challenge to get things to work. Also, the number of available apps for X made having a Linux port very enticing.

Phil: *What were the development steps?*

Orest: X used a lot of [Unix] System 5 stuff so I decided that the job was to make Linux fit X rather than X fit Linux. That is, I started writing the needed System 5 system calls for Linux. The end result is that the X server for System 5 and Linux are almost identical. A side benefit is Linux now includes System 5 as well as BSD functionality. This has made porting other applications to Linux fairly easy.

Phil: *What were the problems along the way? And when did things get exciting?*

Orest: X11R5 was stable but the C libraries for Linux were still evolving, not to mention the kernel itself. The daily task was more dealing with what worked and didn't work in the libraries, which were changing quickly, than doing the actual X development. This was my intent as part of the make-Linux-fit-X effort.

I remember that producing a running X server was a great stress test of both the libraries and the kernel, and we found a number of subtle bugs and compatibility problems in the process.

Because there was no graphical screen code at the time, I wrote a curses driver so I could test the X server code. Getting the server working meant that sockets had to work as did networking and virtual terminals. The big breakthrough was when the server came up with my curses driver. Because each pixel was represented as a dot in character mode, all I could see was an 80x24 patch of the “display” but there it was: the biggest “X” cursor in the world sitting over a huge xterm shell prompt. Huge, but it did work.

Once I was at this point, it was fairly easy to then add the other code to get the video to work in graphics mode. At this point, the pre-XFree86 team was gearing up (I call it “pre-XFree86” because it wasn't called that until much later). David Wexelblat and David Dawes, along with a small group, decided to actively support X11R5 on 386 platforms. This was a tremendous boon for Linux and I never dreamed that XFree86 would become what it is today.

Phil: *The X development team has a different organizational structure than Linux. Well, actually, X has a structure, but Linux decisions really just use the "send it to Linus approach". How do you feel about this?*

Orest: XFree development is more controlled than Linux. They use the BSD method rather than the Linux method. This has put some people off, but it really isn't a problem. In fact, it may be better for X development. Linux development has been very open and we all appreciate that. But Linux development lends itself more to that sort of environment than X, where a tighter decision-making mechanism is needed. I don't think there are any cases where reasonable suggestions have been ignored. People are always welcome to join the team, but the level of involvement may scare off those who only wish to dabble. The current group of active developers spend a lot of time with XFree86.

Today, Linux development needs to maintain backward compability. For that reason I ran the 1.0.9 kernel on my development system until 1.2 came out so I could make sure that new code would not break on the last production kernel.

It is important that Linux offer a stable, reliable path for upgrading as many of today's users are actually just that: users. They use Linux as the operating system upon which they run an application rather than develop software for Linux or modify Linux itself.

Phil: *Are you currently doing any X development?*

Orest: Yes. I am working on moving the X distribution to use ELF libraries. The next release of XFree86, 3.1.2, will probably be a maintenance release where the main distribution will still use a.out binaries, but ELF binaries will be included as well.

I am also doing little things here and there, but it's not the same level of involvement that there used to be. The package has gotten to such a point where it is instantly useful to a very large group of people. New development within XFree86 centers on new drivers and support for future XC projects.

Phil: *Let's predict the future. Where is Linux going, and then, where do you fit in to that future?*

Orest: To be commercially viable, Linux needs applications. The CD manufacturers, to their credit, are trying to make a "shrink-wrapped" version of Linux. What they have done so far is good but there needs to be more.

Phil: *And what is "more"?*

Orest: First, you need easy answers to questions like “can you run *blank* under Linux”. My sister, for example, runs some computer applications that aren't available for Linux. She would also need a nice graphical interface to system management; something as close to “load and go” as possible.

Phil: *Okay, what would your sister need available to make Linux a viable choice?*

Orest: Word processing, a stat package and a database.

Phil: *The big one missing here is a word processor. I don't think she wants vi and troff.*

Orest: Nope. And there are quite a few people without the technical expertise to handle such low-level software. They want to be productive—to solve their problems. Linux can provide a much better platform than anything else to develop these solutions, but the solutions themselves are not quite there.

Phil: *How about a package that had all these applications in them plus a phone book.*

Orest: And a scheduling package, and then you would have something that would meet the needs of an average PC user. The basic question to answer is, “Why should I run Linux rather than another operating system?” The apps we design need to be something that's difficult, if not impossible, to produce on any other platform available today. And note, the competition is not standing still, either. Comparing today's plans against today's commercial offerings means being instantly obsolete once those plans are realized and the competition has had time to catch up.

Phil: *What if we add an Internet-access package? That would use the real multi-tasking capabilities and included networking of Linux.*

Orest: Yes, I think we are on the right track.

Phil: *Okay, where do you fit into this future?*

Orest: I am working on some packages like these. I am really interested in working on this but I have a job which takes up quite a bit of my time. I know this is a common problem.

Because of the delays of the Internet and the part-time nature of people working over the Internet, I think this would need to be done commercially rather than as a cooperative effort over the net. Full-time, highly-focused

development has obvious advantages. That's where you can take the time to really put together something nice—nice enough for my sister to use.

Phil: *If I won the lottery and had money to burn, what do you think it would take to make this whole package a reality?*

Orest: A team of 10-20 people could make the package to end all packages we discussed above happen.

Phil: *I'll let you know if I win the lottery. In the meantime, thanks for taking the time for this interview.*

Orest: You're welcome. I enjoyed it.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

The Linux File System Standard

Garrett D'Amore

Issue #15, July 1995

The Linux File System Standard, abbreviated FSSTND, is important to more than gurus. In this artile, Garrett explains how it has worked behind the scenes to make life easier for all Linux users.

Many of us in the Linux community have come to take for granted the existence of great books on Linux like those being produced by the Linux Documentation Project. We are used to having various packages taken from different Linux FTP sites and distribution CD-ROMs integrate together smoothly. We have come to accept that we all know where critical files like **mount** can be found on *any* machine running Linux. We also take for granted CD-ROM-based distributions that can be run directly from the CD, with only a small amount of physical hard disk or a RAM disk for some variable files like /etc/passwd, etc.

This has not always been the case. Many of us remember Linux from the days of the SLS, TAMU, and MCC Interim distributions of Linux. In those days, each distributor had his own favorite scheme for locating files in the directory hierarchy. (Actually, some can go back further, back to the days when the boot and root disks were the only means of getting Linux installed on your hard drive, but I have not been a member of the Linux community quite that long.) Unfortunately, this caused many problems when dealing with different distributions.

The *Linux File System Structure* is a document created by a mailing list collaboration of contributors who wish to help end anarchy. Often the group, or the document itself, is referred to as the “FSSTND”. This is short for “file system standard”, and was the name of the original linux-activists mailing list channel. (The mailing list has since been moved to a different location.) Together, these people have put together a document which has helped to standardize the layout of file systems on Linux systems everywhere. Since the original release

of the standard, most distributors have adopted it in whole or in part, much to the benefit of their users.

Since the first draft of the standard, the FSSTND project has been coordinated by Daniel Quinlan, Daniel.Quinlan@linux.org, but the development of the standard is nearly as open as Linux itself. The number of people who contributed to the development of the document is quite large, since it was really developed by consensus. Some of the most significant contributors are listed in the FSSTND document itself.

There are a number of specific goals that the FSSTND group set out to accomplish. The first goal was to solve a number of problems that existed with the current Linux distributions at the time. Back then, it was not possible to have a sharable /usr partition, there was no clear distinction between /bin and /usr/bin, it was not possible to set up a diskless workstation, and there was just general confusion about what files went where. The second goal was to ensure the continuation of some reasonable compatibility with the de-facto standards already in use in Linux and other UNIX-like operating systems. Finally, the standard had to gain widespread approval by the developers, distributors, and users within the Linux community. Without such support, the standard would be pointless, becoming just another way of laying out the file system. Fortunately, the FSSTND has succeeded rather admirably in achieving its original goals.

