# *Linux Journal* Issue #47/March 1998


### *Features*

Programming with XView *by Michael Hall*
: This article gives you a high-level introduction to programming with XView, a GUI toolkit that complements the OpenLook interface.

CDE Infrastructure *by George Kraft IV*
: The programming infrastructure, no its productivity tools, is a major strength of the Common Desktop Environment. This article discusses the APIs and desktop services that are benefitting developers and independent software vendors.

AfterStep 1.3.1 *by Guylhem Aznar*
: Mr. Aznar tells us all about the developers' plans for a friendly window manager called AfterStep.

Introducing TkDesk *by John Blair*
: Don't want to give up your Macintosh or Window desktop for Linux—with TkDesk you don't have to.

An Introduction to the GIMP Tool Kit *by Otto Hammersmith*
: The purpose of this article is to give a short overview of what gtk+ is, what it can do and where to gather more information.

### *News & Articles*

Linux Network Programming, Part 2: Creating Daemon Processes *by Ivan Griffin and John Nelson*
: In part 2 of our series we learn how to design and code network daemons to serve our clients well.

### *Reviews*



### *WWWsmith*



### *Columns*

# Programming with XView

**Michael Hall**

Issue #47, March 1998

This article gives you a high-level introduction to programming with XView, a GUI toolkit that complements the OpenLook interface.

There are many window systems out there in the computer world—some are taking control of the desktop, while others are still strong in their workstation niche. Just as most of us would prefer to program in a high-level language rather than assembly, XView provides us with a high-level approach to GUI development. XView components come with default settings that make sense, so that a newly created button, for example, will look and act “correct” as soon as it is made. The API for XView is at once both simple and extensive. Simple, because we can do most jobs using only three different library routines, and extensive because we can control nearly all aspects of each component with those routines.

## First Steps

Let's jump right in and take a look at the simple XView program shown in Listing 1. Make sure you have XView and the OpenLook libraries on your system; otherwise, you'll get compile errors. Put this example into a file called sample.c and compile it using the following command:

```
cc -o sample -I$OPENWINHOME/include\
-L$OPENWINHOME/lib\
-L/usr/X11/lib sample.c -lxview -lolgx -lX11
```

where OPENWINHOME is the home directory on your system for OpenWindows, usually /usr/openwin.

When you run sample.c, you'll be presented with a simple text editor, a functionality of the standard XView text sub-window component. Type in the text window, using the keyboard cursor keys to move around. Highlight words, lines and paragraphs with mouse clicks and drags. The right mouse button

brings up a menu to open, save and insert files, exit the text and search for strings. Grab the end-stop of the slider and pull it down—you now have two views of the same text.

This is truly neat, but how did we get here? Let's walk through the sample program together.

First, we include two header files: one for generic XView information and one for the text sub-window component. If we had additional components (perhaps a pair of push buttons to open and save files), we would also include the header file for those panel components.

Next, inside of our main routine, we declare a **Frame** variable, which is used by XView as a place to put other components. It becomes an invisible layer between our text sub-window and the window decorations from the window manager.

Next, we call **xv_init** to initialize XView and pass the command-line arguments to it. (Check the **xview** man page for all the command-line options.) With this xv_init call, we first come upon a strong theme that runs through all XView calls —a null terminated list of attribute, value pairs. The first argument to xv_init is defined in the XView headers. **xv_init** expects to see two more arguments: the address of main's argument count and the argument values. To tell xv_init that we have nothing more for it to do, we pass one final argument of **NULL** to terminate the argument list.

On the next line, we create our frame, using **xv_create**. In the first argument we specify the parent of this component. Since we're just creating the outermost frame, we have no parent, and we use **XV_NULL**. The second argument specifies the type of component we want to create, in this case a **FRAME**. Notice that the remaining arguments form a null-terminated list of following attribute, value pairs. For example, the following attribute, value pair:

```
FRAME_LABEL, argv[0]
```

sets the label of the frame, which becomes the text on the window's title bar. If we wanted to set the size of the frame on the screen, we would simply add two pairs to set the width and height (before the terminating NULL):

```
XV_HEIGHT, 200,
XV_WIDTH, 400,
```

Similar attributes are used to set the maximum and minimum sizes of the frame, the initial position on the screen, the header and footer, etc.

Moving on, we create the text sub-window. We pass in our frame variable to this **xv_create** call, because we want the frame to become the parent of the text sub-window, affecting its size and position. The type of component we're creating is **TEXTSW** for a text sub-window, and we are adding no other options to it (although many are available).

Next, we call **xv_main_loop**, which handles all X events for the window that appears on the screen. It calls lower-level **xlib** routines to create the window, slider and canvas. It listens for mouse and keyboard input and handles it. **xv_main_loop** doesn't return until the window is closed by the user. At that point, our program can quietly exit.

Before we go further, there are two other important XView routines we need to know: **xv_set** and **xv_get**. As their names imply, they are used to set and to get attribute values for XView components.

In sample.c, we saw how to set the size of the frame when it was created. We also could have set the size after it was created with the **xv_set** call in this way:

```
xv_set(frame,
        XV_HEIGHT, 200,
        XV_WIDTH,  400,
        NULL);
```

Although the vertical formatting isn't necessary, it shows more clearly what is happening: namely, using the **frame** component, we have set its height to 200 and its width to 400.

Suppose that at some point in the program, we needed to determine the position of the frame on the screen:

```
int     x, y;
x = (int)xv_get(frame, XV_X);
y = (int)xv_get(frame, XV_Y);
```

This fragment retrieves the x and y coordinates of the frame on the screen. Yes, it looks like we violated the rule of using a null-terminated list, but since xv_get returns a single value, we fetch only one attribute; therefore, xv_get accepts only two arguments.

### Second Version with Callbacks

Let's extend this sample program by adding two buttons to it. One button will insert a fixed text string into the text sub-window, and the other button will erase the text sub-window. To do this, we will need a place to put them: namely, a control panel which is implemented in the PANEL package. The buttons themselves are panel items called PANEL_BUTTONs. We associate a subroutine, or “callback” routine, with a button, to be called when the user

clicks on the button. The callback routines will manipulate the text sub-window. The second version of the sample.c program is shown in Listing 2.

Let's start with the new code inside our main. We created a panel inside the frame, positioned in the upper-left corner (x=0, y=0), extending to the right edge, 30 pixels tall and borderless.

Next, we added two buttons to the panel (not the frame), each with different button labels and different callback routines. For this example, XView handles the chore of positioning the buttons within the panel. If we wanted, we could use the XV_X and XV_Y attributes to position the buttons within the control panel.

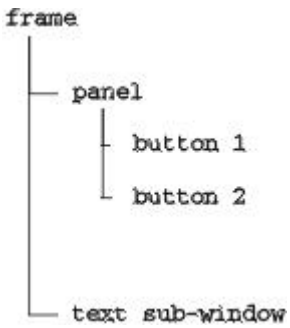

Figure 1. Hierarchy in the Window

Note that a hierarchy (see Figure 1) is forming. The frame is a parent of the panel and the text sub-window. The panel is a parent of the two buttons. When we resize the frame, its child components resize with it. The user interface could become quite sophisticated (i.e., complex) but still remain manageable, because of the relations that form among the frames, panels and other components. Our callback routines are invoked by other routines deep within XView and are passed the component and the event that produced this call (in this simple case, the button and the mouse-button-up event).

For the **insert string** callback routine, we use the global handle to the text sub-window and call an auxiliary routine to insert the literal text into the text pane. For the **clear_window** callback routine, we use another helper function to reset the text sub-window, which erases all the text from its pane. Although we use xv_set and xv_get to manipulate the attributes of the XView components, some components have a nice set of helper functions to make our job easier. The text sub-window is one such component.

Although this sample program doesn't give you an earth-shattering application, it does show you the core features of XView:

- attribute,value pairs
- null-terminated lists

- xv_init and xv_main_loop for setup and event handling
- xv_create, xv_set, xv_get for component attributes
- callback functions for event handling

This article has demonstrated the simplicity, elegance and beauty of XView. Perhaps you will be inspired to look into it further.

XView Components

Resources

Mike Hall is a senior consultant with BALR Corporation, a Chicago-area computer consulting firm. His last assignment used XView and display PostScript on Suns. He can be reached at mghall@balr.com.

Archive Index Issue Table of Contents

Advanced search

# CDE Infrastructure

**George Kraft IV**

Issue #47, March 1998

The programming infrastructure, not the productivity tools, forms the major strength of the Common Desktop Environment. This article discusses the APIs and desktop services that benefit developers and independent software vendors.

When Project Athena at Massachusetts Institute of Technology (MIT) introduced the X Window System (X), they established the mechanism for their distributive "open" windowing protocol. Yet they intentionally left out the policy for others to develop. Later, based on technology from Hewlett-Packard (HP) and Digital Equipment Corporation, the Open Software Foundation (OSF) developed the policies of the Motif Graphical User Interface (GUI) for X. The Motif style guide established strict guidelines for its "human-centric" windowing behavior; however, the Unix desktop was still left without services such as data typing, object methods, plug and play and hypertext help.

The Unix System Laboratory licensed the Destiny Desktop with System V.4, HP developed HP Vue for HP-UX and IBM shipped IXI's X.desktop for AIX. Although these desktops provided services not offered in X and Motif GUI, they caused a divergence in the open systems market. The cost of integration and the need to remain portable kept many developers from migrating to these desktops.

CDE for Linux is offered by three major Linux distributions: Red Hat (*Red Hat CDE*, Don Kuenz, *LJ* January 1997), Xi Graphics (*AcceleratedX CDE and Display Server for PC Unix*, Bradley Willson, *LJ* November 1997) and Caldera.

## The Common Desktop Environment

The Common Desktop Environment (CDE) was the result of a collaboration between HP, IBM, Sun and Novell to establish a unified open system desktop. They took the "best of breed" technology from their existing environments,

then designed, implemented and delivered a new level of Unix desktop services on which developers could rely.

### CDE Programming Infrastructure

CDE's Application Programming Interfaces (APIs) go beyond the basic Motif GUI. CDE provides data-typing, object methods, printing, drag and-drop, additional widgets, plug-and-play messaging and a hypertext help system. This new set of system services at the desktop level is guaranteed for all CDE-compliant systems. Developers can now more easily produce consistent applications and reduce the number of developer-specific core solutions.

### Motif GUI

Motif won the open-system GUI wars; however, vendors have serviced and enhanced various releases of Motif. Although Motif became the GUI standard, portability became risky if any of the vendors' embellishments were used. Fortunately, the creators of CDE helped eliminate those problems by merging their Motif source bases to re-establish a stock GUI.

### Desktop Services

CDE goes beyond the GUI to provide desktop-wide objects and methods for applications to use and call upon. Applications no longer need to depend on old and limited databases like the mime-types and mailcap that are used by applications such as mailers and web browsers.

Listing 1 shows a message dialogue program that initializes itself with the desktop and loads the desktop's data-type and method database. Then, it simply creates a message dialogue and prompts the user. When you select the E-mail button, the application calls the desktop's **Compose** method on the file object. The desktop spawns the mailer with the file object, where it is eventually displayed and ready to be addressed.

### Drag-and-Drop

The desktop provides a convenient and consistent drag-and-drop API for interpreting data transfer across the desktop. Text, file names and buffers can be transferred from the dragged icon to the drop zone. The type of data being dragged determines the drag icon's appearance and configuration. Since Motif can distinguish the different types of data, applications have a more robust drag-and-drop behavior.

### Desktop GUI

The desktop's DtWidget library helps bridge the current gap between CDEs in the current Motif and Motif 2.0. Developers do not need to wait for Motif 2.0 because the spin box, menu button and editor widgets are in CDE. As always, to maintain binary compatibility, developers should take special care to use **XmResolvePartOffset** when subclassing from one widget to make another; otherwise, an updated shared library could cause unpredictable results.

### Help

CDE provides standard help APIs and GUI dialogues, but it also delivers a feature-rich SGML DTD (document type definition) compared with the more limited HTML DTD used on the World Wide Web. CDE's help links support hypertext, definition, man page, execution and application-define links. CDE documents are pre-processed for quicker loading. Since these binary-formatted documents are not human-readable, one of their added benefits is that they cannot be reverse engineered when copied, thus preventing copyright infringements. Publishers who provide documents in HTML format are at a disadvantage because complete unabridged duplicates can be made from most browsers.

### ToolTalk Plug-and-Play

CDE's ToolTalk is a message brokering system that enables applications to communicate with each other without having direct knowledge of one another. Application clients and servers can be developed independently, mixed and matched and upgraded independently through plug-and-play. Applications registered to handle message requests act as servers for applications that broadcast their requests. Message brokering is an evolutionary step beyond file sharing, peer-to-peer and ICCCM inter-client communication.

### Graphical Korn Shell

The Graphical Desktop Korn Shell provides much of the desktop's Motif GUI, services, help, workspace management, session management and ToolTalk plug-and-play. Developers can prototype and deploy with the standard ksh93 scripting language. This means that small to moderate-sized programs can be written, then interpreted on any CDE-compliant system without any additional work.

The **dtksh** shell script in Listing 2 is a conversion of the C program that was illustrated earlier. Unlike the popular Tcl/Tk shell and GUI, dtksh has a nearly one-to-one migration path to native Motif for performance. With dtksh, code

can be easily migrated, and developers find that their knowledge transfers easily between C and dtksh.

### The Desktop's Infrastructure

CDE not only provides a new set of Motif, Drag-and-Drop, Desktop Widget, Help, ToolTalk and DtKsh APIs, but it also provides system services in which applications can participate and follow. The desktop services provide the login manager, session manager, color server, workspace manager and ToolTalk server. It is tempting to provide these features in large software suites; however, if developers try to mimic these desktop services, precious development time and energy is taken away from creating the actual products.

### Login Manager

The desktop login manager provides the basic X display manager protocol (XDMCP) to manage login sessions for X terminals on the network and workstations on the desktop. The login manager starts up the X server on the bitmap display; it also initiates the session manager.

### Session Manager

The session manager uses a set of conventions and protocols that enable the desktop to save and restore a user's session from one login session to the next. Using the session manager, users can also configure a set of **sessionetc** and **sessionexit** scripts to be called when logging in and exiting respectively. This enables user-defined tasks to be performed at login and logout.

The key responsibility of the application is to acknowledge the WM_SAVE_YOURSELF message when the desktop is being shut down by the user, as shown in Listing 3. The WM_SAVE_YOURSELF saveProc routine tidies up for the application, then sets the WM_COMMAND property to be saved and reused later by the session manager to restart the terminated applications.

### Color Server

The session manager acts as the color server that controls foreground and shadow colors, limits color use and restricts the creation of colors in the color map. Using applications that conform to the color server reduces the depletion of the color map and coordinates the color scheme of the desktop. If the color map does become depleted, an “unsocial” application is usually lurking somewhere.

## Workspace Manager

The desktop window manager (dtwm) serves as the default window manager for the desktop and extends the capabilities of the Motif window manager. It provides a control panel for the desktop to launch applications. Its multiple screens, or workspaces, allow users to switch between screens.

Desktops are often considered mutually exclusive end-point solutions, such as web browsers or collaborative software suites. However, CDE views them as application groups or workspaces being managed and serviced by the desktop infrastructure. The desktop window manager is ideal for managing workspaces for Internet suites and collaborative tools.

## ToolTalk Server

The ToolTalk server can be relied upon to broker client and server messages for inter-client plug-and-play. The common object request broker architecture (CORBA) movement has a great deal of strength behind it. But ToolTalk is lightweight, does the job and forms an integral part of CDE, which is deployed across a gamut of Unix platforms. ToolTalk takes message-handling registrations from method servers, then holds that information until client applications broadcast for those registered services. Platform gateways are not needed because ToolTalk interoperates between heterogeneous systems.

## Go For It

There is much more to CDE than meets the eye. Application and workspace developers alike have a rich feature-filled infrastructure upon which to build. Just like in the past, we still rely on the tried and true X and Motif; however, now we can count on the Common Desktop Environment for its development libraries and desktop management infrastructure.

If you are getting ready to write a new application or just thinking about sprucing up something that you have been working on, consider how your application can become more feature rich and desktop friendly with less code.

For a list of CDE reference materials, visit IBM's CDE web page at http://www.rs6000.ibm.com/software/OS/CDE/further.html.

George Kraft is an Advisory Software Engineer for IBM's Network Computer Division. He has previously worked on CDE V2.1 and V1.0 for IBM's RS/6000 Division. He initially started working with the X Window System for the Purdue University Computing Center back in 1987. He has a BS in Computer Science and Mathematics from Purdue University. He can be reached via e-mail at gk4@cactus.com.

Archive Index Issue Table of Contents

Advanced search

# AfterStep 1.3.1

**Guylhem Aznar**

Issue #47, March 1998

Mr. Aznar tells us all about the developers' plans for a friendly window manager called AfterStep.

It seems like only yesterday that I was asked to write an article about AfterStep for *LJ*'s special issue on GUIs (graphical user interfaces), and now here it is. The main points I will focus on are the following:

- History of window managers and in particular AfterStep
- The future of AfterStep
- Desktop philosophy of AfterStep 1.3.1

My conclusion is that the best GUI is the most productive one, but only if it remains user friendly. Users must be free to do anything easily using their window manager.

## History

The first window manager was TWM from Evans and Sutherland, and it was the base upon which Robert Nation created FVWM. FVWM has been used to create most of the modern window managers: FVWM2, Enlightenment, amiWM and Bowman. FVWM2 was used as the basis for FVWM95, and Bowman evolved into AfterStep. In this “tree” hierarchy, some exceptions appear: WindowMaker, the NeXT emulation window manager; and KWM, the KDE project window manager. Both of these managers were written from scratch.

Is starting at the beginning better than modifying an already existing window manager? It may bring personal satisfaction, but it will also create duplication of effort. Since all of these programs (FVWM, Bowman, etc.) are freely redistributable, there's no need to write the same code twice. If a feature you want already exists in another window manager, just take the code and patch it into yours. This method will save you both time and effort. FVWM already had

most of the features of the modern window managers. The concept has progressed only slightly, while the appearance has radically changed.

AfterStep, derived from both FVWM and Bowman, was first written by Dan Weeks, Frank Fejes and Alfredo Kojima to emulate the NeXT GUI. When they realized they couldn't put all the features they wanted into AfterStep, a new project was started—WindowMaker. WindowMaker is targeted at Gnustep (GNU approach to NeXT programming and interface) with very innovative concepts, e.g., dock, DnD and “sticky” menus.

### The Future

Even if it limits you in some ways, the ability to build on a good base is very useful. You don't spend most of your time coding a good equivalent of FVWM with many bugs. AfterStep is now evolving toward new heights, including simpler and more powerful configuration. Rather than maintaining AfterStep and allowing other managers to pass it, we (the developers) want it to become the most innovative window manager. For example, we're planning to add a few new features. First, concepts that already exist in other window managers that we plan to incorporate into AfterStep:

- A file manager in the root window (see Win95 and KDE) with an Irix-like desktop
- WindowMaker Dock and “sticky” menus
- Enlightenment-like window decorations with pixmaps
- FVWM95 task bar
- OS/2-like voice recognition (see the ear voice recognition project on Sunsite)

The code for these features from other window managers can be copied under the GPL (GNU Public License). I strongly believe in and encourage maximum source code re-usage. I like the FVWM95 task bar, Enlightenment decorations and WindowMaker dock DnD; they're very useful and easy to use. Mixing and matching concepts from several window managers will ultimately result in a “best-of-all-worlds” situation. That goal is why some people install and configure everything they find. Merging these features into a single program is a much more efficient solution. Therefore, the idea is to make AfterStep more friendly by including many features into a homogeneous work. I hope we'll be able to meet most individual tastes. And there's no need for coding in a new way (C++, Gnustep, Qt, etc.) to meet such goals since most of these cool features exist in FVWM parented window managers.

**Figure 1. AfterStep Screenshot**

I'd also like to see some new features that are not yet implemented in any window manager, with maybe a new way of coding. That's why AfterStep will first integrate the FVWM family features and then take a different path. New concepts that we wish to add are as follows:

- NeXT desktop, in collaboration with the WindowMaker team
- Module compatibility with FVWM2 and other window managers
- Object-oriented management of AfterStep elements (title bar)
- Animated icon and opaque window rotation
- Total mouse configurability
- Icons/help files/programs database (For example, a painting program without a nice default icon would “auto-magically” use a database icon containing a button for calling **xman** with the program's help file.)
- Source code optimization to keep AfterStep from becoming slow

The sky is the limit. If you implement a good option in AfterStep, just send me your patch and if it's freely redistributable it'll be in the next version. If you think AfterStep is going to become 100MB, don't worry—all of these are compile time options. A patch is put in the executable program only if you request it (that's what we call meeting personal tastes). We're making AfterStep for ourselves, but one thing we also want is features we haven't already imagined. All this may seem quite ambitious, and we agree, we are. AfterStep's “pilgrim fathers” already did a lot, making such a good program, so it's not easy to make it even better.

**Figure 2. AfterStep Screenshot**

### Philosophy

AfterStep 1.3.1 already includes user-friendly configuration and “look sharing”. Here are some examples of the new features already working:

- Take your personal look file, rename it to look.your-name and post it to your friends. They can put it in desktop/looks and select it in the start menu to easily get the appearance of your desktop without losing their own menus, configuration or bindings.
- The start menu is simply a directory in ~/gnustep/Library/AfterStep in which each subdirectory is a menu entry and each file a menu option. If you want to add Ghostview, just type:

  ```
  echo "gv &" >\
  ~/gnustep/Library/AfterStep/start/Programs/Ghostview
  ```

  Spaces are allowed and you can specify command-line options. When you restart AfterStep, Ghostview will be added to the start menu.