There are also some goals that the Linux FSSTND project did **not** set out to achieve. The FSSTND does not try to emulate the scheme of any specific commercial UNIX operating system (e.g. SunOS, AIX, BSD, etc.) Furthermore, for many of the files covered by the FSSTND, the standard does not dictate whether the files should be present, merely **where** the files should be—if they are present. Finally, for most files, the FSSTND does not attempt to dictate the format of the contents of the files. (There are some specific exceptions when several different packages may need to know the file formats to work together properly—for example, lock files that contain the process ID of the process holding the lock.) The overall goal was to establish the location where common files could be found, if they exist on a machine.

The notion of having a standard that defines the location of certain files within a file system predates the FSSTND quite a bit. AT&T's SVID defined the location of some files, as well as a lot of other things that most of us didn't really understand. POSIX provided a clearer standard for a very limited number of files. The genuine FSSTND discussion began in early August 1993. Since then, despite some pessimism from some in the Linux community, the FSSTND has been enormously successful, releasing three public revisions of the document since September 1993. The latest, v1.2, was released on March 28, 1995.

Why do I care?

Now if you're like me, you ask “What does this do for me?” Well, the answer depends upon just who “me” is. The FSSTND has an impact upon Linux users, system administrators, distributors, documenters, and developers.

If you are a Linux user, and you don't administrate your own Linux system (granted, this situation is somewhat rare) then the Linux FSSTND ensures that you will be able to find programs where you'd expect them to be if you've already had experience on another Linux machine. It also ensures that any documentation you may have makes sense. (From my own experience, nothing is more annoying than having inaccurate documentation!) Furthermore, if you've already had some experience with Unix before, then the FSSTND shouldn't be too different from what you're used to using, with some exceptions.

Perhaps the most important thing is that the development of a standard brings Linux to a level of maturity authors and commercial applications developers feel they can support that. The recent explosion of Linux-related books is proof of this. Before the FSSTND, it would have been difficult to write books about Linux, since each system was different. The FSSTND is solving all that.

If you administer your own machine, you gain all the benefits of the FSSTND mentioned above. You may also feel more secure in the ability of others to provide support for you, should you have a problem. Furthermore, periodic upgrades to your system are easier. Since there is an agreed-upon standard for the locations of files, package maintainers can provide instructions for upgrading that will not leave extra, older files lying around your system collecting disk space.

The FSSTND also means that there is more support from those providing source code packages for you to compile and install yourself. The provider knows, for example, where the executable for **sed** is to be found on a Linux machine and can use that in his installation scripts or Makefiles. For instance, I've written a package that does just this to generate man pages with correct file references in them before it installs them. If you run a large network, the FSSTND may ease many of your NFS headaches, since it specifically addresses the problems which formerly made shared implementations of /usr impractical.

If you are a distributor, then you will be affected most by the Linux FSSTND. You may have to work a little extra to make sure that your distribution is FSSTND-compliant, but your users (and hence your business) will gain by it. If your system is compliant, third party add-on packages (and possibly your own) will integrate smoothly with your system.

Your users will, of course, gain all the benefits listed above, and many of your support headaches will be eased. You will benefit from all the discussion and thought that has been put into the FSSTND and avoid many of the pitfalls involved in designing a filesystem structure yourself. If you adhere to the FSSTND, you will also be able to take advantage of various features that the FSSTND was designed around. For example, FSSTND makes “live” CD-ROMs containing everything except some of the files in the / and /var directories possible.

If you write documentation for Linux, the FSSTND makes it much easier to do so, which makes sense to the Linux community. You no longer need to worry about the specific location of lock files on one distribution versus another, nor are you forced to write documentation that is only useful to the users of a specific distribution. The FSSTND is at least partly responsible for the recent explosion of Linux books being published.

If you are a developer, the existence of a FSSTND greatly eases many of your headaches. You can know where important system binaries are found, so you can use them from inside your programs or your shell scripts. Supporting users is also greatly eased, since you don't have to worry about things like the location of these binaries when resolving support issues. If you are the developer of a program that needs to integrate with the rest of the system, the FSSTND ensures that you can be certain of the steps to meet this end.

For example, applications such as **kermit**, which access the serial ports, need to know they can achieve exclusive access to the tty device. The FSSTND specifies a common method of doing this so that all compliant applications can work together. That way you can concentrate on making more great software for Linux instead of worrying about how to detect and deal with the differences in flavors of Linux.

As I've already indicated, the widespread acceptance of the FSSTND by the Linux community has been crucial to the success of both the standard and operating system. Fortunately, nearly every distribution of Linux has been constructed to conform to the the Linux FSSTND. The odds are, if you're running Linux, you are running a FSSTND implementation. If your implementation isn't at least partially FSSTND-compliant, then it is probably either *very old* or you built it yourself. The FSSTND itself contains a list of some of the distributions that aim to conform to the FSSTND. Be aware, however, that some of these distributions are known to cut some corners in their implementation of FSSTND.

What's next?

There are, of course, still unresolved issues which the FSSTND group is still going to have to face. One of the upcoming ones is going the be the organization of architecture-independent scripts and data files (/usr/share). Up until now, Linux has only been widely available on i386 and compatible machines, so the need for standardization of such files was non-existent. The rapid progress being made by the ports to other architectures (MC680x0, Alpha, MIPS, PowerPC) suggests that this issue will soon need to be dealt with. Another issue that is under some discussion is the creation of an /opt directory as in SVR4. The goal for such a directory would be to provide a location for large commercial or third party packages to install themselves without worrying about the requirements made by FSSTND for the other directory hierarchies.

By now your appetite should be whetted and you may want to read the actual FSSTND document yourself. Information about how to participate in the continuing development of the FSSTND is contained in the FSSTND itself. The participants would appreciate it if you carefully read the complete document *before* you join in. That way you'll be better prepared to contribute in a valuable way to the development of the standard.

The FSSTND provides the Linux community with an excellent reference document and has proven to be an important factor in the maturation of Linux. As Linux continues to evolve, so will the FSSTND. We are grateful to Linus Torvalds, H. J. Lu, the Free Software Foundation, and many others who have helped make Linux *available*. The FSSTND team has helped to *standardize* Linux, making it useful to a wider audience.

I would like to thank Daniel Quinlan for his many valuable corrections and suggestions for improvement to this article.

You can obtain the FSSTD document via FTP from tsx-11.mit.edu in /pub/linux/docs/linux-standards/fsstnd/. There is also an excellent FAQ (answers to Frequently Asked Questions) about the FSSTND, which is maintained by Ian McCloghrie, in the same directory.

Garrett D'Amore is a senior in computer science at San Diego State University, where he works with Linux and SunOS in the development of TCP/IP-based client-server applications using C++. He will be graduating in December, so he invites inquiries from interested employers. You can find him via e-mail at garrett@sdsu.edu or via the WWW at http://www.sdsu.edu/~garrett/

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

AX Graphical Display Server

Mark Ganter

Issue #15, July 1995

AX is a replacement X-Windows or X-11 server based on X11R5 and is available for most 386-, 486-, and Pentium-based UNIX systems (including our favorite—Linux).

The Accelerated-X (AX) Graphical Display Server version 1.1 is sold by X-Inside Incorporated. AX is a replacement X-Windows or X-11 server based on X11R5 and is available for most 386-, 486-, and Pentium-based UNIX systems (including our favorite—Linux).

The AX product is under continuous development, and as of this writing is available as AX version 1.2 Beta/4 for beta testers and AX version 1.1 for users who wish to purchase the current stable version (1.1). The product arrived in a plain white envelope which contained a single 1.44MB diskette, a single 3/8 inch-thick User Guide, and a two-page release note.

System requirements: any 386/486/Pentium system with 4MB of memory, Linux, at least 4MB of free disk space under /usr, a mouse, and a supported graphics board and monitor (more about this later).

System recommendations: any 386/486/Pentium system with at least 8MB of memory more than the OS requires, a 3-button mouse, swap space which is 2/3 times the amount of system memory, and a high-end graphics card and monitor.

As a general recommendation, X-inside suggests, as do I, that you get the XFree86 distribution and get it working in at least one graphics mode for your system. You will at least need the XFree86 distribution installed, as AX is only a server, not a complete X distribution.

Before you start, you will need the graphics board name and vendor, the amount of graphics memory, the graphics chip set, the display capabilities of

the board; the monitor name and vendor, the maximum resolution, and maximum refresh rate; the national language of the keyboard in use, the mouse name and vendor, whether the mouse is 2 or 3 button, and the mouse interface (and if it uses a serial interface, then you will need the name of the serial device the mouse is using).

With this information, the install action is simple and straightforward. As root, insert the diskette, **cd /usr/X386/lib/X11** , then **tar -xvzf /dev/fd0** (assuming your 1.44MB is /dev/fd0). Next, run the install script **AcceleratedX/bin/Xinstall** , which creates two directories, /usr/X11/bin/Xaccel and /usr/X11/bin/Xsetup, then renames the existing X server to /usr/X11/bin/X.LINUX and makes a link from /usr/X11/bin/X to the Xaccel server. Lastly, run Xsetup to choose your graphics board, chip set, resolution modes, mouse, etc., to create the configuration file /etc/Xaccel.ini. When you exit Xsetup, you are ready to run AX.

Don't make any other changes to your system or to X-Windows. Furthermore, AX is even set up to be uninstall friendly. Xinside provides a script AcceleratedX/bin/Xuninstall which does the bulk of the work. Several files are still left on your system which you must remove by hand—/etc/Xaccel.ini and any $HOME/.Xaccel.ini files in user directories.

I took the time to run some benchmarks with the xbench suite to determine if the manufacturer's claims were real (up to 450,000 xstones and getting faster everyday). The test system was a 486/DX33 with 16MB of memory and VLB. I tested a ISA ET-4000 (9427 xstones), ISA Orchid F1280 (59,189 xstones), and a VLB Genoa Phantom ET4000/W32 (112,442 xstones). The benchmark numbers that were generated matched very well with those available from Xinside. Therefore, I feel that Xinside's benchmarks are true and believable (for more information, see their WWW site, send them e-mail, or call them). Generally, I found that AX produced a speed increase between 10 and 500 percent in various X-Window operations with higher performance gains dependent on better (read “more expensive”) graphics boards in comparison to XFree86. The AX server has direct support for almost all of the high-end graphics boards (in the price range of $250-$2500).

The most amazing part of the AX product is the configuration (or the re-configuration) process. Xsetup is a menu-driven configuration program. Simply choose your graphics board/chip set, resolution, etc., and the Xsetup program does the rest. AX has graphics board support for 170 different boards and over 36 monitors. If, however, you decide you need written instructions, the user manual provides fourteen pages of configuration details and about forty pages of technical details. I am pleased to say I did not need to use the manual. Xsetup and AX did what I expected.

The environment for which I use Linux and X-Windows often requires changing graphics boards and graphics monitors. Before AX, I might spend one-half of a day just reconfiguring X-Windows (especially for each new graphics card and monitor combination). After AX, it took about 2-5 minutes to reconfigure (generally, it took more time to close the computer case than to reconfigure).

AX has support for a variety of hardware that I did not get a chance to test, but does warrant mention. Most notably, AX contains multi-headed X-server support which allows up to 8 graphics cards to be installed and in use and permits the cursor to be moved from one screen to another (generally, each screen is displayed on its own monitor). Each board may have different resolutions and depths. Generally, all displays share all pointing devices. AX also has built-in support for graphics digitizing tablets which allows you to use a tablet instead of a mouse. Most common tablets are supported.

While AX is a good product, it is not perfect (but who can honestly say their software is bug free?). One of the mice I used during testing was not correctly identified as being a three-button mouse (instead it functioned as a two-button variety). This **only** happened after a cold boot; otherwise it was identified correctly. Also, the release notes warn that the VGA console may be set to a non-standard character mode (i.e. other than 80x25 mode) during the Linux boot. If you are running in such a mode, upon returning from AX your console will have its contents skewed. Currently, there is no workaround from AX. Xinside suggests that you use the 80x25 standard mode and does not consider this a bug because they have valid technical reasons for this behavior which they will explain to technically curious users.

I have been running AX for several months (during this review process) and have not had a single problem with compatibility or with stability. The AX product just seems to work. If your system requirement includes high performance X-Window graphics, then you should be taking a serious look at AX. To receive more information, contact X-Inside Incorporated via e-mail at info@xinside.com, or by ftp at ftp.xinside.com, or at their website on www.xinside.com.

Mark A. Ganter is an Associate Professor in Mechanical Engineering at the University of Washington. Mark has enjoyed Linux since the early days—version 0.12. His professional interests lie in the area of computational geometry and computer graphics. You may reach Mark at ganter@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Metro-X

Bogdan Urma

Issue #15, July 1995

Why should I pay for a commercial X server when I can get a free one from the XFree86 distribution? Three reasons come to mind: compatibility with more obscure or higher-end graphics cards, speed, and ease of use.

Metro-X Enhanced Server Set, the commercial X server from Metro Link Incorporated, satisfies all these needs. An X11R5-based server, Metro-X replaces your current XFree86 server while fitting in nicely with the rest of the XFree86 distribution, whether it is XFree-2.x or 3.x. You can successfully run an X11R5-based server on a X11R6 system with no problems at all. I tested it with Linux 1.2.x and the XFree-3.1.1 distribution.

Metro-X has accelerated support for over one hundred different graphics cards, including a variety from Diamond and Matrox, and the very high-end Imagine-128 from Number Nine. There is also touch screen support for Carroll and Elographics serial controllers and multi-headed display support, which allows you to have more than one graphics card and monitor under X.

Metro-X for Linux comes on two diskettes, together with a bound manual and a quick installation guide. The instructions in the quick installation guide are brief and to the point. After making two symbolic links, extracting the product from the disks using cpio, and then untarring it, I was ready to start the configX program which Metro-X installs in your X11 bin directory. Metro-X also installs a touch-calibration program in your X11 bin directory for users who plan to use serial controllers for touch screen support. All the other Metro-X files are installed in a /usr/lib/X11/Metro directory, which makes it easy to keep track of where everything is.

If you have ever tried configuring XFree86, you'll be very pleased with how easy it is to use configX to configure Metro-X. configX presents you with a menu from which you can select the server, graphics card, video modes, and mouse you plan to use with X. When you select the server option from the menu, you

are presented with a choice of servers: one for S3 based cards, one Matrox cards, and so on.

After selecting the correct server for your video card, you get to select which extra X extensions you want your server to have. Your choices include XIE 5.0, PEX 5.1, and DEC Xtrap 3.3. I chose both XIE and PEX. configX proceeded to build the server using the shared libc and libm libraries. You can always go back and rebuild the server again should you want to add or remove extensions to or from your server.

After building the server, you select the correct video card and the video modes you wish to use. I selected my Diamond Stealth 64 from the graphics cards list and chose my video modes. configX allows you to select as many video modes as you want with different resolutions and refresh rates, as long as they all have the same number of colors. Next, you choose your mouse from the mouse menu and, finally, select the *Save and Exit* option, which saves the newly created Xconfig file for use with Metro-X. That is all you need to do to configure X, and I did it in under 5 minutes.

After building and configuring my server, I proceeded to fire up X using the startx command. Within seconds, X was up and running in 16.7 million colors at 1024x768 resolution, and it was fast. The installation and configuration, which took less than ten minutes combined, worked flawlessly, and I was very impressed. Switching resolution modes works just like it does with XFree86, using the **CTRL ALT +** and **CTRL ALT -** key combinations. After playing around for a bit, I decided to do some benchmarks with the xbench suite to see how my new server would compare to XFree86. On all the tests I did, at 256 - 16.7 million colors, and at different resolutions, Metro-X outperformed the XFree86-3.1.1 S3 server using my Diamond card by about 20k - 30k xstones. I was even more impressed.

If you have a graphics card which isn't supported by XFree86 (such as some Diamond cards, Matrox cards, and the Number Nine Imagine-128), if you want a simple menu-driven configuration system for X, or if you simply want more speed, I highly recommend purchasing the Metro-X Enhanced Server Set for your Linux machine. At $99, it is a real bargain. Metro-X comes with 90-day free tech support and a 30-day money back guarantee. Contact Metro Link Incorporated at (305) 938-0283, or e-mail at sales@metrolink.com, or check out their WWW page at www.metrolink.com.