- Backgrounds are put in the directory ~/gnustep/Library/AfterStep/desktop/backgrounds. Put the picture you want there and when you restart AfterStep, the Desktop/Backgrounds/Pictures menu will contain this option. Of course, if you select it, AfterStep will remember it for the next session, just like colors (in desktop/colors) and look files.
- An included patched **rxvt** called **xiterm** allows pictures to be displayed in the background with Offix DnD compatibility (just like the Wharf). Select a file in **xfm**, drag it to xiterm, and its name will appear on the command line you're typing. It's lighter than xterm, supports color and will be more linked with AfterStep in version 1.3.2, becoming a standard part of its GUI, like title bars, Wharf and Pager. Configure-time options are made via **menuconfig** (just like the Linux kernel); this will also be true for future versions of AfterStep.
- Four 2x2 pagers (that's sixteen screens, but you can configure it if you want more or less space on your desktop) with individual background pixmaps and names to allow you to use many programs at the same time (the task-list is very useful too).
- Jump button allows you to focus on the next window by clicking on the right triangle in the right part of the title bar.

As you have learned in this article, AfterStep is strongly linked to the FVWM family but is no longer restricted to NeXT GUI emulation. Before moving to totally new concepts, we're first trying to put in all the nice features that already exist without making AfterStep hard to use. I hope you'll find 1.3.1 as simple to use as I think it is. But remember, if you think of a missing option, code it and send in the patch. With look files and other configuration files, AfterStep can turn into anything you want it to be.

**Guylhem Aznar** is a medical student. He likes swimming the medley, listening to rock and classical music and programming (xiterm, linuxbbs, AfterStep). He writes HOWTOs for the LDP (French and BBS) and maintains a few other HOWTOs. He can be reached at guylhem@danmark.linux.eu.org.

Archive Index Issue Table of Contents

Advanced search

# Introducing TkDesk

**John Blair**

Issue #47, March 1998

Don't want to give up your Macintosh or Window desktop for Linux? With TkDesk you don't have to.

TkDesk, developed over the past several years by Christian Bolik, is currently one of the most powerful desktop managers for Unix and Unix-like environments. Christian points out in his documentation that the system was designed especially for Linux, which is primarily where I use it. However, I also use it successfully under AIX and reports have been favorable under Solaris and other Unix flavors.

## A Tour

**Figure 1. TkDesk Screen Shot**

**Figure 2. File Browser**

Rather than provide a user's manual or specific customization tips, which you can learn by reading TkDesk's excellent documentation, I'll describe in this article why I think TkDesk is such a useful and important Unix tool. As an example, this morning I logged into my computer through **xdm**. All I have scripted to start automatically is TkDesk and a few utility applications, like **xautolock** and **xsysinfo**. After seeing the TkDesk startup screen, I am presented with the AppBar (see Figure 1) and a few file browsers (see Figure 2), positioned where I left them the last time I was logged in. From the mailbox icon I see that I have e-mail. Clicking the mailbox starts **exmh**, and I read the messages that have arrived since yesterday. Someone has found an error in a CGI script. Figuring it won't take long to find the problem and fix it, I start up XEmacs and Netscape, again by clicking icons in the AppBar. I point one of my file browsers at the directory containing the CGI script and another to a development directory. By clicking and dragging with mouse button two, I make a copy of the script into my development directory (see Figure 3). I then drag this copy onto

the XEmacs icon in my AppBar, which loads the script into my previously started XEmacs session. Looking through the code I realize that I may have flipped the arguments to a CGI module function that I called. Clicking with button three on the Library icon in the AppBar shows my customized documentation menu (see Figure 4). I select the reference to my locally installed Perl documentation. A Netscape window opens up listing the directory and I click on the cgi.pm.html link. Seeing my error, I make the necessary changes. I also need to check another Perl script. This time, instead of dragging the script onto the XEmacs icon, I click the file once to select it, then choose the "Open File New Frame" I have configured under the XEmacs menu. This creates a new XEmacs window displaying the script.

**Figure 3. Copy Operation**

**Figure 4. Documentation Menu**

After double checking this code, I finally have to start a shell so I can test the changes I made to the script off-line. Clicking with button three on the file browser listing that represents my development directory causes a pop-up menu to appear, much as clicking on an icon in Windows 95 or Windows NT 4.0 with the right mouse button causes a menu to appear. I select the "Start XTerm Here" option and an XTerm window opens with the working directory set to this location. When I am satisfied with the script's behavior I use **su** to go to the WWW administrative account and install the script. Unfortunately, TkDesk doesn't provide me with an easy way to accomplish this last step—at least not yet.

Last, curious if anyone is currently accessing the web server, I run the **netstat** command. However, I don't run it from the command line. I choose the "netstat" option I have set under the System Statistics icon, which opens the "Periodic Execution" tool, running the netstat command (see Figure 5). The Periodic Execution tool, in this case set to run netstat every 30 seconds, is one of several general purpose tools which TkDesk provides.

**Figure 5. Screen Shot of Periodic Execution Tool**

### Taming the Anarchy of Unix

This scenario demonstrates only a small subset of the wide variety of tasks I regularly conduct quickly and efficiently using the TkDesk desktop manager. I find this account significant because this is not how most people think of interacting with Unix. Being a long-time Macintosh user, I have always been convinced that a well-designed GUI is more efficient than the command line for most activities I conduct on my computer. Unfortunately, until TkDesk came along I hadn't found a Unix file manager that I felt made me any more efficient

than working within the shell. A possible exception to this is NextStep. This is appropriate since the look and feel of TkDesk borrows heavily from the NextStep Dock and file browser. The NextStep file browser did deal poorly with non-NextStep native files and applications when I last used it, circa NextStep 2.0. Things may have changed since then.

The reasons for not having an efficient desktop manager are varied, but chief among them is the difficulty of dealing with the relative anarchy of the Unix environment. A monolithic GUI desktop manager works well only within the context of a well-defined API. The Macintosh Finder and the Windows Explorer as of Windows 95 can depend on nearly every application to support basic operations, such as opening a file, in the same way. Obviously, Unix contains no such API. To most Unix gurus this provides a level of flexibility that is considered a strength, not a drawback.

Instead of breaking in the face of this anarchy, TkDesk is able to effectively work with it. Unlike the other monolithic file managers, TkDesk was written almost entirely in Tcl/Tk. This means that instead of relying on a specific class of procedures to deal with files, TkDesk can react to each file type differently by referencing a chunk of Tcl code. An example of this can be seen in how TkDesk deals with XEmacs and Netscape when I double click on a file ending in .html; TkDesk loads the file into Netscape using the Netscape-Remote extensions to Tcl. However, when I drag the same file onto the XEmacs icon or choose the XEmacs option I have added to the pop-up menu associated with HTML files, TkDesk calls the **gnudoit** function to send a load command to XEmacs. Dragging a group of documents onto the Netscape icon works equally well, since a small “foreach” loop applies the operation to each file, causing a Netscape window to appear for each file. This technique of encapsulating chunks of Tcl code behind icons and menus is used to define most of the behavior of TkDesk. Since each of these chunks of code is accessible in the many TkDesk configuration files, changing the behavior of TkDesk is fast and easy.

I think the best way to understand TkDesk is to think of it as a kind of universal file selector. TkDesk provides you with a clean, intuitive interface to apply specific commands to different types of files. The built-in commands supported by TkDesk are the types of operations you would expect from any complete file manager: copying, deleting, moving, renaming, linking and symbolic linking. The delete command, like any self respecting desktop manager, can be used to copy files into a “trash can” before permanently deleting them.

A file's *type* is determined by matching it with a global pattern. One configuration file, the FileTags file, maps global patterns to the icons, fonts and colors which describe how the file will appear in the file browser. Another configuration file, the PopUps file, maps global patterns to lists of Tcl

commands which will be available when one clicks on the file with mouse button three.

This design allows you to tie specific menu options, icons and other behavior to a file or directory by matching its extension, as one does under Windows. However, it also lets TkDesk react to other types of file patterns, such as Emacs checkpoint files, README files and Makefiles in specific ways. With TkDesk I can now unpack a source distribution, run the configure script and build the system without opening a command prompt. A handful of Tcl functions that could be considered the TkDesk API make writing these code fragments easier and more consistent. One function is used to run a command in the background, displaying notice in the status bar of the file browser when the program starts and exits. This same function adds this command to the command history menu. One function allows you to display your own messages in the status bar. Other functions allow you to ask for confirmation or for strings to be entered, to display the output of programs in the built-in editor, to play sounds, to start commands in the periodic execution tool and more. A “Local” configuration file is even provided should you wish to write your own functions for use in the other configuration files (and I'm sure most readers of this article will sooner or later feel the urge to add their own functions).

While the core of TkDesk is the AppBar and file browser, a collection of smaller tools is also supplied. My favorite is the convenient help viewer, which can parse the text representation of a Linux HOWTO document (see Figure 6). One of the initial default configurations is a global pattern to recognize gzipped HOWTO documents, unpack them and display them in this help viewer.

Figure 6. Help Viewer

Figure 7. Screen Shot of Information Box

Another tool is the built-in file information box that provides a quick way to view the attributes of any given file, change the access mode of the file, “touch” the file, and more (see Figure 7). The built-in editor is comparable to Windows Notepad or Macintosh TeachText and provides a quick way to view and scan text files. The **find file** tool encapsulates the varied options of the Unix find command behind an easy to use dialog box, including filtering all the found files through **grep**. As you would expect, the results of the find command are displayed in a list which provides access to the same pop-up menus available from the file browser (see Figure 8).

Figure 8. Find File Command

## Closing Thoughts

Despite this glowing review, TkDesk is not without its flaws. The most obvious complaint is its speed. One frequently notices operations that slow down because of the large amount of Tcl being interpreted behind the scenes. I usually choose not to display the icons in the file browser—updating the icons in a large file list often takes too long. However, this is likely to change very soon. Tcl 7.6 and earlier versions were internally based on no more than substituting strings over and over again. While this allows for the design of a very simple interpreter, it does cause a performance hit in frequently executed code. Tcl 8.0 has an internal byte-code interpreter and provides speed comparable to Perl. Unfortunately, the object-oriented extension to Tcl, [incr tcl], that TkDesk is written in, is not yet ported to Tcl 8.0. The work to port [incr tcl] to Tcl 8.0 is ongoing and may be finished by the times this article goes to press. Once [incr tcl] works with Tcl 8.0 we can expect a big speed increase in TkDesk.

I hope that TkDesk is not the last word in GUI desktop managers for Unix and, in particular, Linux. Several other projects, like the K Desktop Environment and GNUStep show much promise. Just as TkDesk currently demonstrates, I think the future of GUI design lies in tying the widgets that these projects provide to a flexible scripting language like Tcl, GUILE or other scheme variants, Python or some yet-to-be-invented scripting language. The success of the GIMP project is another testimony to the success of this type of design. For now, though, TkDesk is the ruler of my desktop.

Resources

**John Blair** is currently a system administrator in The University Computer Center at the University of Alabama at Birmingham where he tends several Unix and (shock, horror) Windows NT servers. By the time this article is published he may be working someplace else. John is also the author of SAMBA: Integrating Unix and Windows, published by the same people who bring you *Linux Journal*. Feel free to contact him via e-mail at jdblair@uab.edu.

Archive Index Issue Table of Contents

Advanced search

# An Introduction to the GIMP Tool Kit

**Otto Hammersmith**

Issue #47, March 1998

The purpose of this article is to give a short overview of what gtk+ is, what it can do and where to gather more information.

The GIMP tool kit **gtk+** is the GUI libraries written for use in the GIMP as an alternative to Motif. (See the GIMP series of articles by Michael Hammel in the November to January issues of *LJ*.) Although gtk+ was created for the GIMP, it can be used for any GUI application. In fact, recently the gtk+ distribution has been split out from the GIMP distribution in anticipation of the 1.0 release.

## What Can gtk+ Do?

Like all GUI tool kits, gtk+ provides a variety of GUI components, a.k.a. widgets. One thing that sets gtk+ apart from tool kits such as Motif is the breadth of the widgets provided. Naturally, all tool kits provide some form of a button widget. **gtk+** also provides components such as tabbed notebooks and fully functional color pickers not included in the core widgets of other tools kits. Screen shots are provided in Figures 1 and 2 as examples of the appearance of gtk+ widgets.

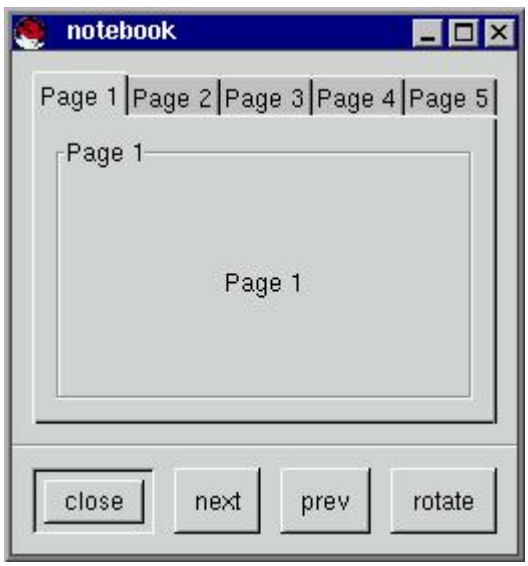

Figure 1. Notebook Widget

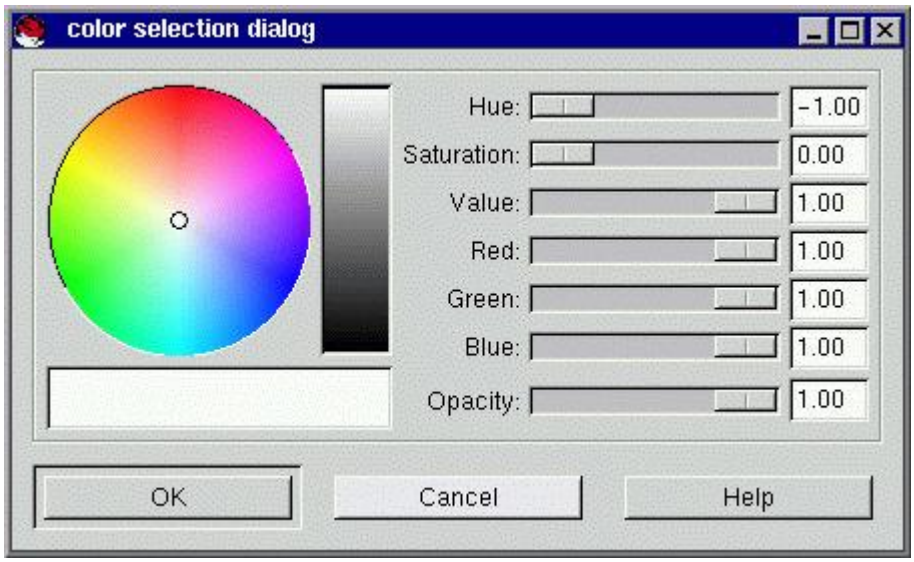


Figure 2. Color Selection Widget

Another advantage of gtk+ is that it's not limited to a single language. Even though gtk+ is written in C and was originally intended for use from C, there are bindings to many other languages. Just a few of them are C++, Objective C, Python, Scheme, Perl—the list goes on and on with new additions frequently. Chances are someone has already started on bindings for your favorite language.

Best of all, gtk+ is under the Library GNU Public License (LGPL), so there are absolutely no licensing concerns when writing software.

### The Pieces

The GIMP tool kit is made up of three different layers. Each layer has its own distinct functionality and depends on the layers below it. The three layers are:

1. **glib**: This is a support library that is of general usefulness in many C programs, even non-GUI applications. Most of its utilities are common data structures, such as linked lists (**GList**), and convenience functions, to ease porting between Unices, e.g., wrappers around **malloc** and **free** (**gmalloc** and **gfree**).
2. **gdk**: This library is basically a thin layer over Xlib and some of Xt to handle the basics of drawing on the X display and handling X events. Having this library means that applications writers never have to see the underlying X libraries to get work done. This library also contains several useful functions for things like handling timers, polling file descriptors and more.
3. **gtk**: This library is the meat of gtk+, where all the GUI elements are defined. This library relies heavily on gdk and glib to do its job.

## Learning gtk+

For someone with experience in C programming, gtk+ should be fairly easy to pick up. The internals of gtk+ use some advanced C features like function pointers and some tricks with structs, but application writers should never need to worry about the internals of gtk+. Naturally, any experience with other GUI tool kits (especially Motif) will make learning gtk+ easier.

For a long time the only documentation of gtk+ was the source code, and the sample application, the GIMP. Fortunately, the gtk+ tutorial at http://levien.com/slow/gtk/ has filled that void.

However, gtk+ is a moving target (still being developed) so be wary. If there's any doubt about the validity of some documentation, always double check it against the source code. And, as always, nothing can beat actually writing code to test an option out.

There is also a mailing list dedicated to discussing gtk+. To subscribe, send mail to gtk-list-request@redhat.com with a subject of “subscribe *email-address*”, replacing *email-address* with your e-mail address.

## gtk+ Status

When learning about gtk+, remember that it is still under development, but it is certainly usable for non-trivial applications, the GIMP, for example. There are still plenty of minor bugs and even a few major ones you'll have to work around. For example, at the time of this writing, the text widget still needed a lot of work. But in the two months or more before anyone will read this article in print a lot can happen. The best place to check on the current status of the gtk+ libraries and to get the source is at http://www.gimp.org/gtk/.

Otto is a developer at Red Hat Software who is currently very busy with the next release of Red Hat Linux. He can be contacted by e-mail at otto@redhat.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL



# Linux Network Programming, Part 2

**Ivan Griffin**

**John Nelson**

Issue #47, March 1998

In part 2 of our series we learn how to design and code network daemons to serve our clients well.

Daemon processes are servers which run in the background servicing various clients. You should be aware of the following few issues when creating daemon processes. During development, it is always advisable to run the server in the foreground in order to use **printf** or **write** for debugging. Also, if the server happens to go berserk, it can be killed by sending the interrupt character (typically **ctrl-c**). While being deployed for use, the server should be coded to act like a daemon. Daemon programs in Unix typically end in the letter *d*, e.g., **httpd** for the HTTP daemon (web server).

It is always nice to have a daemon automatically put itself in the background when run. This is quite easy to achieve using the **fork()** call. A well-behaved daemon will close all the open file descriptors it inherited from its parent after the fork. This is especially important if the files are terminal devices, as they must be closed to allow the terminal state to reset when the user who starts the daemon logs out. Use the **getrlimit()** call to determine the maximum number of open file descriptors and to close them.

The process must then change its process group. Process groups are used in distributing signals—those processes with the same group as the terminal are in the foreground and are allowed to read from the terminal. Those in a different group are considered in the background (and will be blocked if they attempt to read).

Closing the controlling terminal and changing the session group prevents the daemon process from receiving implicit (i.e., not sent by the user with the **kill** command) signals from the previous group leader (commonly a shell).

Processes are organized within process groups and process groups within sessions. With the **setsid()** system call, a new session (and thus, a new process group) is then created with the process as the new session leader.

Once the daemon is without a controlling terminal, it must not re-acquire one. Controlling terminals are automatically acquired when a process group leader opens a terminal device. The easiest way to prevent this is to fork again after calling setsid(). The daemon runs in this second child. Since the parent (the session and process group leader) will terminate, the second child will obtain a new process group of zero (since it becomes a child of **init**). Therefore, it cannot acquire a new controlling terminal, since it is not a process leader. Many standard library routines may assume the three standard I/O descriptors are open. As a result, servers commonly open all three descriptors, connected to a harmless I/O device such as /dev/null.

Daemons are typically started at boot-up and remain running throughout the uptime life of the system. If a daemon was started from a mounted file system, it would be impossible to unmount the file system until the daemon was killed. As a result, it is a sensible daemon programming practice to perform a **chdir()** to **/** (or perhaps to a file system which holds files relevant to the operation of the daemon). Daemons inherit **umask** information from the process which created them. To prevent problems with file creation later within the daemon, it is commonly set to zero using **umask()**. Listing 1 illustrates these points with some sample code.

For systems that do not support sessions (e.g., some systems other than Linux and Solaris), you can achieve the same result as setsid() using the code from Listing 2.

When the forked children of the main server code exit, their memory is deallocated but their entries in the process table are not removed. In other words, the processes are dead, i.e., they do not consume system resources, but they have not been reaped yet. The reason they stay around in this *zombie*-like form is to allow the parent to gather statistics from the child if necessary (such as CPU usage, etc). Obviously, a daemon does not want to fill the process table with zombie processes.

When a child dies, it sends its parent a **SIGCHLD** signal. The default handler of this signal causes the child to turn into a zombie, unless it is explicitly reaped by its parent, as in Listing 3. Alternatively, as shown in Listing 4, you can ignore the signal and allow the zombie to die.

It is also quite common for daemons to ignore most other signals or to re-read any configuration files and restart after being sent **SIGHUP**. Many daemons

save their PID (process identification) to a log file, typically called /var/run/foobar.pid (where **foobar** is the name of the daemon), which can aid in stopping the daemon.

When the system is being shut down (or changing from multi-user to single-user), the **SIGTERM** signal is sent to notify all processes. The **init** process then waits a specific amount of time (20 seconds for SVR4, 5 seconds for BSD, 5 seconds for Linux init, 3 seconds or a passed command-line argument Linux **shutdown**). If the process is still alive, a **SIGKILL** signal which cannot be ignored is sent to it. Therefore, daemon processes should catch the **SIGTERM** signal to ensure they shut down gracefully.

### Network Daemon Designs

**Figure 1. Three Designs for a Network Service Daemon**

In Figure 1, the diagrams show three potential designs for a daemon providing a network service to prospective clients. In the first picture, the daemon follows the most common technique of forking off a separate process to handle the request, while the parent continues to accept new connection requests. This concurrent processing technique has the advantage that requests are constantly being serviced and may perform better than serializing and iteratively servicing requests. Unfortunately, forks and potential context-switches are involved, making this approach unsuited to servers with very high demand.

The second diagram shows the iterative, synchronous, accepting and handling of a request within a single context of execution, before another request is handled. This approach has the drawback that requests which occur during the handling of the request will either get blocked or rejected. If blocked, they will be blocked for at most the duration of the request processing and communication. Depending on this duration, a significant number of requests could potentially get rejected due to the listen queue backlog having filled. Therefore, this approach is perhaps best suited to handling requests of a very short duration. It is also better suited to UDP network daemons rather than TCP daemons.