Bogdan Urma is studying physics and computer science at Cornell University and hopes to get his B.S. by next year. He has been using Linux since 1993 and spending way too much time with it. He welcomes your comments sent by e-mail to burma@newton.ruph.cornell.edu, or by snail mail c/o *Linux Journal*.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

Motif(s) for Linux

Bogdan Urma

Issue #15, July 1995

A look at two popular Motif for Linux distributions, GUI Corporation's SWiM Motif 2.0 and Metro Link Inc.'s Motif 2.0.

One of the most frequently asked questions in the Usenet Linux newsgroups is “Motif for Linux?”. Why? To answer that, I should first explain a little about what Motif is and why it is in such demand in the Linux community.

Motif is a set of programming libraries and header files, copyrighted by the Open Software Foundation (OSF), which allow programmers to create X programs using the Motif “look and feel”. Similar to the Athena toolkit (libXaw), the Motif toolkit (libXm) is also based on the Xt toolkit (the X “Intrinsics”), but the Motif toolkit has a distinct 3-D look to it which makes it much more pleasant-looking. Motif also includes a window manager called **Mwm** which, as of version 2.0, allows virtual desktops, a feature that users of earlier versions of Motif have wanted for a long time. Many people think of Motif and Mwm as the same thing, but Mwm is really just a small part of Motif. Motif is accepted as the standard X toolkit by major Unix vendors like IBM and HP, so it seems natural that Linux users want to use Motif on Linux platforms as well.

Even if you don't ever intend to develop Motif applications, having Motif on your system will be useful. Having access to the Motif shared libraries, which are included with Linux Motif distributions, will allow you to compile many freely-available Motif applications with shared Motif library support, resulting in much smaller binaries which take up significantly less memory than statically-linked binaries which you can retrieve from the Internet or copy from CD-ROMs.

Which Motif for Linux?

This is another frequently asked question by Linux users. Two of the more popular Motif for Linux distributions, GUI Corporation's SWiM Motif 2.0 and Metro Link Inc.'s Motif 2.0, are reviewed here.

Metro Link

Motif 2.0 for Linux from Metro Link consists of five 3.5" disks, a quick installation guide, and the OSF Motif 2.0 User's Guide booklet. The user needs to have the XFree86 3.1.1 libraries already installed and a minimum of 8 MB RAM for things to work. Included on the disks are the Motif man pages, source to the OSF/Motif demos, precompiled binaries, Mwm, shared and static libraries, header files, the Uil compiler, Mrm, some extra utilities such as a pixmap editor, an xbm browser utility, and a huge set of colored pixmaps. Metro Link also includes the Xpm pixmap library, which OSF requires for building Motif 2.0 apps.

The installation is completely trivial as long as you follow the quick installation guide. Simply extract the files from the disks, untar them, and you're ready to play. The product takes up anywhere from 20 to 30 MB of disk space, depending on whether you wish to install the sources to the man pages. I chose not to, since the preformatted cat pages were also provided, and it saved some disk space.

Having everything installead, I proceeded to compile Mosaic, xmcd, and a bunch of other free Motif apps available on the net. Everything went smoothly, xmkmf working flawlessly with Metro Link's new Imake.rules, Motif.rules, and Motif.tmpl files. Not one problem. I also decided to build the included OSF/Motif demos, and that also worked out painlessly. I was a very happy camper. I actually expected something to go wrong, but that never happened. **Mwm**, the window manager, also ran smoothly, although on an 8 MB machine that might not be true. Compiling Motif apps on an 8 MB machine will also probably be very slow, but on my 16 MB machine (the recommended RAM amount) everything worked great. Metro Link offers free technical support for Motif 2.0 by phone, fax, and e-mail, but I can't tell you how good the support is because I never had to use it. I never ran into any problems with their product.

SWiM

Next up is GUI Corporation's SWiM Motif 2.0. The main distributors are ACC Corp. (U.S.) and Lasermoon Ltd (Europe). SWiM Motif 2.0 ships on CD. Also included are the OSF Motif 2.0 User's Guide booklet and an installation guide. Similar to Metro Link's distribution, SWiM also includes Mwm, shared and static libraries, header files, Uil compiler, Mrm, man pages, and source to the OSF/

Motif demos. In addition, the SWiM CD includes the source to all OSF Motif 2.0 documentation, including the Motif 2.0 Programmer's Manual, X11R5 and ELF shared libraries (as well as X11R6 ones), the complete XFree86 3.1.1 distribution, live file systems for XFree86 3.1.1 and SWiM Motif 2.0, and an archive of over 300 MB of Motif programs gathered from different places. The distribution takes about the same disk space as Metro Link's, and the memory requirements are the same.

After mounting the CD, I proceeded to install SWiM. The installation guide that I had was geared towards a floppy disk installation, based on SWiM's previous release which was not on CD. The README in the top CD directory did not really give any installation instructions, but after navigating around the subdirectories, I managed to find the directory with the main distribution. Along the way I came across directories which had X11R5 and ELF libs, so I assumed that the default installation was using X11R6 (XFree 3.1.1) libs. I was right. I then ran the Install script and everything went well until the script bombed because it was trying to untar something to the current directory, which was read-only since it was on the CD. As the last step of the installation, SWiM is supposed to untar a file (mwm_make.tar) and then compile mwm from the contents of that file using the shared Motif libraries that it had installed earlier. Since I was on a read-only file system it naturally failed. I looked at the Install script and proceeded to untar the mwm_make.tar file in /tmp and build mwm there. This worked out and then I was done with the installation.

I then started X and Mwm and proceeded to compile some programs. Everything worked perfectly and smoothly. SWiM comes with a mxmkmf program (and mx5mkmf for X11R5 users) which replaces xmkmf when compiling Motif programs using Imake files. For use with mxmkmf, SWiM also installs new Imake.tmpl, Motif.tmpl, and Motif.rules files in /usr/lib/X11/config, which are, by default, for use with X11R6-based distributions. **X11R5** replacements are also installed in a seperate subdirectory. Instead of running xmkmf, I ran mxmkmf to compile the demos. Things worked according to plan.

I did run into some problems later when trying to use the uil compiler to produce some uid files required by some of the demo programs. It seems that SWiM installed an X11R5-based uil binary during the installation process, even though the distribution was set up for an X11R6 based system. Since I couldn't find an X11R6-based uil binary, I had to fetch some X11R5 shared libraries in order to run the uil compiler—not too big a deal, but I would have preferred to have an X11R6 binary. Other than that, and some of the installation problems, SWiM worked fine. Free technical support is also available in Europe for SWiM from Lasermoon Ltd.

One major difference between SWiM and Metro Link Motif 2.0 is that while Metro Link's shared Xm library requires linking with the Xpm library, SWiM's doesn't, since the Xpm library is compiled into SWiM's Xm library. I mention this difference because shared binaries compiled with SWiM won't work on a Metro Link system and vice versa, due to the differences between the shared Xm libraries.

At the time of writing this article, the Linux community is moving towards ELF shared library support from the current a.out format. The SWiM CD includes ELF shared libraries, but the Metro Link distribution does not; since ELF libraries are still under development and changing very rapidly, Metro Link has decided to wait until ELF is stabilized and in wider use before releasing ELF libraries with their Motif 2.0 package. By the the time you read this article, this change will probably have already taken place.

[InfoMagic just released Moo-Tiff and it will be reviewed in a future issue of *Linux Journal*—ED]

Contact

- Metro Link Motif 2.0 Metro Link Incorporated at (305) 938-0283 or e-mail at sales@metrolink.com
- SWiM Motif 2.0 U.S. - ACC Corporation Inc. at (800) 546-7274 or e-mail at info@acc-corp.com
- Europe - Lasermoon Ltd at +44 (0) 329 826444 or e-mail at info@lasermoon.co.uk

Bogdan Urma is studying physics and computer science at Cornell University and hopes to get his B.S. by next year. He has been using Linux since 1993 and is spending way too much time with it. He welcomes your comments by e-mail sent to burma@newton.ruph.cornell.edu, or by snail mail c/o *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

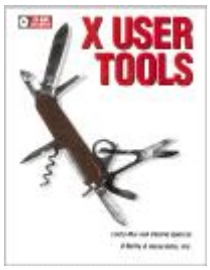
X User Tools

Danny Yee

Issue #15, July 1995

Browsing X User Tools is a more pleasant way of finding new programs than long ftp sessions, and it's a lot more fun to read than manual pages.

- **Authors:** Linda Mui & Valerie Quercia
- **Publisher:** O'Reilly & Associates
- **ISBN:** 0-56592-019-8. 812 pages, CD-ROM, index
- **Price:** $49.95
- **Reviewer:** Danny Yee



X User Tools is more than just a guide to various X programs. It begins with a fairly general introduction that assumes no prior knowledge of X and finishes with almost a hundred pages on X system administration, including an introduction to Tcl and Tk. The approach is unstructured - most chapters or chapter sections could stand by themselves and coherence is provided by plentiful cross-references rather than by linear progression - and chatty rather than formal. The programs covered include: desktop accessories (clocks, calendars, screen savers); network applications (mailers, xarchie, xftp, Web browsers); editors; games; xterm; window managers (twm, olwm, fvwm and mwm); resources and fonts; graphics tools; system administration utilities and lots more. (I'd call most of these applications rather than tools, but that's quibbling.) The included CD-ROM contains binaries (Alpha OSF/1, HP7000, HP/UX, Sun3, Sun4, Solaris, RS6000 and DECstation Ultrix) and sources for all the tools discussed which aren't in standard distributions (and some which are). Well over one hundred different programs are included.

Yes, these programs *can* all be ftp-ed, installed and run without this book, but browsing *X User Tools* is a more pleasant way of finding new programs than long ftp sessions, and it's a lot more fun to read than manual pages. While the serious X system administrator will want a book devoted solely to administration, and the complete novice to X with no Unix background may find *X User Tools* a bit overwhelming, almost anyone who uses X should find something of value in this volume. The people likely to appreciate it most are those running X under Linux or FreeBSD at home, who must do basic system administration jobs themselves and who may not have ftp access, making the CD-ROM invaluable.

All book reviews by **Danny Yee** are available via anonymous FTP ftp.anatomy.su.oz.au in /danny/book-reviews (index INDEX) or URL www.anatomy.su.oz.au/danny/book-reviews/index.htmlCopyright **Danny Yee** 1995. Comments and criticism welcome.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Letters to the Editor

Various

Issue #15, July 1995

Readers sound off.

What's Free?

For several years, my son has been telling me about free software, and I have failed to understand completely what he meant. I associate “free” with “without cost”, and since I spend very little on software anyway, I have felt no need to use “free software”. I did not until recently understand that the freedom to change the software, which access to the source code provides, is far more important than lack of cost.

This misunderstanding is widespread. Therefore, I would like to propose an alternate term for software with source code—“liberated” software. Liberated software liberates the programmer. —Daniel L. Johnson, M.D., F.A.C.P johnsond@uwstout.edu

Usefulness

Once again I'm at work in front of my HP workstation, about to request more information on a software product I noticed in an *LJ* ad, that just might solve a problem a co-worker asked me about last week. A few months ago my system administrator bought BRU (from an *LJ* ad I gave him) for an HP workstation headed to a tele-commuting co-worker's house. I occasionally daydream about having time to explore my Linux system at home, but I consistently read *LJ* because its columns, articles, and advertisements give me information I use at work. —Greg Deitrick deitrick@shell.com

Thanks!

I would like to say that I was absolutely impressed with the May issue of the *Linux Journal*, especially with the articles which dealt directly with system

administration such as how to set up a WWW site, the article of an ISP using Linux, the Majordomo setup/configuration article, etc.

Keep up the good work. Now I remember why I subscribed to this magazine in the first place! —John Coy jcoy@magic.yournet.com

PVM

Several times your articles have mentioned a guy from NASA who uses PVM instead of supercomputers, and that he gave a talk on it at a conference, etc.

Please have him, or someone in attendance, paraphrase what he is doing, and perhaps speculate that nnn Pentium 90's equals such and such Supercomputer, at so many gigaflops, etc.

Please also consider a `Porting Corner' article every month to summarize the progress of ports to other computers (i.e. Alpha, MIPS, PowerPC, etc). Only a quarter page or smaller would be required, with a line or two on each port. —Doug Fortune

LJ Responds:

In response to your first point, we do intend to have an article on this system. However, there is no way to say that nnn Pentium 90's is equivalent to any supercomputer; a loosely-coupled parallel system like that works well only on certain problem domains. We intend to have an article on the Beowulf system, but it is currently an ongoing research project, and is not ready for an article at this time.

The architects of Beowulf are about to build a second-generation system based on their current experience and research with their first system, and the results from that system will be more interesting and worthwhile to readers with a serious use for the technology. All the software that they are using will be released as a package, and when it is, we will certainly pursue an article.

In response to your second point, progress on the ports to various architectures does not progress in a way that facilitates monthly reporting. We will report on significant progress on the ports. In particular, our “Stop The Presses” article in this issue mentions that the Linux/Alpha port is now self-hosting. We will also report on the state of all ports from time to time, as we did recently.

Corrections

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Linux at DECUS

Michael K. Johnson

Issue #15, July 1995

This year, an entire track was devoted to Linux, and Linus's presence was announced in a front-page article in the show daily, Update.Daily.

Last year, DECUS (Digital Equipment Computer User Society) surprised some people by bringing Linus Torvalds to New Orleans to give two short talks about Linux. Why Linux at a Digital-oriented trade show? What did Digital Equipment Corporation (DEC) and DECUS care about Linux? Why would DECUS members be interested in hearing Linus Torvalds talk about Linux?

This year, an entire track was devoted to Linux, and Linus's presence was announced in a front-page article in the show daily, *Update.Daily*. The Linux-related sessions were well-attended, not only by Linux die-hards, but also by fans of Digital Unix (née OSF/1), OpenVMS, and even Windows NT. Why?

Part of the answer is that DEC has not only given Linus a computer with DEC's new Alpha CPU in order to port Linux to it, but is actively sponsoring it, supporting it, and participating in it at the corporate level. One of DEC's booths included an Alpha-based system running Linux/Alpha as a demonstration of the success they have had with the ongoing port, and it was well received. Although the X Windowing System and networking are not yet running, Linux/Alpha is now “self-hosting” (meaning you can compile the Alpha kernel on the Alpha under Linux), and DEC projects an end-user release sometime this fall.

When asked why DEC was participating so heavily in porting a free operating system—one which could be perceived as a competitor to other DEC operating systems, especially Digital Unix—to the Alpha, Jon “Maddog” Hall, Senior Leader in the Digital Unix Marketing Group, said, “It was inevitable. Either I got the Alpha for Linus, or he was going to do the port to the PowerPC... DEC can either try and stop it and look foolish and stupid, or they can help it along and look like heroes.”

Jim Paradis, Principal Software Engineer of the Alpha AXP Migration Tools and the leader of the DEC team working on the Linux port, agreed. “If we didn't do it, *somebody* was going to do it. When you have a hot chip like Alpha in low-cost platforms, and a low-cost operating system like Linux, sooner or later, someone is going to want to put the two together. We can either be ahead of the curve or behind the curve.”

He also said that DEC needed a smaller operating system than Digital Unix, OpenVMS, or Windows NT, and they considered porting the system then known as Chicago (currently “Windows 95”) to the Alpha. However, Linux was considered a better choice, not only because it is technically superior, but also because Linux has become so popular that many users will purchase an Alpha instead of some other high-performance CPU such as a Pentium or PowerPC.

Guru

Linus himself gave several talks, including a DC-area Linux User's Group session Tuesday night, a highly-publicized “featured presentation” Wednesday afternoon, and a 5-hour Linux tutorial on Thursday. His talk on Wednesday was part of a series of 10 talks throughout the day on various Linux-related topics, and was an introduction to Linux. His well-attended tutorial was, essentially, a technical session which gave an overview of the Linux kernel. All of Linus' talks included question and answer sessions.