### Process Pre-allocation

The third diagram (Figure 1) is the most complicated—it shows a daemon which pre-allocates new contexts of execution (in this case, new processes) to handle the requests. Note that the master calls fork() after **listen()**, but before an **accept()** call. The slave processes call accept(). This scenario will leave a pool of potential server processes blocking an accept() call at the same time. However, the kernel guarantees that only one of the slaves will succeed in its accept() call

for a given connection request. It will then service the request before returning to the accept state. The master process can either exit (with **SIGCHLD** being ignored) or continually call **wait()** to reap exiting slave processes.

It is quite common for the slave processes to accept only a certain number of requests before committing suicide to prevent memory-leaks from accumulating. The process with the lowest number of accepted requests (or perhaps a special manager parent) would then create new processes as necessary. Many popular web servers implement pools of pre-forked server threads (e.g., Netscape, Apache).

### Delayed Process Allocation

If the server process time of a request is very short (the usual case), concurrent processing is not always necessary. An iterative server may perform better by avoiding the overhead of context-switching. One hybrid solution between concurrent and iterative designs is to delay the allocation of new server processes. The server will begin processing requests iteratively. It will create a separate slave process to finish handling a request if the processing time for that request is substantial. Thus, a master process can check the validity of requests, or handle short requests, before creating a new slave.

To use delayed process allocation, use the **alarm()** system call, as shown in Listing 5. A timer is established in the master, and when the timer expires, a signal handler is called. A fork() system call is performed inside the handler. The parent closes the request connection and returns to an accepting state, whereas the child handles the request. The **setjmp()** system call records the state of the process's stack environment. When the **longjmp()** is later invoked, the process will be restored to exactly the same state as saved by the setjmp(). The second parameter to longjmp() is the value that setjmp() will return when the stack is restored.

### Threading

All of the forking in these examples could be replaced with calls to **pthread_create()** to create a new thread of execution rather than a full heavyweight process. As mentioned previously, the threads should be kernel-level threads to ensure that a block on I/O in one thread does not starve others of CPU attention. This involves using Xavier Leroy's excellent kernel-level Linux Threads package (http://pauillac.inria.fr/~xleroy/linuxthreads/), which is based on the **clone()** system call.

Implementing with threads introduces more complications than using the fork() model. Granted, the use of threads gives great savings in context-switching

time and memory usage. Other issues come into play, such as availability of file descriptors and protection of critical sections.

Most operating systems limit the number of open file descriptors a process is allowed to hold. Although the process can use getrlimit() and setrlimit() calls to increase this up to a system-wide maximum, this value is usually set to 256 by **NOFILE** in the /usr/include/sys/param.h file.

Even tweaking **NOFILE** and the values **NR_OPEN** and **NR_FILE** in the /usr/src/linux/include/linux/fs.h file and recompiling the kernel may not help here. While in Linux the **fileno** element of the **FILE struct** (actually called **_fileno** in Linux) is of type **int**, it is commonly **unsigned char** in other systems, limiting file descriptors to 255 for buffered I/O commands (**fopen()**, **fprintf()**, etc). This difference affects the portability of the finished application.

Because threads use a common memory space, care must be taken to ensure this space is always in a consistent state and does not get corrupted. This may involve serializing writes (and possibly reads) to shared data accessed by more than one thread (critical sections). This can be achieved by the use of locks, but care must be taken to avoid entering a state of deadlock.

### Problems with init

**init**'s primary role is to create processes from information stored in the /etc/inittab file. It is either directly or indirectly responsible for all the user-created processes running on a system. It can respawn processes it starts if they die.

The respawning capabilities of init will get quite confused if the daemon forks as per the code in Listing 1. The original daemon process will immediately exit (with a child daemon continuing to run), and init will take this to mean the daemon has died. A simple solution is to add a command-line switch to the daemon (perhaps **-init**) to inform it to avoid the forking code. A better solution is to start the daemon from /etc/rc scripts rather than from the /etc/inittab.

### SVR4 Style /etc/rc

The System V layout of /etc/rc is used in the popular Red Hat and Debian distributions of Linux. In this system, each daemon that must be started/stopped has a script in /etc/rc/init.d for Red Hat and in /etc/init.d for Debian. This script is invoked with a single command-line argument **start** to start the daemon, and a single command-line argument **stop** to stop the daemon. The script is typically named after the daemon.

If you want to start the daemon in a particular run level, you will need a link from the run-level directory to the appropriate script in /etc/rc/init.d. You must

name this start link S*xxfoobar*, where *foobar* is the name of the daemon and *xx* is a two digit number. The number is used to arrange the order in which the scripts are run.

Similarly, if you want the daemon to die when changing out of a particular run level, you will need a corresponding link from the run-level directory to the /etc/rc/init.d script. The kill link must be named K*xxfoobar*, following the same naming convention as the start link.

Allowing system administrators to start/stop daemons (by calling the appropriate script from /etc/rc/init.d, with the appropriate command- line argument) is one of the nicer advantages of the SysV structure as well as its greater flexibility over the previous BSD-style /etc/rc.d layout.

The shell script in Listing 6 shows a typical Red Hat style example in /etc/rc/init.d for a daemon called foobar.

### Using syslog()

It is often useful for a daemon to log its activities for debugging and system administration/maintenance purposes. It does this by opening a file and writing events to this file as they happen. Many Linux daemons use the **syslog()** call to log daemon status information etc. The syslog is a client-server logging facility, originating from BSD 4.2. I am not aware of any SVR4 or POSIX equivalent. Messages to the syslog service are generally sent to text files described in /etc/syslog.conf, but may be sent to remote machines running a **syslogd** daemon.

Using the Linux syslog interface is quite simple. Three function calls are prototyped in /usr/include/syslog.h (see the syslog.3 man page):

```
void openlog(char *ident, int option, int
        facility);
void syslog(int priority, char *format, ...);
void closelog(void);
```

**openlog()** creates a connection to the system logger. The **ident** string is added to each message logged and is generally the name of the daemon. The **option** parameter allows for logging to the console in case of error, logging to stderr as well as the console, logging of the PID and so on. The **facility** argument classifies the type of program or daemon logging the message and this defaults to **LOG_USER**.

The syslog() call does the actual logging. Values for **format** and the variable arguments are similar to printf(), with the exception that **%m** will be replaced by the error message corresponding to the current value of **errno**. The **priority**

parameter indicates the type and relative importance of the message being logged.

To break the connection with the system logger and close any associated file descriptor or socket, use **closelog()**. The use of openlog() and closelog() is optional. More detailed information on these functions is available in the syslog (3) man page.

Resources

BSD daemon()

**Ivan Griffin** is a research postgraduate student in the ECE department at the University of Limerick, Ireland. His interests include C++/Java, WWW, ATM, the UL Computer Society (www.csn.ul.ie/) and, of course, Linux (www.trc.ul.ie/~griffini/linux.html). His e-mail address is ivan.griffin@ul.ie.

**Dr. John Nelson** is a senior lecturer in Computer Engineering at the University of Limerick. His interests include mobile communications, intelligent networks, Software Engineering and VLSI design. His e-mail address is john.nelson@ul.ie.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL



# The SANE Scanner Interface

**David Mosberger**

Issue #47, March 1998

SANE makes it easy to support a wide variety of devices and of applications with a minimum amount of programming effort.

SANE stands for “Scanner Access Now Easy”. It is a universal interface that enables you to acquire images from any device that produces raster images, including flatbed scanners, video and still cameras and frame grabbers. The intent of SANE is to make it possible to write image-processing applications without having to worry about peculiarities of individual devices. Looking at it from the other side, SANE makes it possible to write a device driver once and only once. That same device driver can then be used by any SANE-compliant application.

## Introduction to SANE

Have you ever wanted to scan an image under Linux? If so, you probably know the feeling of being faced by a bewildering number of scanner-driver packages. At the time of this writing, at least fourteen different scanner packages exist for Linux. While each individual package is usually of high quality, it is often difficult to determine which package should be used for which scanner. Furthermore, some packages come with a command-line interface, others include Tcl/Tk based graphical front ends, still others come with full-featured, graphical front ends. While variety is said to make life sweet, in this case it's more likely to cause a sizeable headache.

SANE was created to provide a solution to this dilemma. The basic idea is simple: if there was a general and well-defined application programming interface (API), it would be easy to write applications independently from scanner drivers. Thus, the author of a new scanner driver would not have to worry about writing an application for the driver. There are benefits for the application programmer as well; since SANE is universal, an application can be written independently of the devices that it will eventually control. Suppose we

wanted five applications to support ten different devices. With the old approach, 5*10=50 programs would have to be written. With SANE, only 5+10=15 programs have to be written. SANE has advantages for the user too. It gives the user the liberty to choose whichever application he likes best, and that one application can be used to control all image-acquisition devices the user can access. Thus, SANE makes it possible to present the same consistent interface independent of the particular device that is in use.

Of course, SANE is not the first attempt to create such a universal interface. You may have heard of TWAIN, PINT or the Linux hand-scanner interface. The problem is that these older interfaces prove to be lacking in one way or another. For example, PINT is really a somewhat primitive kernel-level interface and the hand-scanner interface by definition is limited to hand-scanners. In contrast, SANE is general enough to support any device that acquires raster images. The closest thing to SANE is probably TWAIN. The fact that the two rhyme is not coincidental, but that's a different story. The main reason TWAIN is not SANE is that TWAIN puts the graphical user interface to control the device in the driver instead of the application. This makes it unsuitable for Linux or networked environments where the scanner driver might run on one machine and the application on another. In contrast, SANE enforces a strict separation between the actual driver and the user-interface for its controls. Indeed, the current SANE distribution includes support for network-transparent scanning.

## Using SANE

To start using SANE, fetch the latest version of the distribution from the ftp directory ftp://ftp.mostang.com/pub/sane/.

If you want to build the graphical-user-interface programs that come with SANE, you will also need to fetch, build and install the GIMP or, at a minimum, the GTK distribution. Both GIMP and GTK are available at ftp://ftp.gimp.org/. GTK is the user-interface toolkit that originally has been developed for the GIMP, but is now being adopted by many other projects, including SANE. Note that the SANE distribution will build just fine without the GIMP/GTK libraries. However, that way none of the nice graphical-user-interface programs will be built, thus taking away much of the fun. So, unless you are building SANE for a server only, I recommend that you install at least GTK, if not GIMP.

After fetching the SANE distribution, unpack the compressed tar file and follow the instructions in the README file. The README explains how to build and install SANE. Also, take a look at the file called PROBLEMS; it contains a list of known problems and their work arounds.

Note that you don't have to own a scanner or a camera to play with SANE. The distribution includes a pseudo-driver that simulates a scanner by reading

portable “anymap” (PNM) files. Also, SANE is not limited to Linux. Besides Linux for Alpha, x86 and m68k, it includes support for AIX, Digital Unix, FreeBSD, HP-UX, IRIX, NetBSD, SCO, Solaris, SunOS and even OS/2.

After installing SANE, you should be able to type the command

```
scanimage --list-devices
```

and get the output shown below:

```
device `mustek:/dev/scanner' is a Mustek MFC-06000CZ flatbed scanner
device `pnm:0' is a Noname PNM file reader virtual device
device `pnm:1' is a Noname PNM file reader virtual device
```

As the listing shows, in this particular case, a Mustek scanner is available under name mustek:/dev/scanner and two fake devices called pnm:0 and pnm:1 are available that can be used to read PNM files. To get list of all options for a particular device, for example pnm:0, simply type:

```
scanimage --device pnm:0 --help
```

This will produce the help message shown in Listing 1.

The SANE package comes with a detailed man page that explains the specifics of the **scanimage** program. As an example, suppose we had a PPM file named input.ppm. We can use the scanimage program to “scan” that image and increase its brightness by 50% using the following command:

```
scanimage --device pnm --brightness
50 input.ppm > out.pnm
```

If you look at file out.pnm with an image viewer such as xv, you should be able to see that output.ppm is noticeably brighter.

You may say: cool, but where is the graphical user interface? Assuming you had the GTK libraries installed when SANE was built, you can invoke a program called **xscanimage** that will present you with a dialog box containing a list of available devices. If you double-click on the “pnm:0” entry, you'll get the dialog shown in Figure 1. As you can see, the dialog includes two text-entry boxes labeled “Filename” and a slider labelled “Brightness”. If you enter “out.pnm” in the first text-entry box and “input.ppm” in the second box and move the brightness slider to 50.0, you can press the Scan button at the bottom left and get the same result as with the scanimage command line shown above. Of course, before doing the actual scanning, you could press the Preview button at the bottom right to pop up a preview window (see Figure 2). In the preview window, you can push the Acquire<\!s>Preview button to obtain a low-resolution preview of the final image. For example, by moving the brightness slider around, you can see how the brightness of the image is affected. After

moving the slider, you'll need to press the Acquire Preview button to get an updated preview.

Figure 1. SANE Dialog Window

Figure 2. SANE Preview Window

When scanning an image with a real scanner or camera, you'll usually want to enhance it in various ways, such as making it appear sharper. The nice thing about the xscanimage program is that it can also be run as a GIMP extension. To do this, simply create a symlink from the GIMP plug-ins directory to the xscanimage binary. Assuming the SANE installation defaults, you could do this with the following command:

```
ln -s /usr/local/bin/xscanimage ~/.gimp/plug-ins
```

After making this link, xscanimage will attach itself to the GIMP's “Xtns” menu the next time you start it. This makes it possible to invoke, for example, the PNM pseudo-device by selecting “Xtns**->**Acquire Image**->**pnm:0”. When invoked in this manner, pressing the Scan button will put the newly scanned image inside a GIMP window (instead of saving it to disk). Now, the usual GIMP image-manipulation functions can be used to enhance the acquired image before saving it.

2395f1.gif

Figure 3. Mustek Dialog Window for xscanimage

The PNM pseudo-device may be fun, but what does a real scanner interface look like? Figure 3 shows the xscanimage dialog as it appears for Mustek flatbed scanners. The figure also demonstrates another feature of xscanimage: tool tips (also known as “balloon help”). Tool tips make it easier for new users to get acquainted with the capabilities of their scanner or camera. In the figure, the mouse points to the Scan<\!s>Source menu and, as a result, the help information for that menu is shown in the yellow box below the mouse pointer. Tool tips are handy for new users, but after a while, they tend to get in your way. Thus, xscanimage allows advanced users to turn off the tool tips using the Preferences sub-menu.

As you can see, the Mustek dialog looks quite different from the PNM pseudo-device interface. This is because the underlying devices have different capabilities. In fact, the device dialog depends not only on the selected device, but also on the mode of the device. For example, when turning on the “Use custom gamma table” option near the bottom of the dialog, the interface changes, and the result is shown in Figure 4. As you can see, the right half of

the dialog now contains a graph editor that allows the user to modify the intensity, red, green or blue gamma table. In other words, xscanimage displays precisely the options that are active or meaningful for a given scan mode, greatly reducing the likelihood of confusing the user.

2395f2.gif

Figure 4. Mustek Dialog With Gamma Table Editor

Looking at the image-intensity gamma table in the right half of the figure, you can probably imagine that it would be rather annoying to define the gamma tables each time you started xscanimage. Once the ideal tables have been found, it would be nice if it were possible to save them. For this purpose, xscanimage allows saving the current device settings through an entry in the Preferences sub-menu. Once saved, whenever xscanimage is started, it automatically restores the last saved option values for that device.

### What Else Comes with SANE?

Now that you have seen how to use some of the programs that come with the SANE distribution, it is time to tell you what else is included. At the time of this writing, the package includes drivers for the following devices:

- Connectix QuickCam (color and monochrome)
- Some Epson SCSI scanners
- Hewlett-Packard ScanJet SCSI scanners
- Microtek SCSI scanners
- Mustek SCSI flatbed scanners (both one-pass and three-pass scanners are supported)
- PINT devices: PINT is a Unix-kernel interface for NetBSD, OpenBSD and SunOS. SANE's PINT driver provides access to any scanner for which there is PINT support
- Most UMAX SCSI scanners

Support for many other scanners and cameras is planned and some of them should be ready by the time you read this article. For the latest information, please visit the web page listed in the Resources.

Available applications are the command-line **scanimage**, the graphical **xscanimage** (either stand-alone or as a GIMP extension) and **xcam**, a graphical user interface suitable for cameras which produce a continuous stream of images (such as the Connectix QuickCam).

In addition, there are SANE API bindings for Python and Java API and a network daemon called **saned** that provides network-transparent access to remote devices. Assuming you have the appropriate permissions, this makes it possible to control a camera running in the U.S. from a machine running in Europe—all courtesy of SANE and the Internet.

### How Does It Work?

When building a SANE application, it must be linked against the shared library called libsane.so. In reality, libsane.so is just a symlink to one of the SANE drivers. Since every SANE driver exports the exact same interface, you can change the libsane.so symlink at any time and effectively change which driver the application is using. While this is useful in the sense that it allows upgrading to a different driver without having to relink all the applications, it would not be very convenient if you had to change a symlink whenever you wished to switch the scanning device. For this reason, SANE supports two pseudo-drivers called **dll** and **net**. They are pseudo-drivers because rather than talking to physical devices, they talk to other SANE drivers, as illustrated in Figure 5.

For machine A, the libsane.so symlink points to the dll pseudo-driver (called libsane-dll.so). That pseudo-driver uses dynamically linked libraries (dll) to access other SANE drivers. In the example, dll is configured to use the pnm, mustek and net drivers. The net driver is again a pseudo-driver; it provides access to remote scanners by connecting to the SANE daemon (saned) running on machine B. Machine B in turn uses dll again to provide access to a variety of other drivers. As you might imagine, the exact configuration is entirely up to the system administrator(s) of machines A and B. It is fairly typical to have libsane.so be a symlink to the dll pseudo-driver, but there is no reason it couldn't point to the net pseudo-driver or just the mustek driver. Of course, in the latter case the implication would be that applications could access the mustek driver only—but that's perfectly reasonable for certain environments.

**Figure 5. Possible SANE Hierarchy**

This approach is very flexible, but it raises an interesting question: how do we name devices in such an environment? The answer is that every real driver has its own device name space. For example, the Mustek and HP drivers use the path for the Unix special device that controls the device, such as /dev/scanner. With pseudo-drivers, things get a bit more interesting. Since dll must guarantee that each device name is unique, it simply prefixes the name of each subordinate device with the name of the subordinate driver, separated by a colon. Thus, on machine A, the mustek scanner would be called mustek:/dev/scanner. The net pseudo-driver does something similar: it prefixes the remote device name with the remote host name (again using a colon as a separator). For example, HP scanner 1 on machine B would appear on machine A under

the name net:B.domain.com:hp:/dev/scanner1. While this doesn't make for the world's prettiest names, the information contained in the names is actually quite useful. In essence, much like a Unix path name, the device names convey the path through the SANE hierarchy that leads to a particular device. For example, if you know that machine B is down, it's pretty obvious that net:B.domain.com:hp:/dev/scanner1 will be down as well. If someone feels strongly about these names, it is possible for an application to let a user or system administrator define aliases that are more concise. For example, an application could let a user rename the above device to “HP Scanner 1”, which may be easier for beginners.

### Programming With SANE

By definition, SANE is only as good as the programs that use it. This means the more applications and the more devices that use SANE, the merrier. The SANE distribution comes with a detailed document that explains the SANE API; however, the interface is quite simple. The six main functions are listed below:

1. handle **<- sane_open**(*device-name*) allows you to open a SANE device by name (e.g., pnm:0).
2. **sane_close**(handle) allows you to close a SANE device by name.
3. **sane_get_option_descriptor**(handle, *option-number*) is used to query the controls available to the device (such as the brightness control in the PNM pseudo-device driver).
4. **sane_control_option**(handle, *option-number*, *action*, *value*) is used to get or set the value of an option. For example, it can be used to set the value of the brightness option to 50 percent. In addition, some options support an auto-mode where the driver picks a reasonable value. For such options, sane_control_option() can also be used to turn auto-mode on or off.
5. **sane_start**(handle) is used to start the acquisition of an image.
6. bytes-read **<- sane_read**(handle, *buffer*, *buffer-size*) is used to read the actual image data until the entire image has been acquired.

The SANE API is simple by design. The goal was to make it possible to accomplish a simple task in a small amount of time while still providing enough functionality to enable sophisticated drivers and applications. The simplicity is best evidenced by the fact that it took just two evening sessions to convert the **hpscanpbm** program into a SANE driver for HP scanners. On the other end of the spectrum, the Mustek driver and xscanimage are fairly complicated programs and SANE had no problems accommodating them.

## SANE and Commercial Applications/Drivers

What's our position with respect to commercial SANE drivers or applications? In the spirit of the GNU Public License, it is preferrable to have the source for SANE programs available. However, it is permissible to write a dynamically loaded, commercial SANE driver on Linux and other platforms that support dynamic loading. (Drivers are always dynamically loaded, so this doesn't cause any extra work.) By the same token, it is also proper to write a commercial application that links with the libsane.so shared library. The basic ideas supporting this position are:

1. Healthy competition between commercial and free programs is an asset, not a liability.
2. The more wide-spread use SANE finds, the better for the Linux/Unix community.

## Future Plans

In the immediate future, the plan is to add support for many more devices. For example, Agfa and Plustek scanner and Nikon filmscanner drivers are planned, and there is hope that drivers for some of the more popular digital cameras will materialize soon as well. To get the ultimate in network connectivity, there are also plans to implement a scanner application in Java, making it possible to control your scanner from your favorite Java-enabled web browser.

In the long term, it would be interesting to generalize SANE to embrace other multimedia devices including audio sources or video tape recorders.

In other words, SANE has just started, and there are many exciting projects to come. If you're interested in pursuing some of these by all means get in touch with other developers through the SANE mailing list (see Resources).

Resources

Acknowledgements

David recently graduated with a Ph.D. in Computer Science from the University of Arizona and is now a Member of the Technical Staff at HP Research Laboratories in Palo Alto. David first got involved with Linux when writing the Reed-Solomon error-correction code for the floppy tape driver. Then he pretty

much forgot about it until he needed an affordable Alpha workstation. That's when he got involved with the Linux port to the Alpha. Ever since that time, he has been hanging around in the free software community. When not playing with computers, he enjoys spending time with his lovely wife. He can be reached via e-mail at David.Mosberger@acm.org.

Archive Index Issue Table of Contents

Advanced search

# GPIB: Cool, It Works with Linux!

**Timotej Ecimovic**

Issue #47, March 1998

GPIB is a standard bus used in laboratory and industry data acquisition and experimental control that is now available for Linux.

Gambling is a way of life with computers. The technology is changing daily, and anyone involved with system administration should try to keep up with it at any cost. Sometimes it's all too much. You are asked whether a new feature can be introduced into the network. If you say *no*, you might fall behind the competition; if you say *yes*, you're bluffing and you might be shooting yourself in the foot. Linux became my most loved operating system, when I realized that saying *yes* is usually more gambling with a good chance of success, than bluffing with an empty hand. My involvement in GPIB for Linux is such a case.

## GPIB? What does it stand for?

GPIB is a standard bus in data acquisition and industrial process control. It was developed in 1965 by Hewlett-Packard and was first called HPIB. It was subsequently renamed GPIB (general purpose interface bus) or the IEEE488 interface. GPIB hardware is available everywhere. All kinds of AD converters, digital meters, even printers and plotters use it as the bus of choice. Clean up those dusty corners in laboratories, and you may find an ancient piece of hardware with a GPIB interface at the back. As PCs arrived on researchers' desks, GPIB interface boards were developed, enabling control of GPIB hardware chains via a computer.

## How Linux Support Came to Be

I am involved with the Laboratory for Technical Physics in the Faculty for Mechanical Engineering in Ljubljana, capital of Slovenia. The staff uses GPIB for experimental control and data acquisition, and any computing environment introduced into the lab must support GPIB if at all possible. After I converted the laboratory to Linux, an end was put to printing via floppy disks and waiting

in queues for the daily mail on the single computer hooked to the Internet. Standard Linux applications replaced most of the everyday software previously in use. However, GPIB support soon became an issue. I was not really a GPIB freak myself, but I knew about its wide use, and I knew the spirit of Linux from my long-time experience with it. The dice were rolling as I assured everyone, "GPIB for Linux? Yes, of course!" All I had in hand at the time was a short note in the Linux Hardware HOW-TO, directing me to the Linux Lab Project in Germany (see Resources).

### Software on the Web

On one fine web surfing day, I found what I needed: *The Linux GPIB package*, written by Claus Schroter at Freie Universitet, Berlin. The package includes a loadable GPIB driver module, the basic library for accessing the bus functions from C and a Tcl interface, enabling GPIB for the Tcl language.

I had Slackware 3.1, with kernel version 2.0.0 on a 100MHz Pentium board with 16MB RAM. The GPIB interface card available was the CEC PC488. A rather low-end ISA board but good enough for my testing purposes. Informative documentation of the package states that the following boards are supported: National Instruments AT-GPIB, NI PCII and PCIIa and compatible boards, IBM GPIB adapter, HP82355 and HP27109 adapters.

The module and GPIB library compiled out of the box, and after actually reading the README file, Tcl support compiled as well. The driver module can be configured at compile time, at module load time or via library calls. Necessary parameters are the hardware address of the GPIB board, the DMA channel and the IRQ being used.

Before use, the library must be configured as well using the /etc/gpib.conf file by default. A specific configuration can be done for every hardware device attached to the bus. A special identifier name should be provided for each device so that the library can access any hardware device by a reasonable name. I found this method of configuration very convenient. All GPIB devices on the bus have different addresses and their initialization strings vary. With configuration via /etc/gpib.conf file, the necessary parameters must be determined and written into the configuration file only once. Then, all you need to do is remember the arbitrary name you assigned to that device. There is also a Tcl/Tk-based application called **ibconf** which simplifies maintaining the configuration file.

The feature that drew most of my attention was remote GPIB, **rGPIB**. It is a very cool option that enables computers without a GPIB board to access a GPIB board on a remote computer across a TCP/IP network. It is much cheaper than buying interface cards and much simpler than swapping one GPIB board

between computers. Remote GPIB uses RPC (remote procedure calls) for communication between client and server, so the RPC portmapper must be up and running before the rGPIB server can be used.

### What To Do Now

With the package compiled and ready to use, I took a low-frequency spectral analyzer HP3582A with GPIB interface and a HP3312A function generator to feed the analyzer. For readers who aren't familiar with these machines, imagine yourselves counting the flow of traffic on the highway. The highway represents the function generator. You count the vehicles, determine the increase of traffic from the last count and type the results into a portable computer. With these actions, you act as an analyzer—give or take a few Fourier transforms. That's the basic idea.

The computer should bring the analyzer up with initialization strings sent over GPIB and set it into a mode, such that all its functions can be controlled via the bus. With this control established, the computer must ask the analyzer for data, and then listen as the data is transmitted to the bus. As it is not my goal to discuss the philosophy of GPIB in this article, it should be enough to note that the GPIB software interface must provide the means of writing and reading strings to and from the bus, plus some extra functions for status and event-driven operation.

I attached the hardware to the Linux workstation and realized that strings could now be transmitted flawlessly in both directions. My work was done, and other members of the laboratory could now use the new setup for serious experimentation. However, as a Tcl/Tk fan, I couldn't stop at this point—I had to check out the promised Tcl capabilities. A shared library, loadable from Tcl is provided. It adds the new command **gpib** to Tcl interpreter and all the bus functions can be accessed via the new command.

### Somebody Stop Me, Please!

The bluffing worked out one hundred percent, and I held four aces in one hand and a full house in another. Stop? No way! It occurred to me that I could create a user-friendly interface to the HP3582A spectral analyzer using Tcl/Tk power.

I started working on the user interface, then decided to do my own Tcl interface library. Not because anything was wrong with the existing one, I just needed an extra flag to disable actual calls to the GPIB library, because I was doing part of the program development at home without either a GPIB board or network access to a remote GPIB. I added a flag at the Tcl level to enable all of the functions to operate without actually calling the low-level library. With the

manual for the spectral analyzer and Linux as the development platform, I created a neat user interface for remote analyzer operation (see Figure 1).

**Figure 1. User Interface for Remote Analyzer**

### What Was This All About?

When I finally recovered from the programming spasm, I pulled away from the keyboard and took time to reconsider the improvement in the laboratory situation. Previously the laboratory had one ancient Motorola-based HP 300 workstation as the main work horse for experimental control. Programming was tedious because most work was done via device file without any high level library. Our other choice was a PC with MS-DOS, which is a fine machine for experimental work, but to my thinking useless as a good development platform. I feel the new solution using Linux workstations is superior to either of the other choices. You can read daily e-mail, work on a GPIB application and perform non-critical measurement at the same time on a single computer. If you are not scared of writing few lines of C code, hacking a script or two and merging together different development tools, a Linux workstation with GPIB support is a splendid machine for an experimentalist. It was my first involvement in GPIB, but the neatness and freedom of the environment raised my enthusiasm and pushed me beyond my original intentions.

There is, of course, the fact that at the time of writing most of the commercially available software for experimental control is native to Microsoft-based platforms or proprietary Unix workstations. But this is changing with Linux gaining ever more acceptance in products for measurement and control. In large environments, where vendor support is an important issue, commercial packages still prevail. However, for universities and research laboratories with enthusiastic staff and less critical demands, the Linux solution is worth trying. Of course, if you are on a tight budget, you don't have much choice. Linux and the GPIB package are free of charge and usually do not require new hardware. Linux might even save you a bill or two in the future on network-based capabilities. With real-time Linux being introduced to the laboratory now and in the future, there should be no restrictions on the seriousness and the importance of the experiment.

Resources

**Timotej Ecimovic** is progressing towards his graduation in physics at moderate speed. He lives with his girlfriend Manca in Ljubljana, Slovenia and enjoys one very lovable parrot Hannibal, several aquarium fish with bizarre names and an old guitar with the gift of creating very weird sounds. And of course, he is a Linux enthusiast. As he tries to stay clean of the omnipresent and lethal “Oh man! I haven't got time! Got so much work to do!” disease, he can be reached at cic@fiz.uni-lj.si.

Archive Index Issue Table of Contents

Advanced search

# Getting Rid of Spam

**Brandon M. Browning**

Issue #47, March 1998

Here's a way to filter unwanted mail using procmail.

If you regularly receive a high volume of mail (e.g., you subscribe to several mailing lists) or are the target of unsolicited bulk e-mail (UBE), mail filtering may have crossed your mind. **procmail** is a flexible tool that allows you to process incoming e-mail and perform user-definable actions, such as filtering, prioritizing or informing you of new mail. **procmail** is part of a larger toolset that also includes **formail**, a program that can handle tasks such as recognizing duplicate messages, digest bursting, header extraction/addition and the generation of auto-replies.

## Setting It Up

If you do not have procmail on your system already, you can pick up the latest copy at ftp://ftp.informatik.rwth-aachen.de/pub/packages/procmail/procmail.tar.gz. The current version as of this writing is 3.11pre7. (Don't let the “pre” scare you; this version is very stable.) The package compiles out-of-the-box for Linux, but you may want to change a few compilation parameters (like installing into /usr/local instead of /usr). Read the INSTALL file included in the archive to get the complete installation process.

Once you have compiled and installed procmail, you can start using it. Since it is designed to process incoming mail, you need to modify your ~/.forward file (or create one if you do not have one) to include the following line that will automatically invoke procmail:

```
"|IFS=' '&&/usr/local/bin/procmail -f-||exit 75 #
```

Include all quotes and do change *YOUR_LOGIN_NAME*. The reason for doing this is described in the FAQ that is bundled with the archive. Next, you will need to create a *recipe file* that procmail will use to filter your mail.

## procmail Usage

When executed, one of the things procmail does is look for a file called .procmailrc in your home directory. This file holds all of the commands, called recipes, that tell procmail what to do with the incoming mail. A recipe has three parts: the start (**:0**, followed by optional flags and local lockfile), the optional conditions and the action to be taken if the conditions evaluate out to true. Each part of a recipe is on a separate line.

The flags tell procmail which part of the message to look at (headers, body or both) and whether the recipe is a delivering (terminating) or non-delivering recipe. The local lockfile ensures that concurrently running procmail processes do not interfere with each other when writing out to a mailbox and should be used only for delivering recipes. The optional conditions start with an ***** (asterisk). There can be more than one condition or no conditions as you feel necessary. Everything proceeding the ***** is passed to an internal regular expression (regex) engine, which is compatible with **egrep**. All conditions are logically ANDed together. If you choose to have no conditions, procmail defaults to a true result (which is what you would expect). The action line tells procmail what action to take if the all the conditions match. The action line can start with a **!** (to forward the message), a **|** (to pass the message to a program), or a **{** (to start a nested block). Anything else is taken to be a mailbox name.

Here is an example of a recipe that will filter all mail coming from the Whitehouse into a mailbox of the same name:

```
:0:
  * ^From: .*whitehouse\.gov
  whitehouse
```

Let's analyze this one line at a time. The first line tells procmail that we have started a recipe (**:0**) and that we want procmail to determine the local lockfile (the second **:**). The next line is the condition that procmail must match in order for this recipe to be true. By default, procmail scans only the headers, which is what we want. The last line is the action, which tells procmail to write the message to a file called whitehouse.

For contrast, here is a non-delivering recipe:

```
:0 f
  * ^X-Face:
  | formail "I X-Face"
```

This one uses **formail** to strip out an unwanted X-Face header. Notice the lack of a local lockfile. Since this is a filter, a lockfile is unnecessary. **procmail** will still work if you place one there, but it will complain.

## Unsolicited Mail

At the beginning of this article, I mentioned that procmail is useful to filter unwanted mail. UBE (or spam as it is more commonly known) has become an annoying trend and a nuisance. The volume of spam is believed to have increased substantially on Usenet, where people excessively post the same message to various newsgroups. Usenet spam is perceived to be “cancellable”, meaning a posted message can be deleted by the moderator before being read by too many people. To get around this type of cancellation, someone got the idea to send the message to you directly rather than posting it to Usenet where it might be deleted before you read it. Hence, UBE started to infiltrate users' in-boxes. Pioneers of this form of marketing quickly found out that many users disliked spam in any form, and often found their own mailboxes full of flames. They started to obscure headers to make it hard to find out where the message really came from.

Why is spam considered the bane of the Internet at large? Unlike the junk mail you receive in your postal box at home, spammers rarely pay as much to send the spam as the recipient does to receive it. The fact that they pay less than you is called “cost-shifting”. Another form of this shifting is the use of third-party computer resources by the spammer for sending their bulk e-mail without permission. By doing so, the spammer is costing that innocent company both time and money spent to clean up after the spammer. Another tactic widely used is the munging of the headers in such a way that uneducated recipients may waste the time of innocent third parties who had nothing to do with the spam in the first place. This type of deceitfulness can also be considered cost-shifting and has been ruled illegal in the U.S. Courts.

Cost-shifting is not the only argument against spam. There is no single removal point as each spammer generally runs their own list. To that end, they are not required to honor a removal list. As more and more people send spam, you will never be able to remove your address from every single list. Why should you have to if you didn't ask for it in the first place? Finally, a great deal of the spam is or could be considered illegal, such as pyramid schemes, multi-level marketing schemes and lotteries.

## Dealing with UBE

Rather than having all that garbage clog up your in-box and make it unusable for real work, you can now use procmail to filter it out. Earlier I mentioned that spammers try to obscure headers to make it hard to trace. By doing so, they sometimes give inadvertent “signatures” that you can tell procmail to filter on. For instance, a popular bulk e-mailer, the Stealth Mailer, inserts a false Received: line to deter flames. However, both versions generate the wrong time

zone. Armed with this knowledge, you can now filter out a great deal of spam. I have yet to see a false positive on this one.

```
# Filter spam that used the Stealth Mailer Classic
  :0
  * ^Received:.*id GAA.*-0600 \(EST\)$
  spam
```

Another great spam filter looks for a “Comments: Authenticated sender is” header. Unfortunately, filtering on that alone does not do the trick because Pegasus Mail (a popular mail client for the Windows operating system) uses this header legitimately. Fortunately, Pegasus adds an X-Mailer: header in addition to the Comments: header. If both the Comments: and the X-Mailer: exist, then a Pegasus Mail user sent the message (and is probably legitimate); otherwise, it is a bulk mailer. The following recipe will filter this situation. (Note that there is a space and a tab between the square brackets. Unfortunately, procmail does not have a whitespace escape sequence as Perl does.)

```
# Only Pegasus Mail for the WinOS generates a
# valid "Comments: Authenticated sender is ..."
# header. If this is present and the X-Mailer is
# not; then the message in the question is almost
# certainly spam.
:0
* ^Comments:[   ]*Authenticated sender
* !^X-Mailer: Pegasus Mail
spam
```

These two recipes alone filter out a majority of my spam. You can quickly see that a list of these recipes strung together would be beneficial. This is exactly what several free packages have done. My personal choice came down to Alcor's filters (http://alcor.concordia.ca/topics/email/auto/procmail/spam), which I found to be non-intrusive, easy to understand and quite flexible. Alcor's filters work by applying over 1300 filters to the message. If a filter is matched, the message is tagged with a special header. Then, all you have to do is take whatever action (e.g., delete, write sender, etc.) you deem appropriate for these messages with the special headers. I personally avoid “reply” because I dislike using auto-responders, and “delete” because I believe in checking for false positives (of which you will unavoidably get a few).

I recommend downloading all of the tag recipes (use the “save as source” [not text] feature on your browser). I placed the filters in a new directory, cleverly called ~/.procmail. You will most likely need to edit the file **tag-radical** in order to comment out (using a # at the beginning of the line) or change the three uncommented **INCLUDERC** lines. Otherwise, you will see annoying “Couldn't read xxxx” errors in your $LOGFILE each time you process a message. Once that is done, add the following recipe in your ~/.procmailrc at the point you wish to check the incoming message for spam. I check mine at the very top before I do any kind of filtering and have found that this works well.

```
# This enables Alcor's tagging filters
INCDIR=$HOME/.procmail
INCLUDERC=$INCDIR/tag
INCLUDERC=$INCDIR/tag-agis
INCLUDERC=$INCDIR/tag-aol
INCLUDERC=$INCDIR/tag-contents
INCLUDERC=$INCDIR/tag-jdfalk-cyberpromo
INCLUDERC=$INCDIR/tag-jdfalk-llv
INCLUDERC=$INCDIR/tag-jdfalk-nancynet
INCLUDERC=$INCDIR/tag-panix
INCLUDERC=$INCDIR/tag-radical
  :0:
  * $ ^$special_header
  spam
```

That is all. If everything goes well, you should notice that most (if not all) of your spam now goes into your mailbox named spam. You can test it by sending a message to yourself that contains content that these filters will catch (try sending yourself a message with **-- Headers --** somewhere in the body).

### I've Filtered. Now What?

Alcor's tagging system might catch legitimate mail, so I do not recommend deleting anything before you look at it. Once you have verified that it is spam, you have two options: complain or delete. If you want to fight spam, I recommend you to read the SPAM-L FAQ (http://www.ot.com/~dmuth/spam-l/) and possibly join the mailing list. Instructions on how to do so are in the FAQ.

### Conclusions

This article is only the tip of the iceberg on using procmail and its accompanying programs. If you are interested in the continued use of procmail to filter your e-mail, I recommend the procmail mailing list. The regulars there are knowledgable and willing to help. You may also want to search out other procmail solutions. To put these filters through a stress test and to help further develop them, I have subscribed to a special mailing list that sends nothing but spam that is forwarded through it, which takes special care to try and filter duplicates. At the time of this writing, 83% of mail I have received from this list was properly filtered.

Resources

Acknowledgements

**Brandon M. Browning** is a Software Engineer for NorthWestNet, Inc., an ISP located in Bellevue, Washington. When he is not hacking Perl or fighting spam, he can often be found pursuing his other interests: The Tick, Babylon 5, Star Wars and on occasion sleeping. He can be reached by e-mail at brandon@powertie.org.

Archive Index Issue Table of Contents

Advanced search

# qvplay and the Casio QV-10 Camera

**Bob Hepple**

Issue #47, March 1998

Linux software to control the Casio QV-10 camera is now available. Mr. Hepple tells us how to use qvplay.

In the continuing battle with the Evil Empire, I have recently reduced my dependence on “Them” by one program—I discovered a way to use my Casio QV-10 digital camera with Linux.

This article is a simple HOWTO on the method I used—hopefully to encourage more people and Casio to use Linux.

One of these days, every peripheral will ship with Linux drivers and support software—until then, let's “share and enjoy!” (Douglas Adams, *Hitchhiker's Guide to the Galaxy*).

## The Casio QV-10 Camera

Figure 1. Casio Camera

Describing the camera in full is probably best left to Casio but suffice it to say that it:

- has fixed focal length lens (no zoom)
- has fixed resolution of 320x240 pixels in 24-bit colour
- stores up to 96 shots in Flash memory (immune from low battery power)
- takes 4 AA alkaline cells or an optional, external power source, but not NiCad

- has a serial link to a PC
- has a separate printer available, driven directly by the camera
- can also output to TV/video
- has CAM default file format, proprietary to Casio
- sells for about $400 in Singapore (about $263US), although I paid a lot more than that last Christmas—prices have dropped.

The resolution of the pictures is adequate for simple web pages but nowhere near as fine grained as some of the models recently on the market. By comparison, a Kodak PhotoCD image digitised from a 35mm slide or film would be about 3000x2500 pixels—more than 100 times the number of pixels. On the other hand, while the resolution is so-so, the 320x240 jpeg files that are produced are reasonably fast to download with current modems.

It is quite possible that Casio's proprietary CAM format is superior to others for certain purposes (I am no graphics expert), but it is no good for web pages, which need JPEG or GIF files. The Windows software that Casio provides makes converting more than one or two files to JPEG format very difficult. No filter is provided for batch conversion of files to other formats. The process for each conversion is:

1. Open CAM image on PC (10 seconds on my 486)
2. Pull down File|Export menu or press **ctrl-E**
3. Pull down Format|JPG
4. Type in new file name (.jpg extension is provided automatically)
5. Select OK

On one file this is manageable, but on 96—forget it.

### The qvplay Software

**qvplay** was written by ken-ichi HAYASHI and Jun-ichiro “itojun” ITOH to control the QV-10 and is really great. Itojun also wrote a filter called **cam2jpeg** to convert CAM files to JPEG. (I haven't tested this program yet.) With qvplay running on Linux you can:

- Download individual or groups of pictures to or from the camera in JPEG, BMP, PPM, RGB or CAM formats.
- Instruct the camera to take a picture.
- Delete pictures from the camera's memory.
- Instruct the camera to display a certain picture (or 4 or 9 thumbnail pictures).
- Protect or UN-protect specific pictures from deletion.

It seemed to me that the quality of the pictures from qvplay was better than from the QV-LINK software provided with the camera—apparently, some older versions of QV-LINK do some automatic re-touching of the pictures which seems to make things worse. Also included in the package were two utilities:

- **qvrec**: send CAM files to the QV-10 camera.
- **qvalldel**: clear the camera's memory.

### The Software

I used version 0.92 of qvplay. Just use your favorite search engine to find qvplay-0_92_tar.gz and download from the site nearest you.

Configuration couldn't be easier. Expand the distribution file and follow the instructions. A **setup** script is provided that worked just fine "out of the box". I use Red Hat Linux 4.1 with a 2.0.18 kernel. One thing you might want to tinker with is the default serial port for qvplay to use. I changed the supplied default of /dev/cua1 to /dev/cua0. Once it's working, use **strip** on the executables to remove extraneous lines such as debug commands and run the command **make install**.