A Panel

The last session Wednesday was a panel discussion with several panelists, including Linus. In reality, the entire audience became the panel, and talked about relevant issues, including how local Linux users' groups (LUGS) can be helpful both for Linux users and for spreading Linux. Linus pointed out that what Linux really needs now is more mainstream commercial applications. Following are other Linux talks that took place on Wednesday: Digital and the Linux Operating System; Introducing Linux on Alpha; Parallel Processing with Linux and Transputers at YSU; Linux: Past, Present and Future; Linux in the Real World—A Powerful Idea; Distributed Computing with Linux on the Beowulf Parallel Workstation; Publishing and the Linux Market; Painless GUI Programming in Linux OS; Writing Linux Device Drivers; Linux: What is it Good for.

Linux's Wide Appeal

At the *Linux Journal* booth, we saw significant and surprising interest in Linux. At trade shows, we often see seasoned Unix users who are interested in Linux, but at DECUS, we saw a very large number of VMS users, including DEC

employees, who expressed strong interest in using Linux. Also, the Linux BOF (Birds-Of-a-Feather) session was attended by diverse users.

Next Year

The Linux track was so successful that an even larger track is being planned for December 2-7, 1995 at the Moscone Center in San Francisco. DECUS `95/San Francisco will include the possible mbone broadcast of sessions for those with high-bandwidth internet connections who can't make it to the show. The show promises to have more sessions, more symposia, and (we hope) more vendors.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Novice to Novice

Dean Oisboid

Issue #15, July 1995

This month, Dean takes some time off to play and reports his successes and failures.

I have to start this month's article with an apology. The original introduction focused on Unix as a mature operating system but one that lacked maturity in the area of entertainment. After a week of exploring the variety of Unix games, I realized my mistake. Unix games may not be as profitable or numerous as DOS games but they have been tremendously influential, and that needs mention.

The first popular games started on the mainframes. “Adventure” spawned “Zork”, which spawned the adventure game industry. On the strategy side, I believe Empire was translated from the mainframe. Mac and MS-DOS versions met with great success. Empire also influenced the strategy genre by combining a “conquer an unknown world” motif with multi-player capability, an influence that, in part, resulted in Warlords II, one of my favorite strategy games. Even today, Unix is still a major influence on the gaming world, via the Internet, through Multi-User Dungeons (MUDs). These interactive sessions began the requirement now for many non-Unix games, that they allow for multiple players over a modem or network. Many on-line services are now offering multi-player versions of the most popular MS-DOS games.

Novice to Novice will take a little recreational detour this month and explore a variety of games that may come with Linux disks or CDs. In my case, Slackware Professional 2.1. The eventual goal of this article will be to get DOOM running and use a Sound Blaster 16 because DOOM without the gruesome sounds of unbridled carnage just isn't right.

Slackware allows you to install two sets of games. One set, “Y”, contains the BSD games collection and a Tetris clone. The other set, “XAP”, contains X-Windows applications and includes some games like GNU chess and xboard (which allows you to run the ASCII GNU chess in X).

I loaded the BSD collection and soon after removed it, with one exception: Mille, a clone of Parker Brothers' popular card game Mille Borne. The binary found on the main Slackware disk had a bug that occurred when the game was extended. Sometimes there'd be an annoying buzzing lockup. However, I found a version on the tsx-11 archive disk that ran smoother. Very addictive!

There are more games than the BSD compilation available. If you snoop around on any of the archive CDs you're bound to find gold. When I found something of interest, I would copy it over to /usr/local/games. It seemed like an appropriate spot to put things. [/usr/games is for games that are installed by the system; /usr/local/games is a good choice for user-installed games —ED]

The Sunsite disk had a plethora of games conveniently divided into general categories like action, strategy, X11, and RPG (role-playing games) which was a major stopping point for me. The names sounded familiar. Some of the games like Rogue, Moria, and (Net-) Hack have been successfully translated to DOS, and in at least one case, given a major facelift. For example, SSI's Dungeon Hack is directly evolved from Hack but redone with SuperVGA graphics and a variety of other additions.

For the heck of it, I copied over Rogue. Any game with a name like my favorite comic book heroine automatically gets special consideration. The version I copied was 5.3pl2. Besides the source code, the archive had a precompiled binary, which I appreciated, but when I ran it, it crashed: “missing library”. Fine. I compiled the source cleanly and ran the new binary. It ran for a few keystrokes before stopping from “segmentation faults”. Sigh! I dismissed Rogue and looked for other entertainment.

I found something called Omega (version 0.78.1) on the TSX-11 CD that I thought would be like the DOS tank programming game of the same name; instead, it was another Hack clone. This one, unlike Rogue, ran well and my wife had to drag me away after a half hour of playing. It's easy to forget that you don't always need snazzy graphics to produce a deep game. ASCII role-playing games like this amaze me with their detail. Sure, you can add graphics, but then that 500K game explodes into multi-megabytes of size. Also, a keyboard interface can allow you a greater variety of options, which Omega has, than a mouse-based interface cluttering up a graphics screen. The problem is simply remembering the overwhelming number of commands, but a printout of the help file solves that easily enough.

Chess! I've always loved chess, not just the game but the lore and history around it. Since I had installed GNUchess and xboard, well, how about a game of chess? I ran GNUchess straight up and got the ASCII board, moved a few pieces, and lost. Admittedly my chess rating, if I had one, would be somewhere

in the low hundreds. But GNUchess worked! Now for the X-Windows interface. I activated my swapfile, entered X-Windows, called up an Xterm shell, and typed in **xboard**. Quickly a rather pretty chess board appeared. Quickly my hard drive started whirring. I made my first move and the opponent's clock started its countdown from 5 minutes. The hard disk continued to go berserk for 2.5 minutes before I destroyed it out of mercy. This long amount of time for the first move was unacceptable. Perhaps it was the swapfile. I exited X-Windows, shut off the swap, and tried xboard again. This time the board appeared along with an error message about "GNUchessx exited unexpectedly". Sigh! So much for chess. [Adding more swap would have solved the problem with xboard. — ED]

I discovered Empire (Chainsaw version 3.12) under "strategy" and debated compiling it. The README mentioned that it would require TCP/IP among other things. Sure Linux has that and more, but I backed down from compiling. I wasn't hooked up to any network; I just wanted to see what one of these games looked like without having to compile anything. Luckily, I found Conquest (5.5.1a) which, after copying to / and unarchiving, produced an executable game with little modification necessary. Running X-Windows and calling up Xterm, I typed:

```
xset +fp /usr/X386/lib/X11/fonts/misc/xconq
xset fp rehash
```

to set some appropriate paths; entering xconq produced a large window showing a very complex game. No help file was immediately available from within the game so, after moving a few pieces, I exited. Besides, it was a network game so I really couldn't have gotten the full experience from it, playing alone.

A strong case for Unix games is that for many you get the source code. If the game is too easy or too hard, or needs a wandering wizard—rewrite it! The only answer to this from the DOS side of things is the occasional release of construction sets. Generally, however, these sets do not allow you to change all aspects of a game, just what the designers will allow you to change. That's not quite like having the source code. The negative side is having the source code and not having it compile cleanly. To an expert hacker, this presents no obstacle—dive into the code and make corrections, but for novices, it's agony. That's why, when some of the above games crashed, I moved on. And on that line of thought...

On to DOOM!

On the primary Slackware disk, I found an untarred, unarchived DOOM v1.666 under /usr/games/doom. Also, I found an archived version under /sunsite/

games/X11/action/doom. Eager to play, I copied the four files (a README, a sound driver, the data file and the binary) to my own /usr/games/doom. Setting my swap file active, I launched into X-Windows, opened a terminal window, went to the proper subdirectory, typed **linuxxdoom** and waited. The terminal window showed the familiar DOOM launch lines.

I waited. My hard drive started grinding. Terminal messages gave a variety of errors which I assumed would be related to the lack of a sound driver.

And waited. A minute passed and the hard drive was going berserk. Anxiety crept into the room and waited to pounce.

And waited. A small window appeared. The opening screen of DOOM kind of showed up. It looked horribly dark and grainy, but when I clicked the mouse button, the background to X-Windows changed colors and the DOOM window suddenly, dramatically cleared up.

I was in DOOM! The movement was, surprisingly, a little jerky. I brightened up the screen via the F11 key (gamma correction). Yes, this was DOOM. I played for a bit and then quit. DOOM without the gruesome sound is gruesome itself.