I have written man pages for the **qvplay**, **qvrec** and **qvalldel** software (not in the original distribution). These pages are available at http://home.pacific.net.sg/~bhepple/qvplay/qvplay.html.

### Cameras Supported by qvplay

According to the source code, qvplay appears to support the following cameras:

- QV-10 in its various flavours, e.g., QV-10a
- QV-11
- QV-30
- QV-100 (including the fine resolution)
- QV-300 (including the fine resolution)

The only camera I have tested is the QV-10.

### Handy Scripts for qvplay

Once I got the hang of qvplay, I wrapped it up in a couple of simple scripts to do the things I normally do without reading the manual pages. These two scripts are:

1. **get_a_pic**: A simple script to get one photo. (See Listing 1.)

2. **get_all_pics**: Another way to get all the photos from the camera. (See Listing 2.)

### Post-processing of JPEG Files

One weird thing that you have to do is fix the size of the JPEG images (see the get_a_pic script). I must confess I don't fully understand what's going on here but apparently the images come across as 480x240 pixels and you must change them to an aspect ratio of 4:3 or 320x240 pixels. You can do this with the **xv** program or using the Independent JPEG Group's commands, **djpeg** and **cjpeg**, along with the Poskanzer portable bitmap utilities. These utilities are normally found in the various Linux distributions. For example:

```
qvplay -g 1 | djpeg | pnmscale -xsize 320\
        -ysize 240 | cjpeg > foobar.jpg
```

### Using a WWW Browser to View Files

You can view your JPEG files quite nicely with xv(1) or with a WWW browser such as Netscape. In the latter case, you might want to generate HTML index files for your shots using something like the following automatic procedure.

Assuming your JPEG files are sitting in a directory—e.g., I keep all the files from one day's shooting together under a directory labelled with the date, something like ~/photos/971128/*.jpg—I then run the following script on them to create an index page viewable by the browser. This could be put into a Makefile:

```
(cat hdr
ls $i*.jpg |sed "s/^/<IMG SRC=\"/" |sed
"s/$/\">/"
cat tlr) > index.html
```

The file hdr simply contains a standard HTML startup:

```
<HTML>
<HEAD>
  <TITLE>Photo viewer</TITLE>
  <META NAME="Author" CONTENT="Bob Hepple">
</HEAD>
<BODY>
<H1>Photo viewer</H1><HR>
```

Similarly, the file tlr contains your standard HTML wrap-up script:

```
<P>
<HR>
<ADDRESS>
<A HREF="mailto:bhepple@pacific.net.sg">Bob
Hepple</A> <P> Copyright © 1997 Bob
 Hepple. All rights reserved.
</ADDRESS>
</BODY>
</HTML>
```

## A Graphical User Interface for qvplay

**qvplaytk** is a Tcl/Tk wrapper for qvplay which provides a GUI interface. Figure 2 is a screen shot of the program which can be found at its author's (Mr. Amano) home page at http://www.bekkoame.or.jp/~tormato/qvplayk.htm.

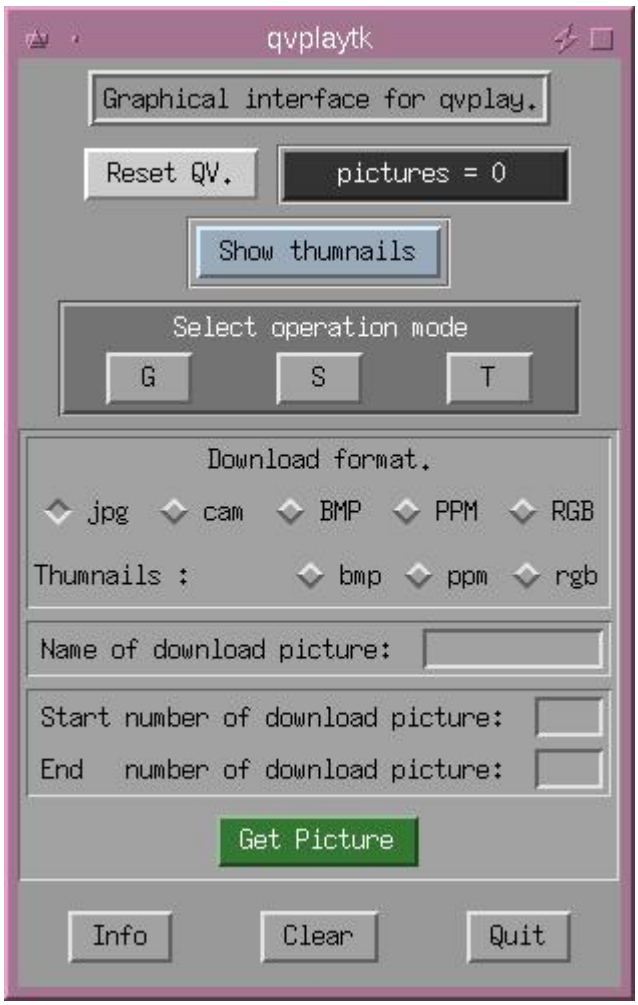


Figure 2. Screen shot of qvplaytk

As a Tcl/Tk script, qvplaytk is very easy to configure and adapt. For example, you might like to change the obscure “G”, “S” and “T” buttons to “Get”, “Save” and “Take”.

One very nice feature of qvplaytk is that the “Take” mode allows you to take a photo every N seconds—this could be used in a remote monitoring application. Perhaps it could be used for one of those strange web sites which offer a changing view of the level of the coffee in the kitchen, or for keeping an eye on your kids in the next room.

Apart from qvplay, qvplaytk requires Tcl 7.4 and Tk 4.0 or Tcl 7.5 and Tk 4.1. It also relies on **xv** for the viewing functions.

## Converting Casio CAM Files to JPEG or PPM

Itojun wrote cam2jpeg (sometimes written as camtojpeg) as a filter to batch convert CAM files to JPEG or PPM formats. It works with a whole range of Casio products that output the CAM format such as the QV-10, 10A, 30 and 100. It can also be found through ken-ichi HAYASHI's home page at http://www.asahi-net.or.jp/~xg2k-hys/.

Figure 3. Singapore River

Figure 4. Lau Pau Sat Food Centre

## Example Pictures

The camera works well in full daylight, as shown in Figure 3, a shot of the Singapore River from the famous “Boat Quay”. It also works well in very low light conditions, as shown in Figure 4, a shot of one of our wonderful food centres, “Lau Pau Sat”, in the business district.

**Bob Hepple** has been hacking at Unix since 1981 under a variety of excuses and has somehow been paid for it at least some of the time. It's allowed him to pursue another interest—living in warm, exotic places including Hong Kong, Australia, Qatar, Saudi Arabia, Lesotho and (presently) Singapore. His initial aversion to the cold was learned in the UK. His ambition is to stop working for the credit card company and tax man and to get a real job. Bob can be reached at bhepple@pacific.net.sg.

Archive Index Issue Table of Contents

Advanced search

# X-Designer

**Timotej Ecimovic**

Issue #47, March 1998

X-Designer generates native code for the OSF/Motif widget set, with sources either in C, C++ or UIL, the Motif user interface language.


- Manufacturer: Imperial Software Technologies (IST)
- E-mail: sales@ist.co.uk
- URL: http://www.ist.co.uk
- Price: $3445 UK Pounds Sterling for commercial single user
- Platform: Virtually any Unix/X platform including Linux
- Reviewer: Timotej Ecimovic

X-Designer is a state-of-the-art, multiple-award-winning GUI builder that runs on most Unix/X workstations. It generates native code for the OSF/Motif widget set, with sources either in C, C++ or UIL, the Motif user interface language. Not just for Motif, it can also produce code which compiles on Microsoft Windows platforms, using the MFC class library for GUI elements. Sounds promising, doesn't it? The very edition I had in my hand read *X-Designer release 4.6: Java Edition*--it can produce Java code as well.

## Installation

Naturally, my excitement was growing as I opened the stylish box and got ready to install this expensive piece of software on my PC. Inside the box, I found a

small manual called “Installation and Release Notes”, a large book of 840 pages with the title “Release 4.6 User Guide”, some advertisements and licensing material, two reference cards and a white CD-ROM containing the software.

The target PC was a 100MHz Pentium with 16MB RAM and 16 bpp XFree3.2 server for the ATI Mach64 graphics adapter with 2MB memory. This was more than enough memory for X-Designer. The Linux residing on the PC disks was a Slackware 3.0 distribution with 2.0.0 kernel. I had no Motif, just freely available **lesstif** libraries version 0.81. After all, X-Designer manuals claimed that Motif is not needed for running the builder and only Motif version 1.2 is needed for the created applications, not the newer 2.0 version.

Installation was smooth and trouble-free. As soon as the CD-ROM is mounted, X-Designer can be run in demo mode if the ISO 9660 Rock Ridge extensions are supported. The installation is done with a single **Install** command, where the only mandatory specifications are the location of the LINUX distribution files (e.g., /cdrom/LINUX) and the destination directory (/usr/local/xdesigner/ in my case). The installation took about 21MB of disk space and was accomplished in a robust and straightforward way. No fancy installation options, but you can rely on its successful completion.

Immediately after installation, the X-Designer still ran only in demo mode, which doesn't enable saving the designs or generating any code. I proceeded to the licensing section of the installation manual. To obtain the license, a special host ID must be calculated with a provided program and e-mailed to IST for the license request. The support staff responded within minutes. Using the very good manual section about setting up the license file and running the provided IST licensing daemon, I had X-Designer up and running under full power less than 30 minutes from the time I started the installation.

### The Basic Functionality

**Figure 1. Main Window**

The user interface is appealing and is designed with programmers in mind, not the users who go “aaaah” at every flashy icon that pops up. Help was there, as it should be with any Motif application. By default, the provided XD/Help browser is launched at help request, but there is an option to use Netscape or any other browser for the same purpose. I prefer the provided XD/Help utility, since Netscape offers the same functionality but eats up system resources mercilessly compared to the small XD/Help and even X-Designer itself. Help is all in text mode, but I believe that after using X-Designer for some time, the need for the on-line help slowly vanishes. There is no Motif programming information provided, and the widgets are described only briefly. A Motif reference book should be handy while seriously using X-Designer. I believe the

help system serves its purpose very well and is designed with the correct assumption that advanced users do not need a lot of colorful diagrams, examples and tips, but just a quick reminder of what is happening.

2641f2.gif

Figure 2. Help Utility

The work area of the main window provides a fair display of the widget instance tree structure. As the widget trees can grow large for larger applications, many options are provided to keep much of the design within visual range. Branches can be collapsed, and the icons for the widgets can be set to a smaller size. The tree can be left-justified and the special annotation bitmaps can be placed beside widgets to visually remind the user of their special properties. The work area has a highly professional design and provides the developer with the needed information about the designed GUI.

The application appearance is built simultaneously as the elements are added to the design tree. X-Designer doesn't simulate the appearance; it actually builds the widget tree from the real widgets. This makes it a sort of WYSIWYG GUI builder, which introduces a dangerous possibility. If the actual widgets created while building the application are connected or configured in a forbidden manner, X-Designer may crash because an unexpected widget condition will appear within the X-Designer executable. Being aware of this problem, I tried to create all kinds of weird designs, but X-Designer is very strict and positively sure it knows more about Motif than I do—it would not allow me to add invalid widget relations to my design.

**Figure 3. Typical Resource Box**

All the standard editing options are present, including support for printing a PostScript image of your widget trees. The resource boxes are available, as one would expect from a GUI builder. A separate resource panel and core resource panel are available for each widget. The panels are aware of the Motif defaults and let you set any resource. The Motif power is not hidden in any way; it is just easier to use.

Callbacks can be manipulated in a similar way. The dialogs for setting callbacks allow you to type in the name of the function and the client data which is to be passed to the callback along with the other parameters. X-Designer does not implement an editor for directly editing the code, but there is an “Edit” button in the callback dialog. It runs your favourite editor as defined by the EDITOR environment variable. As I am an Emacs fan, I use **emacsclient** as my editor, which connects to the running Emacs server for file editing. At first I was

annoyed to find the extra xterm popping up to run emacsclient. However, the User's Guide is very informative, and with its help I quickly found the location of the shell script which X-Designer uses to run the editor. I added the emacsclient, and no more annoying xterm. The configurability of X-Designer is worth all the money you pay for it.

Editing the callbacks actually means editing the stubs file, which is generated during the code-generation process. X-Designer puts a short comment before each callback, which must be left intact if you want to keep your callback code after generating successive stubs files. You can even add some code into the stubs file which has nothing to do with the callbacks, and it will be left there after regeneration of the source files. Just follow the well-documented method by which X-Designer treats those comments.

Besides the usual callbacks, links can be created as well. They are special simple callbacks which can be attached to buttons in order to trigger simple actions like enable/disable, manage/unmanage or show/hide certain widgets. Basically, links are just callbacks which are saved with your design into an .xd file and actually work in the prototype GUI built dynamically by X-Designer. Note that the regular callback code is not saved anywhere other than in the stubs file.

### Basic Code Generation

Code generation is the core operation of every GUI builder. Though X-Designer's friendly interface leads you through the GUI design, code generation is the reason you bought it. I think the Motif/C combination for Unix/X workstations is the native configuration. I tried to create a few small GUIs; they all worked and compiled without warnings. All I had to do was manually set the location of the Motif libraries and headers in the Makefile, since I keep them in the /client/lesstif directory. The same applies for the C++/Motif combination, which also worked flawlessly.

Portability is an important issue today, so I set out to explore the Java and Microsoft Windows code generation options. I had a design targeted for a Motif/C combination, and I tried to change it to Java code. Of course, the callback code was useless; I didn't expect X-Designer to translate my C code into Java code, but links also work in Java. The prerequisite for Java designs is to include the path to a set of Java classes, which are provided with X-Designer, in the **CLASSPATH** variable. The classes are put into the package called uk.co.ist.mwt. They provide components which have counterparts in Motif but are missing in Java. There are also some widgets which are added to the standard Motif set to implement some Java components not present in Motif. They are container widgets, such as the Card Widget, Flow Widget, Border Widget, Grid Widget and GridBag Widget. This enables the WYSIWYG development of Java applications.

Changing my test design for Java required toggling the Java compliance toggle-button under the Module menu. X-Designer then checks the current design, giving you messages about any non-compliant parts. Sometimes the “Fix” button will be enabled, allowing you to trigger the automatic fix for Java Compliance; otherwise, you have to do it yourself. If you know about Java and Motif, the messages generated by X-Designer are more than enough to give you a clue about what is happening. I don't know much about Java, but I know enough to make the design Java compliant within minutes. The Java code generation dialog is different than the C/Motif one in that you must specify which file to create. A file for every class, the top-most application class file and separate sources for string, color, font and other objects are included. Using JDK 1.0.2, I compiled and ran the application without any problems as a Java application. I could have built a Java applet, in which case additional constraints might have popped up.

All of X-Designer's dialogs use the same convention of marking the resources and settings which are relevant for Java. Whenever there is a steaming coffee cup icon next to the resource, it means that it will be honoured in Java code. A very good feature, since at the time you make a change you can note immediately whether or not the change code remains Java compliant.

Using X-Designer for building Microsoft Windows applications is another prime feature. X-Designer must be started in Windows mode to enable this cross-platform development. It can be done via resource, or via running it by typing **xdesigner<\!s>-windows**. In this case, an additional button is added to the tool bar, to toggle the Windows compliance. The procedure is similar to the one used for Java. If you read an already created Motif design into Windows-aware X-Designer, it prompts you with a dialog containing all the warnings and reasons for Windows non-compliant design. Sometimes it can fix the problems, other times it cannot.

Code for Windows is always generated as C++, in three flavours: Motif, MFC or Motif XP. Motif XP is a set of provided classes similar to Motif, but named to match the Microsoft Foundation Classes. Since Motif is much more flexible and allows more freedom than MFC, the basic task X-Designer must perform is to apply additional constraints on the design to force Windows compliance. This is the only method of MFC code generation. In all the dialogs, a convention is used for the MFC-honoured resources; if the entry field or button is painted pink, it means that the resource is honoured only by Motif and not by MFC.

X-Designer contains very advanced tools for C++ class structure and hierarchy maintenance. A subset of a design can be created as a C++ class, and access control can be applied for any widget member. One can use methods such as callbacks and even create custom preludes into the source code to add

additional class members. An interesting feature is definition creation, which can be put on the widget palette as a reusable group of widgets. C++ programmers will find the abundance of features extremely useful, as the class structure becomes the property of the design and can be maintained in cross-platform applications.

All the code-generation options are well beyond the scope of this review, but I am convinced that even though Motif is the native toolkit of X-Designer, Java and MFC support are advanced, and most importantly, they work. For any design information that can be ported to Java or MFC, X-Designer keeps track of it and properly implements it. Every issue is considered, including font inconsistencies, color incompatibilities, different native pixmap formats and problems with long filenames under Windows platforms.

### Advanced Features

X-Designer hosts many features which deserve a review of their own but will receive only a brief description here.

**Figure 4. AppGuru Tool**

**AppGuru** is the interface to previously designed templates for GUIs. Initially, it contains the default template which enables you to quickly create applications with standard components, such as a menu-bar, a tool bar and file selection dialogues. At first the AppGuru looks like a cute toy, but its power lies in its configurability. Each design can be changed into a template. Using X-Designer resources, the new template can be incorporated into the AppGuru menu with control over the separate components, which may or may not be added into the new application. The freedom with resources is great, so the X-Designer administrator in a large software company can add pixmaps which represent components. Users can easily select ones which are really needed. I think this is a great tool for a software engineering environment, where several pre-designed interfaces can be set up according to internal policy. Company information dialogues and standard application parts can be automatically inserted into new designs.

User-defined widgets allow the GUIs created with X-Designer to look beyond Motif and use widgets provided by a third party or even custom widgets. When I first saw X-Designer, I noticed the special widget for OpenGL as supplied by the MesaGL package in the palette of X-Designer widget icons. I was curious how it got there since it is not a Motif widget. The basic idea behind this very complex concept lies in the interface between X-Designer and the widgets used in the GUI building process.

Any widgets can be chosen for incorporation into X-Designer. You must run the provided **xdconfig** utility to create the configuration files which tell X-Designer about resources, constraints and children all new widgets can have. Even icons can be provided in pixmap form for addition to the X-Designer widget palette. Specifying the code generation options for every new widget is a mandatory task. After all this is done, rebuild X-Designer using all the configuration files and the provided xdesigner.o object file. The new binary will contain X-Designer with the newly added widgets.

Installation provides many configurations for common widget sets, and I tested the Athena widget set which worked fine. I can now build GUIs with the Athena widget set, forgetting Motif. The only drawback to this process is the fact that new widgets are compiled into X-Designer itself. If for some reason, the widgets cause segmentation faults or otherwise behave badly, X-Designer may crash. I actually *managed* to crash it by laying out Athena widgets in an invalid hierarchy. Creating successful widget configurations requires deep knowledge of the widgets, C, **make** and X-Designer.

With X-Designer, an on-line help subsystem can be created for the new applications. All help is written in HTML files with anchors as documented in the User's guide. You must only decide whether your application will use XD/Help, Netscape or FrameMaker for the help browser—just link an additional provided object file with your application.

The **XD Replay/Capture** tool is another amazing tool that comes with X-Designer. With Capture, you select the Motif application and capture its widget instance tree. Only the executable of the application is needed; Capture takes a snapshot of its widget structure which you can then drag into X-Designer and reuse. You can then take your old Motif applications, capture them and develop them further with X-Designer. As I understand, the tool works by inserting a sort of parasite shared object into the application which records the widget tree creation. The application must be dynamically linked with Xt for Capture to work. A similar tool is XD Record/Replay; it records any events within the application in an internal script language. These can later be replayed for the purpose of demonstrations, tutorials, testing and other cases where automatic behaviour of an application is desirable.

Figure 5. Record your application behaviour!

Among simpler design tools are the **XmString** editor, pixmap editor, font and color selection tools and layout editor. With the XmString editor, arbitrary XmStrings can be generated on a WYSIWYG basis, a fact any Motif programmer will appreciate. The layout editor is a cool tool to define the layout of child widgets within the container widget. As this is a tedious business, the layout

editor provides great help with setting up those widgets in a functional and resize-friendly way. Using the pixmap editor, which is functional and serves its purpose well, caused an interesting event. The first time I selected the Pixmap editor entry in the menu bar, X-Designer crashed. Strange behaviour, I thought, but quickly decided to use this crash for testing the IST user support. I sent e-mail to support@ist.co.uk describing my problem, and within less than 30 minutes I got a correct answer to the faulty behaviour. The Pixmap editor clashed with the default .fvwmrc setting for a cryptic Meta 3rd-Mouse-button combination which is locked for FVWM private use. I don't use it, so I removed the line from .fvwmrc, and now the pixmap editor works fine. Well done, IST support team. I can understand a glitch or two in any product, and that's why good user support is necessary.

**Figure 6. Laying Out Widgets**

## Conclusion

X-Designer is a mature tool, and its authors have learned all the tricks of the GUI-builder business. It won two Advanced System Magazine Best Product of the Year awards (1993 and 1995) and was the X Advisor Best Product in 1995. Its features make it the most advanced GUI building tool available, to my knowledge. The basic functionality is accessible via a very straightforward and efficient user interface, which never bothers you with unnecessary dialogues, questions and myriad different windows.

X-Designer is highly configurable on multiple levels, so it can be suited to an individual company's internal demand and further tailored to the needs of each user or department subdivision. You can even change the default widget icons into more colorful ones or disable the use of certain Motif widgets, if you wish.

X-Designer is a tool for tough GUI professionals who cannot let the GUI builder dictate their shipping deadlines. For home users, it is a tool from another world, which makes me sigh sadly at the thought of the approaching date when my reviewer's license expires.

**Timotej Ecimovic** lives in Ljubljana, Slovenia and recently earned a degree in physics. He is in the process of deciding what to do after graduation, so if you

have an idea, let him know. He can be reached at cic@fiz.uni-lj.si and always loves getting e-mail from unknown people.

Archive Index Issue Table of Contents

Advanced search

# Accelerated X Laptop Display Server v4.1

**Michael Scott Shappe**

Issue #47, March 1998

The Accelerated X servers are traditionally excellent products on the desktop—I've been using them on desktop BSD/OS and Linux boxes for a long time—providing both adequate stability and excellent speed relative to the freeware efforts.

- Manufacturer: Xi Graphics
- Phone: +1 303 298 7478
- Email: info@xig.com
- URL: http://www.xig.com/
- Price: $199 US
- Reviewer: Michael Scott Shappe

Those of you who read my review of the Fujitsu 420D laptop know that one of my few complaints with it was that they chose to use the NeoMagic video chip—a chip which XFree86 cannot support because NeoMagic has chosen to keep their programming specifications proprietary. As a result, I have been trapped in sixteen-color VGA limbo for quite some time. This has not been a major hardship, since my primary application is word processing (see my review of WordPerfect 7 for Linux next month), but it's annoying, nonetheless.

The only X server available for Linux at the moment that does support the NeoMagic chipset (among many other chipsets) is Xi Graphics' Accelerated X Laptop Server. The Accelerated X servers are traditionally excellent products on the desktop—I've been using them on desktop BSD/OS and Linux boxes for a long time—providing both adequate stability and excellent speed relative to the freeware efforts. (None of which means that Xfree86 is a slouch either in speed or stability; as free products go, XF86 is excellent. It's just that Accelerated-X is usually somewhat better.)

The laptop edition of the latest version, 4.1, appears to be no exception to this track record. Unfortunately, it appears to carry over a few of the warts of past versions as well and introduces one new one which I find extremely annoying.

## Good News First

The good news is that Accelerated X delivers what it advertises: a faster X server, loaded with the most recent extension standards, with a very wide range of supported hardware. Its modular architecture means that the server software will not become obsolete with innovations in display technology—all it'll need is a new module. New and updated display card modules will be available for free from Xi Graphics' web server. Several other features, including the X server extensions (such as PEX) and the font-rendering engines, are also modularized, meaning that these can also be updated or extended without having to upgrade the server.

Particularly pleasing to me, of course, is that the server supports the NeoMagic 2093, providing 24-bit color at 640x480 resolution, 16-bit color at 800x600 and 256 color at 1024x768 on an external monitor. According to the documentation, it also supports a hot key to switch in real time between the LCD panel and an external monitor. Chips with more video memory are supported with more colors and higher resolutions with the maximum resolution theoretically possible being 1900x1440, although no currently available video chipset for a laptop has that much memory.

Presumably, it would provide 1024x768 on a larger LCD panel, if available; as it is, 800x600 is the maximum resolution supported on my laptop's panel.

Installation is a straightforward affair—become a superuser, mount the CD, change directories to the mount point, and type **./Install**. If you have an older version of Accelerated X installed, the installer will move it out of the way before proceeding. You get a choice of which packages you wish to install, and then the installer just does its work.

The Xsetup program has two modes. The text-based version should be familiar to anyone who has ever installed any version of Accelerated X, and there is now a fully graphical version that, if possible, starts up a minimally-configured X server at 640x480 and allows point-and-click configuration. The first time I ran Xsetup, I got the text version by default; after that, I always got the graphical version unless I asked for the text version. The graphical version has a few more options than the text version, leading me to believe Xi Graphics feels that the text version is to be used to get the basics set up and after that you should be far enough along to use the graphical version.

The improvement in speed over XFree86's VGA16 server was immediately noticeable, even with the greater color-depth (16-bit instead of 4-bit). Window manager menu performance was snappier, as was the minimizing and maximizing of windows. WordPerfect, which previously had shown some lag when typing (a lag which I had already suspected to be a combination of both a non-optimized server and a slower font-server) behaved much better, and scaled fonts seemed sharper.

### Now the Bad News

Before I start on the bad news, I want to make a point. There's probably going to be more complaining here than there was complimenting above. Don't confuse quantity and quality. This product does deliver on its main promise: a better, faster, stronger X server that isn't free but won't bankrupt you.

For all of that, there are a number of problems, ranging from mere warts to outright bugs, that I, for one, do not expect in a product as mature as this one. Many of these complaints seem to boil down to Xi Graphics not completely thinking through the “user experience” on a laptop.

Installation is, as I said, fairly straightforward. Unfortunately, as far as I can tell from the documentation, if you have any brand of X Server installed other than an older version of Accelerated X, it will install over it, rather than moving it out of the way. This isn't entirely a bad thing, particularly if you've installed lots of other packages, but it could lead to complications. Newer versions of Xfree86 ship shared libraries with version 6.3 to match the X release (X11R6.3). Accelerated X, while implementing X11R6.3, leaves the shared library revision at 6.1. With both in place, the higher numbers will be used by the dynamic loader, and the mismatch between libraries and server could lead to strange results. This problem is documented, as is the workaround.

The other installation and set-up problem is that the graphical Xsetup frequently crashes after you've asked it to test out a configuration, leaving you to start the whole thing over again. Worse, it won't let you save after certain changes without testing it first. Since there are other ways to configure things, and since it doesn't happen every time, this is not a show-stopper, but it's hard to excuse.

Documentation, generally, is a problem. The Accelerated X Laptop Server is billed as a separate product, sold separately from the Desktop Server with an entirely separate license. Yet, at least my review copy shipped with a generic manual that answers few laptop-specific questions. In particular, neither the manual nor the Xsetup program makes any effort to point out that it doesn't much matter which monitor you select unless your laptop has an external

monitor feed and you're planning to use it. There is no separate menu item for “LCD Panel”, nor any mention of even the possibility.

On a related note, Xi Graphics does not seem to have taken into account the keyboard provided with most laptops. The standard hot keys for selecting among several possible resolutions (**ctrl-alt-+** [+ on the numeric pad] and **ctrl-alt--** [- on the numeric pad]) are almost impossible on most laptops, and there's no way to remap those functions to other keys.

Then there is the price disparity. The Laptop Server is, as far as I can tell, the exact same program as the desktop server, shipped with different video drivers. That's fine—as a programmer, I'm all for code reuse and modular architecture. But the laptop server is nearly twice the price of the desktop edition and comes with not just different but fewer drivers.

Finally, there is a major wart in the released version, which may not be a bug in the X Server but should at least be documented, that requires more than common knowledge to work around. On some laptops (mine included), if you start up in any “text” screen mode (such as the default 80x25) and then start an X Server in a resolution higher than 640x480, the resulting display area is offset by an inch, leaving a blank space to the left of the display and leaving the rightmost inch of the display unavailable (this is true of XFree86 as well). There is now a patch that solves part of the problem, but I don't think the product should have been released with this very visible problem.

The workaround for this problem requires quite a bit of tweaking of configuration files usually left untouched. The details are arguably beyond the scope of a review, but may prove useful to people who have bumped into the problem (both with this product and with XFree86) and became frustrated. They appear in the sidebar (Some Hints) and also on my “Linux on Lifebook” web page at http://www.publiccom.com/web/mikey/lifebooklinux.html.

That said, I discovered an updated NeoMagic driver on the Xi Graphics FTP site (ftp://ftp.xig.com/) which, when applied, solves part, but not all of the problem. With the patch applied, I can now start up in text mode and then start the X server and everything works perfectly. However, virtual consoles still don't work right—switching results in a blank screen—and the text mode is not restored after the X server terminates.

### The Bottom Line

I must admit, I'm very happy to run my laptop with more than just 16 colors under Linux, and I'm very pleased with the speed of the Accelerated X server. But I am very disappointed with the problems that still linger in the product. At version 4.1 and a price tag of $200US, I expected a highly polished product with

no bugs or glitches in basic functionality. It is particularly frustrating to see the product hitting so close and yet missing the mark.

If Xi Graphics can iron out these problems and perhaps bring the price down to a level with its Desktop Server product, I will have no difficulty wholeheartedly endorsing this product. Even as it is, I do recommend it as an adequate solution for those who find they need more capability than Xfree86 can give them on their hardware.

Some Hints

**Michael Scott Shappe** is a senior programmer for AetherWorks Corporation, a technology startup in St. Paul, MN. He's spent the last 10 years hammering on Unix systems of various stripes and addicting the unsuspecting to the Internet. You can peruse his personal web page at http://www.publiccom.com/web/mikey/ or send him e-mail at mshapp19@idt.net.

Archive Index Issue Table of Contents

Advanced search

# SGML CD: A Complete SGML Toolkit

**Terry Dawson**

Issue #47, March 1998

SGML CD introduces a variety of readily available free software packages that assist authors in editing, validating and processing SGML.


- Author: Bob DuCharme
- Publisher: Prentice Hall
- URL: http://www.prenhall.com/
- Price: $49.95 US
- ISBN: 0-13-475-740-9
- Reviewer: Terry Dawson

*SGML CD* is not a text from which to learn SGML. *SGML CD* introduces a variety of readily available free software packages that assist authors in editing, validating and processing SGML.

Bob DuCharme evidently has a great deal of experience in SGML and his writing reflects this. His writing style is the simple and relaxed style of someone who understands the material covered and wants you to understand it too.

*SGML CD* includes a CD-ROM containing versions of each of the software packages described in the text. The software versions supplied are for Microsoft Windows 3.1, 95 and NT operating systems. Unix versions of nearly all of the packages are available, and most of these are already supplied in modern Linux distributions. I'd have been pleased if the source code for the applications had been provided on the CD-ROM, but instead it provides references describing the location of the software on the Internet. You will have no difficulty locating the code you need.

The text describes one tool per chapter. The main topics covered are:

- Editing SGML documents with the Emacs text editor
- Parsing and validating SGML Documents with **nsgmls**
- Formatting Documents with DSSSL (Document Style Semantics and Specification Language) specifications and **jade**
- Analysing Documents with the perlSGML tools
- Developing SGML Applications with Perl
- Developing Windows SGML Applications with SGMLC-Lite

Some 100 pages of the text cover Emacs. The chapter begins with a brief tutorial on Emacs use, but most of the chapter is dedicated to the Emacs PSGML mode, which is a toolkit of Emacs macros to assist SGML authors. I've never been particularly fond of Emacs, but reading this chapter very nearly convinced me to take another look. The PSGML mode provides an edit-time facility capable of detecting and correcting certain types of common SGML formatting errors, such as incorrect or missing tags, and assists with document structuring by providing automatic context-based indentation and SGML syntax highlights. Information is provided on everything from installation through the use of the PSGML mode.

A chapter each is dedicated to two of James Clark's programs—nsgmls and jade. **nsgmls** is an SGML validator; that is, it checks a marked-up SGML document for syntax and structural errors. **jade** is an implementation of the International Standards Organisation's DSSSL. DSSSL allows you to describe how you want an SGML document to appear in a format-independent way. **jade** is an SGML formatter; it takes an SGML document and a DSSSL specification and produces a formatted output file. Current versions of jade are able to produce TeX or Microsoft RTF output formats. These chapters are the two that I found most interesting. Both nsgmls and jade are covered in some detail, but you'll want a separate DSSSL reference if you want to exploit jade seriously.

The perlSGML package developed by Earl Hood is described in some detail. This collection of tools written in Perl assists the SGML author by providing ways of analysing the Document Type Definition (DTD) file that defines the structure of

an SGML document. For example, **dtdtree** and **dtdhtml** analyse a DTD and provide output that describes the relationship of the various structural elements that make up the document. These can be very useful when learning a new DTD.

The remaining chapters provide an overview of two packages that may be used to develop SGML applications. The first package is two versions of the **sgmls.pl** program for Perl, and the second is the SGMLC-lite application for Microsoft Windows. The text describes their use and provides hints that will certainly be of value if you are using either of them.

*SGML CD* does a good job of introducing the reader to some powerful, useful and free SGML processing tools. Those of you with an interest in writing conformant HTML should use the tools described in this book with the appropriate DTD to validate your work. If you are intending to use SGML, then I recommend this book. If you're intending to learn SGML, get yourself a good text describing SGML and let *SGML CD* be your SGML tool reference.

**Terry Dawson** is a long-term LDP author, who has recently realised he should have learned about SGML a long time ago. He can be reached at terry@perf.no.itg.telecom.com.au.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL



# ISDN and Linux—Surfing at Warp Speed

**Mark Buckaway**

Issue #47, March 1998

This article presents a detailed tutorial on setting up an ISDN link to the Internet with Linux.

A typical question on comp.os.linux.* is “Do I need to get an ISDN driver for Linux to make my Bitsurfr/ISDN router work?” There are a lot of misconceptions about the requirements to get connected using Linux and ISDN. This article defines ISDN and explains some of the terms (see Glossary) involved, as well as describing three different ISDN device types and how to get Linux to use them to get on the Net. There is no time like the present to graduate to a 64K or 128K ISDN Internet connection.

## What is ISDN and How Does it Work?

ISDN, or Integrated Services Digital Network, is telecommunications technology that provides the end user with a method for connecting to the phone network using digital signals. It was originally designed to replace the POTS system; however, it never caught on (in North America at least) for standard phone use because there are different implementations of the ISDN standard. However, with the advent of the Internet, ISDN is starting to take off as a solution for the average user to get high-speed Internet access. There are other digital connections available such as cable modems and ADSL; however, ISDN is here now and available almost everywhere.

ISDN is delivered as BRI service with two 64K B (data) channels and one 16K D (signaling) channel. Some telcos still only provide one B channel, however, this is becoming rare. BRI lines are typically used by small business and individuals.

For the corporate user with a PBX or an access router, a PRI would be used. A PRI consists, in North America of 23 64K B channels and one 64K D, and in Europe, 30 64K D channels and one 64K D channel. This is the ISDN equivalent to a T1 or E1.

In North America, telcos deliver the ISDN BRI line as a U interface. This is a two-wire link that easily works over the standard pair of copper wires used by analog phone lines. In Europe, telcos typically deliver the line as an S/T interface. This is a four-wire link that is able to connect seven ISDN devices to the same line. The U interface requires an NT1 adapter to switch the S/T interface. Where the S/T interface is delivered, the NT1 resides at the telco's switch. Most ISDN products destined for North America also incorporate the NT1, thereby saving money and desk space.

The big benefit of ISDN is the small call setup time; that is, the time elapsed between when the phone number is dialed and when the connection is completed. This is made possible by the use of out-of-band signaling via the D channel. The D channel signals the telco's switch to make a call and, in less than one second, the connection is made. The POTS service uses in-band signaling. When a modem dials a system, it picks up the line and sends tones or pulses in the same band as the data travels.

### Getting Your ISDN Line

ISDN terminal adapter user's guides give you the impression that getting ISDN is difficult. If you ask the right questions, obtaining your ISDN line can be easy. When ordering the line, ask:

- Does the BRI have two B channels? (almost always yes.)
- What switch type should I tell my TA to use? (In Canada, usually NI-1.)
- Is it delivered as a U or S interface? Do I need an NT1? (If you have a line delivered as a U interface you must ensure your terminal adapter has an NT1 built in, otherwise, a separate NT1 is required.)

Check the pricing of ISDN in your area. In some places in the US, Frame Relay connections are cheaper than ISDN. ISDN is typically charged on a per-minute basis as well. For example, Bell Canada's Z@P service charges $1/hour per B channel to a maximum of $50/month per B channel between certain hours.

### Selecting a Terminal Adapter

This is the expensive part of getting ISDN. Typically, a terminal adapter ranges in price from $300 to several thousand dollars depending on the type, features and functionality.

There are three basic types of terminal adapters: ISDN routers, external ISDN modems and internal ISDN network adapters. Most terminal adapters today provide POTS jacks for connecting a standard analog phone device. These jacks give you a place to connect a fax machine or an analog modem.

The type of terminal adapter you buy generally depends on the amount of money you wish to spend. The best device to get depends on your application and budget. An ISDN router is generally easiest to configure and set up, but tend to be a bit more pricey. ISDN modems and network adapters require a bit more work as they require additional software, but tend to be more in the price range of an individual. However, with an ISDN router, it's merely a matter of connecting it to your Ethernet card and you are on-line. ISDN modems and network adapters tend to take a bit more work to get online.

### Getting an ISDN Account with Your ISP

ISDN dial-up accounts tend to be more expensive than a 28.8K connection. This is changing as more and more ISPs convert to access routers supporting both ISDN and 28.8K connections off the same ISDN line. UUNET Technologies (in the U.S.) and UUNET Canada offer ISDN access using this type of technology.

When you call an ISP to get an ISDN account, you need to learn a bit about their service. Ask them what ISDN router they are using. This information can be helpful in tracking down problems should they occur. Not every ISDN router will talk to all other ISDN routers. The Ascend MAX is popular among ISPs. If they permit a 128K 2B account, make sure they support MPPP (Multilink PPP).

Also consider the type of service you need. Most ISPs offer either dedicated or dial-up ISDN. Dedicated services give you a line on the ISP's router for your use only. They typically also allocate you a block of IP numbers with the service. You need a fixed IP number if you intend to pick up SMTP mail over the link. Dial-up situations tend to use dynamic IP assignments, which means you are assigned a different IP number on each call. A few ISPs assign you a single IP number for your use. While this is rare, it is a method allowing pickup of SMTP mail over a dial-up account without using UUCP. How you intend to access the net and for what purpose will determine the type of account you need.

Network Address Translation, or IP Masquarade as it is called by the Linux kernel, is required by a router to provide access to multiple systems to a single dial-up ISDN account. In the case of an ISDN modem or an ISDN Network Adapter, the Linux kernel deals with the translation. If you are assigned a class C block or a point-to-point number by your ISP, address translation is not required.

### To Compress or Not to Compress

The most recent ISDN devices feature some form of compression. Their marketing departments will say that compression will lead to increased throughput on the order of two to four times. They claim a 128K line will seem to run at 256K. Don't be fooled. In real live situations, you will be lucky to

perceive an increase of 10%. Consider most information the device will be downloading is already compressed, and therefore, cannot be compressed further (or at least very little). JPEG and GIF files from the web are already compressed, and most ftp archives are either zipped or gzipped, tar files. Only text benefits greatly from compression, and text is usually only about 20% of the data on a web page.

Thus, when shopping for an ISDN device, don't place compression high on your priority list.

### Using an ISDN Router

An ISDN router is the easy way to get connected—enter in a few bits of information into the router's configuration, and you are on the Net. A router is the best method to get a LAN talking to the Internet, as this is a dedicated computer designed to handle the load of passing packets. The only drawback can be the price. For most individual users, an ISDN router will be beyond the reach of their pocketbook. For business, a router is essential for reliability.

To get started, you need to obtain a network card and install it; refer to the Ethernet Mini-HOWTO for details.

The best Ethernet card to use is one with a memory buffer, as it gives the best performance. Most routers come with a twisted 10baseT cable to connect your router directly to your Ethernet card. If you intend to connect more than one computer, an Ethernet hub is required. Avoid 10base2 (coax) cabling, if at all possible, to avoid hassles. It may be cheaper initially, but it can lead to diagnostic and cabling nightmares.

Two ISDN routers I have worked with and found reliable are the Ascend Pipeline 50 and the Farallon Netopia.

The Ascend Pipeline series is well suited for dedicated ISDN applications, as it was originally designed for routing a netblock or a static IP number. The Ascent router supports address translation; something Ascend calls NAT (Network Address Translation). If POTS jacks are required, then consider the Pipeline 75 which is basically the Pipeline 50 with POTS jacks.

The Farallon Netopia is, by far, one of the easiest routers to configure. The Netopia supports dedicated connections and supports Farallon's version of address translation, called SmartIP, for use with dial-up applications. The Netopia comes in a 12-user version as well as a full routing version. Two POTS jacks are available as an option. What makes this router unique is its PC Card (PCMCIA) slot. Insert a PC Card modem, and the router can be dialed into even when the ISDN connection is down. This is very handy for those users who

require remote configuration and troubleshooting abilities. The Netopia also includes a scheduler, so that connections can be scheduled for variable times during the week. No other ISDN router I've used has this feature.

The Ascend Pipeline series is the most popular, and most ISPs are either currently using them or have worked with them in the past. The Netopia is newer, and its feature set makes it worth noting. Of the many routers I have worked with, the Ascend and Farallon products have most impressed me.

### External ISDN Modems

A fair amount of confusion lies in this area. A typical question on comp.os.linux.networking is “Do I have to include ISDN support to operate my Bitsurfr ISDN modem?” Nope. An ISDN modem is a serial device that connects to the serial port on your PC, “dials” a phone number and translates ISDN data to RS-232 data. The system sees it as a regular modem. You can set up **getty** to answer incoming calls, and you can dial out with **minicom** or **pppd**. ISDN kernel support is neither required nor used. Note that the term “modem” does not really apply to an ISDN terminal adapter, because “modem” implies an analog device which an ISDN TA is not.

The Motorola Bitsurfr Pro is one such ISDN modem. I will discuss its connection to a Linux system in detail, because it has been around for quite some time and works well. It features a 115Kbps serial port and two POTS jacks. Newer ISDN modems may include a 230Kbps or higher serial port and compression (such as the Farallon Netmodem).

The test system for the Bitsurfr Pro was a Pentium 100 with 32MB of memory running Linux 2.0.28. The connection was provided by UUNET Canada and both a dedicated connection as well as a dial-up connection were tested. Both situations worked well with one or two B channels operating. The following procedure outlines setting up Linux to talk to the Bitsurfr Pro over the serial port and configuring pppd to use it.

1. Connect the Bitsurfr Pro to the system and the ISDN line.
2. Configure the serial port with **setserial** as required (e.g., **setserial /dev/cua1 spd_vhi hup_notify**). See the Serial HOW-TO for details.
3. Now the fun part: configuring the Bitsurfr Pro. The Bitsurfr Pro requires a full VT100-compliant terminal emulator to use its local menu. During my testing, I have found only Seyon to work—Minicom did not work for me—run Seyon (i.e., **seyon - modems/dev/cua1**). Type **AT** to get the Bitsurfr's attention. Then type **AT@MENU**. The local menu will appear as shown in Figure 1.