The DOOM readme suggests using version 3.0 of Hannu's sound driver but fails to tell where it can be located. The Sound HOW-TO mentions using a shell script found in /usr/src/linux/drivers/sound/Readme.linux, except that no such subdirectory or file existed in my setup. Snooping with mc (Midnight Commander) in /dev, I noticed many empty files including /dev/dsp, which is used for sound. I also noticed and read through the MAKEDEV file and decided to try it.

Entering **MAKEDEV audio** did nothing—at least, nothing that I could see. The files still remained at size 0. [Device files are always 0-sized. Device files are also sometimes called “special files”, for good reason—ED]

Yes, this has all the makings of a minor crisis. Re-reading the Sound HOW-TO reveals the bad news: recompile the kernel to support sound. No, no, NO! What did I do to deserve this? I just wanted to play Doom with sound. Having to recompile the kernel just to get sound had to be insane, but it seemed that this simple yearning for DOOM had to develop into a learning experience. [Actually, recompiling your very own kernel allows you to make a smaller kernel, which gives you more memory to work with, which means that more of the games will work without the massive swapping experienced here—ED]

After a day of blaspheming a variety of Linux gods (primary and many secondary), I calmed down to tackle recompiling the kernel. The first thing I did

was copy the remaining kernel source code from the “D” set on the CD-ROM. The kernel had been split into three archives, and I had already copied over the third archive as a recommended part of general installation. Admittedly, I found it quite cool that I could fiddle with the source code if I so chose—something I know I'll **never** experience with Windows or DOS.

Next: I read the manual. It mentioned creating two links:

```
# ln -sf /usr/src/linux/include /usr/include/linux
# ln -sf /usr/src/linux/asm /usr/include/asm
```

which I changed to match what appeared under my /usr/src. I typed:

```
# ln -sf /usr/src/linux-1.1.59/include \
/usr/include/linux
# ln -sf /usr/src/linux-1.1.59/asm \
/usr/include/asm
```

which worked for me.

After setting the links, I switched to /usr/src/linux-1.1.59 and entered **make config**, which prompted me with a barrage of questions. For the most part, I accepted the default answers except when it came to SCSI, CD setup (I chose the Sony cdu31a option) and sound. Since I don't have any SCSI devices, I opted to eliminate that ability. For sound, I first accepted the default of all sound options, but when configuring the individual parts, a non-SoundBlaster value conflicted with something I specified for my Blaster. The program stopped because of that. I restarted the configuration and this time chose not to fully install all sound options. This way I got to pick just the SoundBlaster options and configure for them.

After the make **config finished**, I typed **make dep** which (according to the System Administration Guide) fixes all of the source dependencies. No problems here.

Finally the big moment: **make Image** to create the actual image. No, it stopped, saying the command wasn't supported anymore and instead to type **make zImage** to create a compressed version.

I did so and the computer started grinding away. About ten minutes later it crashed: lack of virtual memory. Dumb error on my part! Setting my swap file active with **swapon /swap**, I restarted the compiling and relaxed as the grinding began.

And continued. 5 minutes. 10 minutes. 30 minutes. About an hour later it finished and **finished cleanly**. No problems! My chest puffed out in manly pride, and I drank deeply from a virtual beer.

Now what to do with the two resulting files called zSystem.map and zImage? Using **mc**, I found a similar zSystem.map under /boot. I renamed it and copied the new one in its place. A quick perusal of the System Administrators Guide revealed that sometimes the kernel is called vmlinuz and sure enough I found such a file under /. I renamed it and copied zImage as vmlinuz to /.

[This was enough because Dean uses UMSDOS and does not use LILO. The majority of Linux users, who use LILO to boot, will need to run **lilo** in order to make their new kernel bootable. Many documents, including the article Dean mentions below, document how to do this—ED]

The new kernel turned out smaller—about two-thirds the size of the old kernel. Hopefully this will translate into a slight gain in speed. I also discovered—actually I remembered—that *Linux Journal* had an article in Issue 7 about compiling the kernel, called "*Linux Performance Tuning for the Faint of Heart*". A quick review revealed that I answered the various kernel questions correctly. Very comforting and also informative. The article explained many of the default choices.

Time for the real test: shutdown Linux and reboot. The kernel loaded cleanly and apparently recognized the SoundBlaster. Great! I set up the swap file, fired up X-Windows, called up a term window, got into /usr/games/doom and ran **linuxxdoom**. It did its stuff, grinding and YES, YES, YES, DOOOOOOOOM WITH SOUND!!!!! It worked!

For a while, I thought it wouldn't work because there were many error messages during loading saying "**can't find lumpname ( )**", but it worked!

Now, one more thing to test: whether DOOM would allow me to use an alternate .WAD file (data file) with the game. One of the great appeals of Doom is that the program allows you to use other data files, thus giving you good value for your $US40. People have created 1000s of new levels, sounds, and effects for Doom, Doom II, and the newest family member, Heretic. I doubted that I could use an alternate WAD with this shareware version of Doom because it is version 1.666. Supposedly, the earlier 1.2 version did have the external WAD support, but that meant users wouldn't necessarily buy the full version. Id Software, the publishers, removed that ability in the later shareware versions. Only registered users get to enjoy swapping WADs. I happen to have a WAD handy, so let's try it out:

```
doom -devparm -file acheron.wad -wart 1 3
```

The program starts and ends quickly. As I expected the shareware DOOM v1.666 won't run external WADs. I can vouch that the registered Unix version will allow that ability (having played it on a friend's Sun network).

A Parting Rant

Unix, for all its maturity as an operating system, still lacks success in the entertainment field primarily because businesses and schools, the primary users of Unix, don't require it. Not to deny that Unix hasn't influenced the gaming field—it undeniably has. But Unix still is sold as a high-end solution and hence is priced out of the home market where the entertainment market has its largest hold. Besides, why would anyone run Unix at home?

And yet, with Linux, a compact and essentially free Unix, the home market opens up. If Linux, or any other similar operating system, can create a valid and popular niche in the home market then the scenario changes. With the home market opened up, we see a fantastic new potential for Unix-based entertainment.

Anybody want to design a killer Linux game?

Dean Oisboid , owner of Garlic Software, is a database consultant, Unix beginner, and avowed Strike Commander addict. He can be reached at 73717.2343@compuserve.com

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

New Products

LJ Staff

Issue #15, July 1995

Unix Cockpit, InfoMagic and Lasermoon Release Moo-Tiff and more.

Unix Cockpit

UniX11 Software Co. has released the Unix Cockpit (UC) version 1.2 for all major Unix platforms, including Linux. UC is a new Unix/X11 file manager. File browsers, a directory tree, custom menus and a standard command shell are smoothly integrated into one highly customizable multi-window productivity tool. UC is $25 shareware for Linux and available via anonymous ftp.

Unix Cockpit may be retrieved from ftp.uu.net in /vendor/UniX11/linux/, or from ftp.uni-wuppertal.de in /pub/unix/cockpit/. For more information, e-mail henrik@UniX11.com or snail mail to UniX11 Software C., Moorbachweg 7, 83209 Prien, Germany.

InfoMagic and Lasermoon Release Moo-Tiff

InfoMagic, Inc. and Lasermoon, Ltd. have released Moo-Tiff, a port of OSF/Motif 2.0 for Linux, on CD-ROM for $99. Versions are provided for X11R5 (XFree86-2.1.1) and X11R6 (XFree86-3.1.1). The X11R6 version is provided in both a.out and ELF format. A copy of XFree86-3.1.1 is also included on the CD. The complete OSF documentation set is included on the CD in postscript format (User's Guide, Programmer's Guide, and Widget Writer's Guide). In addition to mwm, shared and static libraries, and header files, the CD includes both source and binaries of the OSF/Motif demo programs and a large collection of Motif “freeware”. Moo-Tiff comes with free technical support via phone, e-mail, and fax.

For more information contact: InfoMagic 800-800-6613 or 620-526-9565 or e-mail info@InfoMagic.com or www.infomagic.com/. Lasermoon +44 (0)329-826444 or info@Lasermoon.co.uk

CODEC for Linux

CODEC, a multiplatform compression utility (COmpressor/DECompressor), allows the user to compress in one operating system and decompress in another, interfacing the physical format of the file between the same or different environments with regards to file organization, record format, record length and block size. For example, it is possible to compress in IBM's MVS and to decompress in a PC. It is also possible to create on any platform a file capable of self decompressing in any one of the allowed operating systems (DOS, OS2, U*IX, VMS, etc.) with self exploding features.

At the URL www.nettuno.it/fiera/telvox/telvox.htm CODEC multiplatform version 3.21 is available as shareware for most common operating systems. CODEC for Linux is distributed as freeware.

BGI Graphical Toolkit for Linux

Borland C and Turbo C users can now easily port their DOS graphical applications to Linux with the availability of BGI Graphical Toolkit from G & Y Systems. The toolkit is a high-level graphical library that provides full source compatibility with Borland BGI, and contains other functions that aid in porting applications from DOS to the Linux environment. The toolkit is fully integrated with the Linux console, and supports transparent graphical-to-text mode switching whenever a console change is detected. It features an integrated mouse driver and support for SVGA adapters based on popular chip-sets such as Trident, Tseng Labs ET-4000, Western Digital, Cirrus, Oak, and others. Toolkit code is highly optimized and features separate optimization for 16 and 256 color modes. Borland-compatible stroked fonts are also fully supported.

Demonstration program and the shareware version of the toolkit are available on sunsite.unc.edu. The full version is available from G & Y Systems, and includes all the sources to the toolkit. Hardcopy documentation will be available in the 3rd quarter of 1995. For more information, contact G & Y Systems, P.O. Box 4925, Foster City, CA 94404, phone or fax 415-638-0703, or e-mail g_and_y@golem.com

Stock Charting Tool for Linux

BB Tool is a stock charting, technical analysis and portfolio management tool with a Motif interface. Key features include automatic stock split detection, customizable technical indicators, customized watch list, extensive portfolio tracking, transaction record keeping, “most active issues” and “big price movers”, etc. In addition, Falkor Technologies provides daily quote and utilities to update your historic quote database. BB tool for Linux costs US$79. Historic stock data on the US Stock Exchange are available at US$1.00 per stock. A daily

stock quote is available free, distributed via e-mail. Free demo version and PostScript formatted document is available via FTP download at ftp.portal.com:/pub/ctor. WWW URL: www.portal.com/~ctor/bb.html For more information, e-mail ctor@shell.portal.com, or call 510-505-0700.

Object REXX for Linux

IBM will soon be releasing (free of charge) a Linux port of its new object-oriented version of its REXX language. REXX is a scripting language that was designed to be easy to use. It is available on most IBM platforms and is also being ported to Windows. In addition, the source code will be made available for Linux to enable porting of the code to other platforms. Object REXX extends the REXX language with classes, objects, and methods. It supports messaging, polymorphism, and multiple inheritance, while supporting existing REXX programs. You can try object-oriented programming without having to make a major investment in it. Instead, you can intermix the use of objects with traditional REXX instructions.

For more information call Tim Browne at 607-752-6030 or e-mail at BROWNE@GDLVM7.VNET.IBM.COM.

Object REXX is a trademark of IBM Corp. Windows is a trademark of Microsoft Corp.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Installing the Xaw3D Libraries

Mark Komarinski

Issue #15, July 1995

Make those graphics come alive with the 3-Dimensional replacement set Xaw3D.

After months of being stuck in my CLI (Command Line Interface) mode, I finally broke down and installed X11R6 on my Linux machine. It's nothing to write home about, really, just a 386/40 which three years ago was state of the art. Oh well. At least I have graphics and can finally see the neat pictures on the *Star Trek: Voyager* WWW site. It also gave me a chance to put my mouse to work after it collected months of dust.

Taking the defaults that XFree 3.1.1 gave me, I found myself inside fvwm and its Motif-like interface, wishing that other programs also had a Motif-like interface. Motif has many advantages, but the drawback is that it costs money that many of us don't have (yet). One of the features that Motif provides is a 3-D interface for buttons and other portions of a window known as widgets. These widgets are designed such that the developer of the software determines what widgets to use and how they work, and the end user is allowed to change how the widgets are displayed. This is similar to the way that X without Motif works now.

To make your screen more 3-D like, there is a drop-in replacement for the Athena Widget set called Xaw3D. The Athena Widgets define how the scroll bars on the side of the screen work. By default, the widgets make the scroll bar for a program like xedit look like what you see in Figure 1. Functional, yet somewhat bland.

The replacement 3-D libraries make xedit look more like Figure 2. You'll notice three areas where Xaw3D has made its presence known. The buttons on the top of the window, the scroll bar on the left, and the three section markers on the top right-hand area.

You can get the 3-D widgets in most of the popular Linux FTP sites, including sunsite.unc.edu as /pub/linux/libs/X/libXaw3d_X11R6.tar.gz. There are also libraries for Xfree 2.1 along with the Xaw3d sources in the same directory.

The installation is pretty straightforward, but you have to be root. There are instructions in the libXaw3d_X11R6.readme file. Tar and uncompress the libXaw3d_X11R6.tar.gz file with:

tar -zxvC / -f libXaw3d_X11R6.tar.gz

then update the shared libraries:

ldconfig

The -C tells tar to untar in the root directory. Since the tar file contains the directory structure to put the libs in usr/X11R5/lib, the -C / will place the file in the correct directory without having to move it all around. Note that this will replace the libXaw.so.6.0 file that already exists there, so you may want to back it up. The ldconfig will update the file links for you, so you don't need to make the common mistake of erasing the library by using the ln command incorrectly.

Once this library is installed, this will only affect dynamically linked libraries. There are other packages available in the same directories as above that provide the libraries for statically-linked programs, should you need them: Most users probably won't.

The next time you start X-Windows, you'll notice changes in the scroll bar and other widgets that normally use the Xaw set. You can also make changes to the appearance of these widgets, using the ~/.Xdefaults file to configure Xaw.

You can use the **shadowWidth* resource to change how each widget displays itself, and you can see the results of your changes by merely saving the file to ~/.Xdefaults and then starting up a new program that uses scroll bars or buttons that are statically linked. Two good examples for this are xterm and xedit.

The **Label*shadowWidth* resource defines the depth of the label widgets to make them appear 3-D. You can add a shadow by setting this number, for example, to 2.

One of the best ways, however, to figure out what each widget does is to borrow someone else's configuration. Michael K. Johnson was kind enough to

provide me with a copy of his .Xdefaults, and I've made my own revisions, which are shown in Figure 3.

! For Xaw3d! gives everything that Motif look to it. Sort of.*background: LightSkyBlue*Form.background: grey90*TransientShell*Dialog.background: bisque3*Command.background: gray80*Menubutton.background: gray80*ScrollbarBackground: gray60*Scrollbar*background: gray70*Scrollbar*width: 16 *Scrollbar*height: 16*Scrollbar*shadowWidth: 2*Scrollbar*cursorName: top_left_arrow! You can change the above to all kinds of icons, including gumby.! Find out what icons are available by looking in! /usr/include/X11/cursorfont.h - be sure to strip off the XC_*Scrollbar*pushThumb: false*ShapeStyle: Rectangle*beNiceToColormap: false*shadowWidth: 2! The above defines all shadowwidths to 2. *Label*shadowWidth: 2! The above overrides the definition from a few lines above for Label! widgets.*SmeBSB*shadowWidth: 2*highlightThickness: 0*topShadowContrast: 20*bottomShadowContrast: 40! The above two lines define how the shadows appear on the top and! bottom of the widgets. The higher the number, the lighter it is. This! particular setting makes the top and left sides darker than the! bottom and right sides. Figure 3. File .Xdefaults

Pulldown menus can have the **SmeBSB*shadowWidth* resource set, which will change the display of pull-down menus.

For more help on how the resources or Athena Widgets work, look at the man pages for the various standard X applications (such as xterm or xedit).

Mark Komarinski graduated from Clarkson University (in very cold Potsdam, NY) with a degree in computer science and technical communication. He now lives in Troy, NY, spending much of his free time working for the Department of Veterans Affairs where he is a programmer.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.