4. From the NET SWITCH menu, set up the switch type and the SPIDs (i.e., the ISDN phone line numbers). The directory numbers are required only if you intend to dial into the Bitsurfr. The Data SPID and Voice SPID2 should be made the same. Selecting “Port To Configure” selects which SPID to set up. Select “Reset Network Link” after the SPIDs are entered to start the Bitsurfr talking to the ISDN switch. If all goes well, the LS (link status) light will turn solid green. This can take as long as two minutes to occur.
5. From the CALL SETUP menu, set the B Channel Speed to 64K unless ISDN in your area does not support 64K. Basically, if calls cannot be made, reduce this setting to 56K.
6. From the PROTOCOLS menu, set “Rate Adaption Protocol” to PPPC.
7. From the LOAD/SAVE menu, select “Save Total Active Profile” and save it to Profile 0 and Profile 1.
8. Install pppd.
9. **pppd** comes with a sample script in the scripts directory of its source tree. Copy **ppp-off** to the the /etc/ppp directory; ppp-off is used unchanged.
10. Refer to Listing 1. Create the scripts as listed in the /etc/ppp directory. These are modified versions of the pppd ppp-on and ppp-on-dialer scripts. The ppp-on script runs pppd and the ppp-on-dialer dials the phone number. The ppp-on-dialer also controls the number of B channels the Bitsurfr uses. Most dial-up ISPs only permit one B channel usage. Check with your ISP for details. For two channels, the Bitsurfr must be told to use MPPP instead of PPP and must be given two phone numbers with the ATD command.
11. Create an options file as in Listing 2. The debug option is useful for initial testing. Watch your syslog for messages from pppd. The name option is required to identify who you are in the chap-secrets file.
12. Create the chap-secrets and pap-secrets files. The contents should be the same in most cases. Listing 3 shows an example. The chap and pap-secrets should have the same contents and should list every router on the remote end. In the example, most routers on UUNET's Alterdial POP in Toronto (Canada) are listed. To determine the remote router names, when connecting, monitor the syslog. The names are reported as part of the negotiations. This is an easy way to build a router list. Alternatively, call your ISP and ask for this information. ISDN routers do not issue normal login prompts and rely on chap or pap for authentication. Even if a login prompt were issued, it would not appear through the Bitsurfr's PPP support. All routers are listed twice in reverse order in case bi-directional CHAP or PAP is requested. (Cisco routers tend to default to two-way CHAP.) If you have a dedicated connection, you need list only the one router you are dialing in to; otherwise, you have to list all the routers you may hit in your dial-up connection.

13. Go on-line by running the ppp-on script. If all goes well, you'll get connected. The DTE (data terminal equipment) light will flash while the Bitsurfr is dialing and glow solid green when the ISDN portion of the link is active. The total time from dialing to authentication is about 2 to 8 seconds depending on the load on the remote routers.
14. Surf's up! Check the routing table to make sure a default route exists (i.e., run **netstat -rn**). If it does, **ping** something on the Internet. Your packets should be returned.

For a dial-up connection, you might want to go one step further and install **diald**, the Linux dialer daemon, to make your Internet connection automatic when Internet traffic is sensed.

There are a few problems that might occur in getting connected with Bitsurfr Pro. Incorrect SPIDs is the most common problem. Check with your telco for the correct format of the SPIDs. Secondly, a typo in the scripts will cause various weirdness. For example, an error in the phone number in the ppp-on-dialer can cause certain grief. Lastly, if the pap-secrets file is not set up correctly, it will cause authentication problems and failed connects. This error type will be noted in the syslog output from pppd when the debug option is enabled. Obviously, a lot of detail is missing from the above instructions; however, after some experimentation, things should start working. Check the PPP HOWTO for more details, as getting the Bitsurfr working is mostly a fight with pppd.

### ISDN Network Adapters

There are a number of ISDN network adapters on the market. As the Linux kernel advances to 2.2, additional devices are being supported by the Linux built-in ISDN support. I should mention that an ISDN network adapter is not a modem. It does not consume a serial port and thus does not have the speed limitations problems of a serial port. One such ISDN network adapter is the SpellCaster DataCommute/BRI. SpellCaster is a Canadian company that is developing the Linux driver and is targeting the Linux market for their product. The card was reviewed recently in the October 1997 issue of *Linux Journal*.

An ISDN network adapter is an ideal solution for someone who cannot afford a full-fledged ISDN router and who does not want to be bogged down by a serial port's overhead. ISDN Network Adapters typically have high throughput compared to an ISDN modem. For example, the SpellCaster card uses a 16K memory buffer to transfer data from the ISDN line to the system and so generates fewer interrupts. Also, the card is not limited to 115K of a serial port. A two-B channel connection with an ISDN network adapter is usually at the full 128K speed. Yes, there are now higher speed serial ports; however, any non-

intelligent serial card requires the processor's attention more often than an ISDN network adapter.

An ISDN network adapter can also be used to roll your own ISDN router; out of Linux. Linux's firewall support is exceptional. This, perhaps, can give that old 386-40 a new lease on life. An ISDN modem also can be used to roll out a router, however, an intelligent serial card would be required to get comparable performance.

To demonstrate the setup of an ISDN network adapter, I will discuss the installation and use of the SpellCaster DataCommute/BRI. The DataCommute uses the ISDN4Linux support of the 2.x kernels. At the time of this writing, SpellCaster is hard at work on a new driver named Babylon, currently available in beta release. It promises to provide additional functionally and overcome some of the shortcomings of ISDN4Linux. Babylon also promises to be easier to set up as the biggest hurdle in getting the DataCommute to work is understanding the ISDN4Linux operation.

The same test system was used to test the SpellCaster DataCommute/BRI. Both dedicated and dial-up situations were tried again to UUNET Canada's ISDN service. The dedicated setup proved to be very fast. Transfer rates greater then an ISDN router were observed.

Setting up the SpellCaster DataCommute/BRI:

1. Compile your 2.x kernel with ISDN support. Run **make config** and say YES to: “ISDN Support”, “Support Synchronous PPP”, “Use VJ-compression with Synchronous PPP” and “Support generic MP”. Compile and install the new kernel.
2. Download the current driver and the ISDN4Linux support files from the SpellCaster web page (see Resources). The driver is the one named scis not dcbri.
3. Extract the driver and compile it by typing **make** in the driver's /src directory. This creates two executable files, **scctrl** and **sctrace**, along with the driver module, sc.o. Use scctrl to configure the DataCommute for the correct ISDN switch type and SPIDs.
4. As root, run **make install** to install the driver to the /lib/modules directory. Change the setup for your Linux distribution to have the driver load automatically on the next boot and then insert it into memory for this session (e.g., **insmod sc.o**).
5. Extract the ISDN4Linux package and compile it by typing **make**. As root, run **make install** to install the ISDN4Linux files. This will place **ipppd** and **isdnctrl** in the /sbin directory along with other programs. **ipppd** is the

synchronous version of pppd. **isdnctrl** sets up the ISDN interface and talks to ipppd to control the ISDN link.

6. Perform the following tasks with **scctl** as root:
   - Reset the DataCommute by typing **scctl -r**. All the lights on the back of the card will go out.
   - Start the firmware on the card by typing **scctrl -g**. The top light on the card will flash to indicate the firmware is running.
   - Program the ISDN switch type by typing **scctl -w NI1** (Use your telco's switch type here; type **scctl -h** to see the available switch types.)
   - Program the SPIDs and directory numbers:

```
scctl -c 1 -s 4165551234 -d 5551234\
        - the first channel
scctl -c 2 -s 4165551235 -d 5551235\
        - the second channel
```

   If all went well, the top light on the card will be flashing and the rest of the lights will be on, indicating the card has negotiated with the ISDN switch successfully. Running **scctl -u** will also show the card's status. The Switch Status will show CONNECTED. Some patience is required, as the negitiation between the ISDN switch and your card can take up to two minutes.

   Note that once the card has been programmed for the ISDN switch type and SPIDs, it need not be done again unless new firmware is loaded into the card. Thus, the above is only done once and not every time the system is booted.

7. Open another VT or Xterm and run **tail<\!s>-f<\!s>/var/log/messages**. This will give a constant output of the syslog file. **ipppd** outputs all debug information to the syslog, and it's handy to see what's going on while initially setting up the connection.
8. Type in the script shown in Listing 4, making the changes to the phone numbers, etc., where appropriate, and save it as **isdnsetup**. This script sets up the ISDN subsystem and runs ipppd.
9. Finally, do steps 11 and 12 from the list shown earlier for setting up Bitsurfr Pro.

Assuming everthing has gone well, run:

```
isdnctrl dial ippp0
```

to dial up your ISP. If you have configured two B channels, type:

```
isndctrl addlink ippp0
```

to bring up the second channel. To kill the connection at any time, type:

```
isdnctrl hangup ippp0
```

Incidentally, the above procedure is by no means the standard for configuring the ISDN subsystem and setting up a connection. With some experimentation in the case where a dedicated link is used, it should be possible to set up the link when the system is booted. Check the man page for **isdnctrl** for additional information. Be sure to disable the debug feature in the isdnsetup script once everything is working to avoid filling up the syslog file.

There is one more thing to note about using the card. Some PCI systems require that the BIOS be set up to reserve the IRQ and memory space for the DataCommute/BRI; this was true in the case of our test system. Generally, this is in the system setup under some form of Plug and Play menu. Otherwise, this card just works and is well worth the money.

## In Conclusion

There are three general methods of getting Linux on the Net with ISDN: an ISDN router, an external ISDN modem and an internal ISDN network adapter. The device you choose should be determined by the function of the link, the time limit on getting the link up and the cost of the device.

No matter which option you select, once you have an ISDN net connection, you will never return to 28.8K—it's just too slow!

Glossary

Resources

**Mark Buckaway** is a Software Developer with UUNET Canada. Mark's ISDN experience came from a year's tenure in UUNET's Technical Support department where he found himself as the ISDN Specialist. On his off time, Mark enjoys his dedicated ISDN connection to the Internet, as well as spending time with his 4-year-old son, Matthew, and 6-year-old daughter, Allison. Mark can be contacted at mark@datasoft.on.ca as well as http://www.datasoft.on.ca/~mark.

Archive Index Issue Table of Contents

Advanced search

# Letters to the Editor

**Various**

Issue #47, March 1998

Readers sound off.

## vi, most used text editor

First of all, let me tell you how much I appreciate *LJ*, for the accuracy of the information it gives and the nice tone it uses.

I write you in reaction to the “1997 Readers' Choice Award” that the text editor vi received in the December issue. I'm an old addict of this program, and I still find it fast and useful. But I have always promised myself that I would learn Emacs one of these days, thinking I couldn't remain an old dinosaur using such an old tool. The problem is that I always found Emacs far too complicated. So when I saw the Award, I was genuinely surprised so many people think like me and I had a good laugh.

In short, after reading *LJ*, I decided that, despite vi being an old and ugly editor, I'll keep using it without any remorse for being an Emacs loser. —Pierre-Philippe Coupard coupard@mipnet.fr

## Fujitsu Lifebook

I read the review of the Fujitsu Lifebook 420D in *Linux Journal* Issue 43 (November). Maybe some of the *LJ* readers would be interested in this information.

Since March 1997, I have owned a 520D, which uses a Chips&Technologies 6555x for its video chip set. It runs XFree86 just fine with up to 64K colors at 800x600 (virtual up to 1600x1200@8bpp), and acceleration using the SVGA server. I don't know if this model is still current; however, it's almost the same as the 420D except it does have an external floppy adapter. —Jeroen Beekhuis j.beekhuis@uci.kun.nl

### Mixing Linux and NT

"Best of Technical Support" in Issue 44 (December) suggested a way to mix Linux and NT. Here's another way using the NT boot loader. Assume NT is on /dev/hda1 and Linux is on /dev/hda2, and that /dev/hda1 is mounted on /dosc.

1. In Linux, edit the /etc/lilo.conf file so that:

```
boot=/dev/hda2
```

2. Run the following:

```
/sbin/lilo
dd if=/dev/hda2 of=bootsect.lin bs=512 \
        count=1
mv bootsect.lin /dosc
```

3. In NT, edit the file c:\boot.ini to include (make it the default if you wish):

```
c:\bootsect.lin='Linux'
```

When you reboot, the NT boot loader will give a menu of operating systems and then timeout to the default. —Valerie Crump valeriec@amc.com

### Jumbled Text and Setting Color

I have a few comments in regards to "Best of Technical Support", November 1997. Shells like bash and zsh perform their own line editing, and to do this they need to know how long the prompt is. With Jim's PS1, bash thinks the prompt is 15 characters longer than it actually is, because the colour setting escape sequences don't take up any screen space. This can easily be seen when typing a command which reaches column 65, because bash assumes it is actually at column 80 and wraps the text at that point. The real solution is listed in bash's man page under PROMPTING:

> **\[** begin a sequence of non-printing characters, which could be used to embed a terminal control sequence into the prompt.**\]** end a sequence of non-printing characters.

Jim's prompt then becomes:

```
\[\033[36m\]\u_\[\033[33m\]\W_\$_--\[\033[32m\]_
```

where "_" is a space. —Carey Evans c.evans@clear.net.nz

Yes, you are correct. My solution does address one problem but upon re-reading Jim's question, I realize that it will not solve his. Your solution is the correct one. Thanks Carey. —Chad Robinson chadr@brt.com

## Linux in the Mainstream?

I was reading the article in *Linux Journal* #43 called “Linux in the Mainstream?” by Phil Hughes. As a Linux user I do like the idea of Linux becoming more mainstream. However, I don't agree with the suggestions that Mr. Hughes makes. Linux is something different, that's why we use it—well, why I and many others use it. Linux started as a small project and was taken on by a large community. It is not like other operating systems. There is no need to *make* Linux mainstream—Linux is becoming mainstream on its own power. This is not because of marketing—it is because Linux is a damn fine OS, and people are finding that out by word of mouth. We, as a community, should not look into marketing and certification programs, but rather we should keep right on with what we have been doing—telling others about this great OS. We don't need a central authority to hand out meaningless sheets of paper.

I am not saying that Linux is not lacking in some areas. It most certainly is, but as a community we should be working on the shortcomings simply for the sake of making a better, free, OS—not to win over new users. —Steve Carpenter sjc@delphi.com

## SCO Unix Review

I really appreciated your recent review of SCO OpenServer. Where I work, we recently had reason to take a computer that had Linux on it and make it into a dual Linux/SCO system. I would like to point out, however, an error in the review. Ken says:

> While you're going through this process, OpenServer is merrily overwriting your master boot record and wiping it free of LILO.

While it is true that SCO overwrites LILO if you have it installed on the Master Boot Record (MBR), it is not true that LILO cannot boot SCO. In fact LILO is more than happy to boot SCO. The problem is that SCO expects its own partition to be active or bootable. From the README file for LILO:

> Some PC UNIX systems (SCO and Unixware have been reported to exhibit this problem) depend on their partition being active. Such a setup can currently only be obtained by installing LILO as the MBR and making the respective partition active.

If, after you install SCO, you reinstall LILO to the MBR and make the SCO partition bootable, LILO will very easily allow you to choose one or the other at boot time. On our setup, we have Linux installed to /dev/hda2 and SCO on /dev/hda4. Our lilo.conf file, therefore, looks like this:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
message=/boot/boot.msg
prompt
timeout=100
#
# Linux partition
image=/boot/vmlinuz
        label=linux
        root=/dev/hda2
        read-only
#
# SCO Unix partition
other=/dev/hda4
        label=sco
        table=/dev/hda
```

Upon bootup, LILO runs and displays our boot.msg file which tells the user how to load either Linux or SCO. This has worked out quite nicely for us. In the past, we had installed SCO on a machine that also used MS-DOS and the only way to switch between the operating systems was by using FDISK to toggle between the partitions. It's nice to see that Linux and its tools are still better than anything else out there. —Tanner Lovelacelovelace@acm.org

### Perl Option Error

Thanks for publishing my message in the “Letters to the Editor” in the December 1997, Issue 44. But you introduced a huge mistake in it, which can have security implications for readers who blindly trust *LJ*.

The message, published under the title “Big Brother”, mentions the **-T** option of the Perl interpreter, saying that “**-T** tests that the file type is text, not binary.” This is ridiculous and I never wrote that. I wrote that every Perl CGI programmer should use the **-T** option and explained that it refers to tainted mode (man perlsec for details). The **-T** *option* (a command-line flag) has nothing to do with the **-T** *function* (which indeed tests if a file is text). Any Perl programmer could have caught that mistake.

It seems to me that the treatment of my alert message (remember that anyone on the Internet could execute any command on a machine which uses the scripts you originally published) exhibited two serious flaws:

It was treated too slowly. Most people trust paper more than Usenet News or WWW. Many people probably assumed that the articles in *LJ* were carefully scrutinized and that the scripts were dependable. *LJ* had, in my opinion, a responsibility to warn users as soon as possible (at least in the next issue) of the mistake and not through a letter to the editor two issues later.

It is perfectly understandable that you edited my message; I know that my English is quite poor. But you could have sent it back to me for a last check. I do not think it is ethical to modify a message, not on a grammatical point but on a

technical one, and to publish it without showing to the readers the edited parts and without sending it to the author for proofreading. —Stephane Bortzmeyer bortzmeyer@pasteur.fr

First, let me apologize for your letter getting changed in a way that changed technical content. We try hard not to let this happen. One of our copy editors thought the **-T** needed more explanation and obviously grabbed the information from the wrong place. I agree he should not have added to the text without consulting you. If you had put as much detail in the first letter as you did above, I don't think he would have felt he needed to add anything. Ultimately, though, I did let his addition pass, and I take full responsibility for the error.

LTE is just about the last column I put together. Consequently, there is not a lot of time to pass it back and forth. It is also the first time I even see the letters, so they can be old. By the time a magazine comes out, the next issue is already at the printer, so errors never get corrected until two issues later. It's too bad, but such is the way of magazine deadlines.

Actually, I think you do quite well with your English —Editor

### Fan Mail

Just wanted to let you know that *Linux Journal* continues to be startlingly good. I'd say that 70% of your articles are of immediate interest to me each month, and the other 30% get re-discovered as useful knowledge when I go through my back issues. It's like you guys can read my mind! Keep up the amazing work. —Manni Wood mwood@sig.bsh.com

### From the Publisher

I got a chuckle out of your “From The Publisher” article on page 10 of *Linux Journal*, Issue 43 (Nov 97). It sounds as if IBM is relegated to the past. We have three IBM AS/400s (64 bit RISC) and 600 users, mostly on Windows 95 clients. IBM has passed Intel and Microsoft in both hardware and software terms (stop by http://www.as400.ibm.com/). OS/400 (V3R2 and later) has everything a company needs to do relational database Intranet/Internet applications. I'm using Net.Data (which is available on many platforms; see http://www.as400.ibm.com/netdata/) to do Provider lookups, map engine address lookups, etc. The inclusion of Java in V4R2 and IBM's business class libraries makes the AS/400 a very reliable database platform for the future. We have one table with over 28 million records, and I'm not quite ready to trust the MS SQL server to handle it.

How do you get Linux to be more mainstream? You have hackers like me who are established in the corporate world. I'm installing Linux to test JBuilder Java applets at home. (I want applications to run on AS/400, NT and Linux without modification.) I have a small network with NT server 4, NT workstation 4, Windows 98 beta and a P150 class laptop with NT workstation. I'm dumping my Windows 98 machine (486/66) to load Slackware Linux. I'm hoping for much better performance. —Steven P. Goldsmith fcci@cirs.com

## SAMBA

I was disappointed with the article “Using SAMBA to Mount Windows95” in the November issue. It gets a number of basic facts wrong.

For a start, SAMBA does not do what the article claims it does. The author is actually talking about a kernel-based SMB file system called **smbfs** written and maintained by Volker Lendecke (among others). Volker is a member of the SAMBA Team, but smbfs is definitely not a part of SAMBA. SAMBA is an SMB file server portable to all Unices, whereas smbfs is currently for Linux only.

The article also says that “SAMBA is a program that allows Linux to talk to computers running Windows for Workgroups, Windows 95, Windows NT, Mac OS and Novell Netware.” The bit about the various Microsoft operating systems is partly right (although he actually meant smbfs) but the part about Mac OS and Novell Netware is totally wrong. For those he is probably referring to MARS_NWE, ncpfs and Netatalk, which are totally separate packages that talk totally different protocols. Perhaps if you are publishing articles on a topic which *Linux Journal* editors are not very familiar with (there is a lot to know about Linux), you should send a copy to someone who is familiar with the topic so they can do a quick check and point out any obvious errors. —Andrew Tridgell andrew.tridgell@anu.edu.au

Well, you caught me—I don't know everything about Linux. However, to make up for this, I do send articles to copy editors who have given me their areas of expertise. The copy editor who worked on this article was a networking expert. I'll keep you in mind for the next SAMBA article that comes in. —Editor

Archive Index Issue Table of Contents

Advanced search

# Linux Wins Nokia Award

**Phil Hughes**

Issue #47, March 1998

On December 10, 1997, the Nokia Foundation granted its 1997 Award to Linus Torvalds, one of the world's most famous young Finnish researchers in the area of information technology.

If you have a Nokia cellular phone, use it to call your friends and let them know that Linus Torvalds won another award. On December 10, 1997, the Nokia Foundation granted its 1997 Award to Linus Torvalds, one of the world's most famous young Finnish researchers in the area of information technology. The award is worth FIM 50,000 (about $9400 US).

Nokia's press release stated that Linus' inspiring example for young researchers was of special interest in his selection. I don't think this comes as a surprise to those of us who have been members of the Linux community for the past few years, but it is nice to see such recognition from a foundation established by a large company.

The Nokia Foundation, established in 1995, supports the development of information and telecommunications education and technologies in Finland. In addition to the award presented to Linus, the Foundation granted more than thirty scholarships and one fellowship to further individual and institutional development at Finnish universities.

At the Grant Holder Announcement event, Nokia President and CEO Jorma Ollila emphasized the importance of education, research and development in maintaining the competitiveness of the Finnish telecommunications industry. Ollila discussed the need to complement study time with practical applications: “If we could shorten the study time by one year, for example, and spend that time on effective research and development work, we would remarkably strengthen our national competitiveness.” Obviously, the Nokia Foundation believed that combining graduate studies with the creation of an operating

system that is used around the world was a pretty good example of these principles.

Congratulations, Linus.

### More Than Research

As we all know, there is a lot more to Linux these days than research. December 19, the movie *Titanic* opened. If you read last month's *Linux Journal*, you know that a herd of Digital Alpha computers running Linux were used in the production of the film. While Finland isn't getting a big kickback from the box office receipts, Mr. Ollila is certainly right that research can make a difference.

Linux is doing a lot more out there than making movies. The Linux Means Business columns we run each month offer detailed looks into how government, business and industry use Linux. Another less detailed resource is a web page put together by Mercury Information Technology, Inc. This is a page dedicated to Linux business applications. What you find there are companies like Sony, Mercedes-Benz, Cisco Systems and Fluke as well as the companies you regularly see in the pages of *Linux Journal*.

This page is a great resource and M-tech has agreed to our mirroring it on the business connection link of the Linux Resources page (http://www.linuxresources.com).

New at *LJ*

Archive Index Issue Table of Contents

Advanced search

# GNU Ghostscript

**Robert A. Kiesling**

Issue #47, March 1998

Need to preview and print PostScript Files? Here's a utility that will do just that.

Viewing an Encapsulated PostScript recycling logo in a TeX document should have been simple enough. Instead, a blank space marked the position on the display where the logo should have appeared. A quick look through all of my LaTeX documentation, including the **xdvi** man page, provided no answer. There seemed to be no way for xdvi to display a TeX \special command in the dvi output; in this case, an embedded PostScript image. I considered this to be a major shortcoming of xdvi, and, of course, TeX. After all, isn't combining text and graphics on a page part of what computer publishing is all about? Why should TeX, one of the most sophisticated typesetting systems in the world, be constrained by the lack of a PostScript previewer? Or the requirement for a PostScript printer, for that matter? It is said that the longest journey begins with a single step. The journey I'm about to describe has brought me a considerable distance toward professional-quality publishing, via a utility that had been sitting quietly in a subdirectory of my /usr partition all along.

The utility is **Ghostscript**. GNU Ghostscript, Version 3.33, is the Swiss Army knife of graphics programs. Essentially, it's a PostScript language interpreter. More specifically, Ghostscript renders PostScript code to any bitmapped device on your system, whether it be the screen, printer, or G3 fax file. The man page says, “Ghostscript is a programming language similar to Adobe System's PostScript language, which is in turn similar to Forth.” (This statement comes almost verbatim from the book: Adobe Systems, Incorporated: *PostScript: Language Tutorial and Cookbook,* Addison-Wesley, 1986.) Ghostscript also substitutes fonts from its own library when rendering documents if the fonts are not otherwise available, scales and rotates text, and performs any of the other typographical feats for which PostScript is famous. Despite the apparent complexity of the task, no knowledge of the PostScript language is required. For the most part, a little experimentation with the command-line options is all that's needed. Yet Ghostscript's PostScript interpreter is fully accessible.

If I want to print that EPS recycling logo at the bottom of a page of DVI output, I would insert the following commands in my LaTeX input file:

```
\usepackage{graphics} ... \vfil \includegraphics{recycle.eps}
```

Then, it is simply a matter of running the input file through teTeX, using the command line:

```
pslatex letter.tex
```

and then converting teTeX's dvi output to PostScript with dvips:

```
dvips -f <letter.dvi >letter.ps
```

The **-f** command-line option tells dvips to act as a filter, reading from standard input and writing to standard output. Now, we have a PostScript file ready for printing. Ghostscript images this file in a format that's understandable to any bitmapped device on your system, whether it is a VGA display or a laser printer.

Ghostscript's command-line options are numerous. Look at ghostscript's USE.DOC file for an complete description. You can view a brief summary with the command:

```
gs -help | less
```

For example, if I want to print the file letter.ps on a Hewlett Packard DeskJet printer, I would use the command line:

```
gs -q -dNOPAUSE -sDEVICE=deskjet
/-sOutputFile=-\
          letter.ps quit.ps >gs.out
```

The first option, **-q**, suppresses messages to the console. However, to run Ghostscript in batch mode, the **-q** option on its own is not enough. That's because Ghostscript uses its own PostScript code at run time to output the image.

From the command line, we define several variables, the first being NOPAUSE, which tells Ghostscript's **showpage()** routine not to pause after each page is output. When viewing output on a screen, it is best to leave NOPAUSE unset and let Ghostscript prompt you to view each page in turn.

Next, we set the DEVICE variable. In the example above, the output device is a HP Deskjet. Ghostscript's dictionary has output parameters defined for all the devices documented in its command-line help.

The next variable which needs defining is OutputFile. Here, we set it to **-** or standard output. This is the file to which Ghostscript will write its image. In

most instances, however, Ghostscript provides no means for that image to be displayed. We'll get to that in a moment. Then we specify letter.ps on the command line as our target input file.

When we write a file with the **pslatex** command, LaTeX uses the font metrics available to it; that is, it uses Computer Modern Roman as its default font. However, the pslatex command has been implemented in teTeX so that dvips, which does the DVI to PostScript conversion, will request PostScript fonts. These fonts can be provided via either the printer or via software. In the latter case, Ghostscript produces the output image using Type 1 fonts from its own library.

Finally, we input the file quit.ps. This file is part of Ghostscript's standard library and is simply the command **quit** followed by a newline character. It is read into the Ghostscript interpreter just like any other input.

### Viewing Images

Viewing images on-screen is only a little more difficult. Ghostscript for Linux comes with both X11 and SVGALib support. The executables for both versions should be in the /usr/bin directory: gs-with-X11 and gs-without-X11. You can run either version using its actual file name, but the more common implementation uses a soft link of the actual executable to the **gs** command. In either case, the executable's permissions should be setuid root so it can access the display. We'll describe how to use Ghostscript as a PostScript previewer with both the X Window System and SVGALib.

### Ghostscript with X11

The X Window System provides a lot of the display support that Ghostscript would need to provide otherwise. As a result, it is by far the way to use Ghostscript as a previewer. As with the above example, we want to pass letter.tex through LaTeX and then convert the output to PostScript with the command line:

```
pslatex letter.tex; \
        dvips -f <letter.dvi >letter.ps
```

Again, we need to specify the DEVICE string to gs, which is simply X11 for the X display. Ghostscript treats an X11 display frame as the standard output, and the X display services provide the geometry to display an entire page. A virtual window manager like FVWM already provides the facilities to scroll the view over the entire page. The ghostscript command, then, is simply:

```
gs -r72 -sDEVICE=X11 letter.ps
```

This will provide us with an actual-size page on the screen, because we've overridden the default resolution with the **-r** switch.

A VGA display provides about 72 dots per inch resolution, so a legible, actual-size full U.S. letter page will not fit on the screen. This is why we rely on FVWM or another virtual window manager to scroll the view across the entire page. Ghostscript draws a page-high window on your X display. About half of the page is visible at a time. You can, of course, view the full page at twice its actual size by specifying your output resolution as 36dpi.

The simplest way to execute the Ghostscript command is in an xterm window. The page appears in a window which displays the child process of the Ghostscript command. Ghostscript writes its messages to standard error, which here is the xterm. Pressing **enter** in the xterm window tells **showpage** to display each successive page of Ghostscript output. You can set the default resolution in your ~/.Xdefaults file by adding the lines:

```
Ghostscript*xResolution: 72 Ghostscript*yResolution: 72
```

and then merge the defaults with the other X server defaults:

```
xrdb -merge ~/.Xdefaults
```

If the resolution isn't specified, the page is displayed on the screen at 300dpi, about one-quarter its actual size, which is visible in a 640x480 view without scrolling.

### Ghostscript with SVGALib

Things get a little more difficult when using a VGA display without X11 support. A standard Linux tty device provides no ready-made provision for paging over a full, U.S. letter-size image. Also, Ghostscript's SVGALib routines must be provided with geometry and resolution information to preview images. The following information is specific to my Compaq laptop, Chanel3, which has a 16 color, 640x480 standard VGA display, GNU Ghostscript Version 3.33 and SVGALib version libvga.so.1.2.10. You'll need to adjust the parameters to suit your hardware, but the basic procedure should be similar.

The settings that Ghostscript recognizes for various hardware configurations are listed in use.doc file and the gs man page. The DEVICE string is “linux” for a Linux virtual console; that is:

```
/dev/tty1 - /dev/tty9
```

This string corresponds to the virtual console's /etc/termcap entries. The **-r** resolution, parameter is one of several dozen VGA modes that SVGALib

recognizes. Ghostscript defines single-digit mode numbers which correspond to standard width-by-height notation. There is a complete list in the Ghostscript man page. In this case, Mode 4 is 16-color, 640x480 VGA. Much of the following information depends on whether SVGALib provides information on display geometry for your particular display. Ghostscript requires this information to display anything. It must be provided, for one thing, with the aspect ratio of the display (the ratio of the display's width to its height) in order to scale the fonts correctly.

The geometry parameter (**-g**) tells Ghostscript the display dimensions. Ghostscript scales the output page to the geometry we specify. We'll address this problem in a moment. With these parameters set, our Ghostscript command is:

```
gs -sDEVICE=linux -r4 -g640x480 letter.ps
```

Remember that gs is really a link to the executable file /usr/bin/gs-without-X11. Here, we tell Ghostscript to display a full page by specifying a display geometry that's twice the size of the actual screen. This gives us a page that is slightly more than twice as large as the video display. To double the size of the virtual display—the “device space” in PostScript jargon—we use the command line:

```
gs -sDEVICE=linux -r4 -g640x960 letter.ps
```

which gives us an actual-size view of the top half of the printed page. To view the bottom half of the page, we can specify an offset of the image's Y origin as half of a U.S. letter-size page:

```
gs -sDEVICE=linux -r4 -g640x960 -dYO=5.5
letter.ps
```

The default units are inches. The image's X origin can be shifted similarly.

### Ghostscript as a Document Post-Processor and Previewer

Integrating Ghostscript into your system is not that difficult. For example, if you routinely write documents in Emacs' LaTeX mode, the following bash script takes the DVI output of Emacs' tex-buffer command, converts it to PostScript, and then post-processes the output through Ghostscript. Finally, it sends the output to the print spooler. This script, **gsprint** (see Listing 1), can be called by Emacs' tex-print command directly. Note that the commands which call Ghostscript and then spool the output to the **lpr** daemon should all be typed on one line.

An even shorter version of this script, **gspreview** (see Listing 2), previews the document and can be called by Emacs' tex-view command under X11. Emacs provides the name of the TeX DVI file as the argument to its tex-print and tex-

view commands. All you need to do is specify the names of the external commands. First, make sure that the scripts are located in a directory in the search path (I use /usr/local/bin for my shell scripts). Give them execute permission with the command:

```
chmod a+x gsprint gspreview
```

Then add the elisp code shown in Listing 3 to your .emacs file. Whenever you use the tex-print or tex-view commands (**ctrl-c ctrl-p** and **ctrl-c ctrl-v**, respectively) in TeX-mode or LaTeX-mode, these shell scripts are called and their commands executed, using the DVI output of the most recent TeX command.

The next bash script, which I named **pvga** (see Listing 4), uses Ghostscript to preview output on non-X VGA displays. It takes as its argument the name of the TeX DVI output file and two optional arguments: a list of pages to be output and the Y-origin offset for each page. This script can be run from the command line or used as the core routine of a more complex VGA previewer. The list of pages that you want to view, formatted according to the dvips documentation, must be specified before the Y offset.

## PostScript in a (Virtual) Box

You can easily replace TeX's Computer Modern fonts with Ghostscript's scalable fonts. By default, dvips calls the MakeTeXPK program, which in turn calls MetaFont, to generate the physical Computer Modern fonts not present on the hard disk.

Printing is faster with bitmap fonts rather than scalable fonts, but scalable fonts that use Adobe's standard encodings provide the complete Adobe character set, including kerning and ligature pairs, which the Computer Modern fonts do not provide. With reasonably fast hardware, you can turn off dvips' font-generation feature and hardly notice a difference in speed. Dvips provides the **-V** command line switch for this purpose. The bash script **vgspreview** (see Listing 5) is a modification of gspreview, above. Remember to specify *zero* after the **-V** switch, which turns the font generation facility *off.*

## Conclusion

There are many other tasks that Ghostscript can perform with ease:

1. Create faxes.
2. Create PDF files that can be read by Adobe's Acrobat reader.
3. Generate a number of different graphics formats.

4. Work with other companies' GUI displays, notably Windows and Macintosh.

Since Ghostscript interprets the PostScript language, you can program directly in PostScript, either via Ghostscript's command interpreter or with \special commands embedded in your TeX and LaTeX files. This article has only scratched the surface of the capabilities of this free program and the many ways in which Ghostscript can perform feats of industry-standard imaging right on your desktop.

Glossary

When Robert Kiesling is not involved with the complexities of PostScript and TeX, he is at work on his “real” writing. This includes several novels, as well as fiction, poetry and nonfiction, which have appeared in literary magazines and newspapers nationwide. When he is not busy with either of the above, he is occupied by maintaining the Linux FAQ, providing editorial support to small presses and answering e-mail at rkies@cpan.org.

Archive Index Issue Table of Contents

Advanced search

# Colleges Using Linux

**Don Kuenz**

Issue #47, March 1998

How Casper College uses Linux to teach computer science courses.

Casper College is located in the city of Casper, Wyoming. Its total enrollment stands at about 4,000, with students coming from Wyoming, 26 other states and 9 foreign countries. You can visit Casper College's home page at http://www.cc.whecn.edu/.

The college uses several Unix systems for e-mail, web services, student records and accounting. At present, the bulk of teaching is done using personal computers running Windows 95, which are networked together with Novell Netware. Netscape is available on most machines.

## A Brief History

Casper College first used Linux as a teaching tool during the Spring 1995 semester when Ralph Anderl taught a short “Introduction to Unix” class. At first, Ralph used an SCO host, and later switched to a Linux host. Ralph worried that Linux might lock up when two dozen students simultaneously accessed it. His worry changed to amazement after Linux proved it could easily handle two dozen students and more. Linux found a home at Casper College and Ralph named the new Linux host “breeze”.

Why breeze? Well, Casper sees a lot of wind. Sometimes airline pilots come to Casper in order to train in windy conditions. The wind blows new pilots miles off course during landings, and even experienced pilots find it difficult to touch

down smoothly. So, in honor of its wind, Casper College gives its hosts names such as: “wind”, “breeze” and “dust”.

### Present Uses

LISP and C programming classes are taught directly on breeze by using a TELNET session on each student's computer. Other classes use home accounts on breeze as a private area where students can send and receive computer files containing handouts, tests and assignments.

I teach a class named “Computer Information Systems”, which introduces students to PC hardware, Windows 95 and applications such as PowerPoint. This is the first computer class for many students. Normally, e-mail is the best method to transfer tests, assignments and handouts between instructors and students. However, in this particular class, most students lack an e-mail account. So I set up a system that allows students to send and receive files using Netscape.

The first step involved setting up a web server on breeze. We used a CERN server at first and later switched to an Apache server. At this point, instructors could publish public course materials such as syllabi, HOWTOs and tests on Casper College's Intranet. However, ensuring private communication between instructors and students required a bit more work.

To ensure privacy, we created a breeze user account for each student which was readable only by the student. That allowed instructors, as root, to place confidential information such as test scores in each student's home directory. It also allowed students to send homework, assignments and tests to a private place.

Classroom experience taught me that students new to computers absolutely must use a web browser to transfer files. Although new students take to a GUI interface like a fish to water, a console interface such as FTP stops them dead in their tracks.

Fortunately, students can transfer files to or from their home directories using Netscape and opening a URL, such as ftp://don@breeze.

This causes Netscape to prompt for a password, then allows the user (in this case, Don) to access files in his home directory. To make things convenient we create mime types for Access, Excel, Lotus, PowerPoint, Schedule, Word and WordPerfect file types. We then add Netscape helper applications to each Windows 95 PC. Now, when a student clicks on a Lotus file, Netscape will automatically start up a Lotus application using the file as input.

Students take tests by using the Edit Document feature in Netscape Gold. This method provides the added benefit of giving them some practical experience with modifying web pages.

Remember that “Introduction to Unix” class I mentioned at the start of this article? Now Ralph uses Linux exclusively to teach the class. He also teaches out of a book which comes with its very own Linux CD-ROM. One class period is devoted to installing Linux on the personal computers in the lab. Many students also install Linux on their home computers. The UMSDOS file system comes in handy for students who are new to hard drive partitions, because it allows them to install Linux into a familiar file system.

## The Future

Casper College starts a new four-year Computer Science degree program during the Fall 1997 semester. You can bet that the presence of Linux will grow even stronger.

In 1975 **Don Kuenz** wrote his first program on a teletype machine connected to an HP computer, in BASIC. Don currently operates the software consulting company Kuenzsoftware, located in Casper, Wyoming. He also teaches computer science classes at Casper College. In his spare time he plays the piano, pedals around on a mountain bike and flies planes. He can be reached via e-mail at dkuenz@wind.cc.whencn.edu.

Archive Index Issue Table of Contents

Advanced search

# New Products

**Amy Kukuk**

Issue #47, March 1998

Arkeia Personal Shareware, Universal Server Enabled Empress RDBMS Version 8, Perforce 97.3 and more.

Arkeia Personal Shareware

Knox Software announced the release of its Personal Shareware version of Arkeia. The version provides the following features: interactive back-up and restore, one backup server, two Linux client machines, single-tape tape drive, SCSI, TRAVAN and no time limit. For more information on this free version of Arkeia, take a look at http://knox-software.vservers.com/downloadfree.html.

Contact: Knox Software Corp., 1325 Howard Avenue, Suite 328, Burlingame, CA 94010, Phone: 888-333-KNOX, E-mail: sales@knox-software.com, URL: http://knox-software.vservers.com/.

### Universal Server Enabled Empress RDBMS Version 8

Empress Software announced its new Universal Server enabled Empress RDBMS Version 8. The latest release extends Universal Server and object technologies for embedded development. These include integrated support for

relational and multimedia data, and the ability to create plug-in, reusable components for information management. Empress support of Universal Server technologies includes User Defined Functions (UDFs) and User Defined Procedures (UDPs). UDFs and UDPs are linked with the database kernel and are available at various levels, including the SQL level. Universal Server technologies are also supported with Persistent Stored Modules (PSMs).

Contact: Empress Software, 6401 Golden Triangle Drive, Greenbelt, MD 20770, Phone: 301-220-1919, E-mail: sales@empress.com, URL:http://www.empress.com/.

### Perforce 97.3

Perforce Software announced the release of Perforce 97.3, a software configuration management system. The new release offers improved usability due to portability and networking abilities. Perforce's database provides on-line backups. It is available for more than 30 platforms and has a suggested retail of $500 US.

Contact: Perforce Software, 2411 Santa Clara Avenue, Suite 40, Alameda, CA, 94501, Phone: 501-864-7400, E-mail: info@perforce.com, URL: http://www.perforce.com/.

### dbops

Simtech announced dbops, a collection of programs designed to manipulate ASCII databases using the Unix toolkit approach. **dbops** is a collection of about 45 binaries and a few scripts which address different aspects of database manipulation, including index building, searching and maintenance; record retrieval, posting, fragmentation and assembly; field editing; record display; menu presentation; screen manipulation and table manipulation and maintenance. A sample application is included which maintains a table that describes multiple PPP target sites. Also included is a set of scripts that will launch a semi-secure PPP connection to a specified PPP target. **dbops** is priced at $137.50 US.

Contact: Simtech, 6427 Windfern, Houston, TX 77040, Phone: 713-460-5921, E-mail: wfs@insync.net.

### Zeus Web Server v3.0

Zeus Technology announced the release of Zeus Web Server v3.0, a web application server. The server includes 128bit SSLv3 encryption worldwide, Apache compatibility and a web-centric management interface. It also offers unlimited virtual servers with configuration, real-time statistics, bandwidth

throttling and logging options. Commercial price for the Zeus Web Server v3.0 is $1699 US.

Contact: Zeus Technology, St. John's Innovation Park, Cowley Road, Cambridge,CB4 4WS, United Kingdom, Phone: 44-0-2334-421727, Fax: 44-0-1223-421731,E-mail: sales@zeus.co.uk, URL: http://www.zeustech.net/.

### ACUCOBOL GT Version 3.2

Acucobol, Inc. released ACUCOBOL GT Version 3.2, a portable and year 2000-compliant development system. Features include multi-threading, modeless windows, a dual file system, a full set of graphical controls and the ability to execute GUI programs in a non-GUI environment. There is a free evaluation copy available at their website.

Contact: Acucobol, Inc., 7950 Silverton Avenue, Suite 201, San Diego, CA 92126-6344, Phone: 619-689-7220, Fax: 619-689-7251, E-mail: info@acucobol.com, URL: http://www.acucobol.com/.

### WvDial 0.1

Worldvisions announced the release of WvDial 0.1. Some features of WvDial 0.1 include the ability to detect the user's internal or external modems on any of his serial ports and the ability to automaticall log in without a login script. The WvDial 0.1 release is available both as source code and as a Debian package. You can get either or both from the company's web site.

Contact: Worldvisions, E-mail: wvdial@worldvisions.ca, URL: http://www.worldvisions.ca/.

### TeamWave Workplace 2.1

TeamWave Software announced version 2.1 of TeamWave Workplace. TeamWave allows teams to work in shared virtual rooms across the Internet. TeamWave's rooms are customized with shared tools like whiteboards, chat, calendars, bulletin boards, documents, brainstorming and voting. Team members can work in rooms in real-time or leave information for later. Regular licenses are $50 US with quantity discounts available.

Contact: TeamWave Software Ltd., 100 Discovery Place One, 3553-31 Street NW, Calgary, Alberta T2L 2K7, Canada, Fax: 403-282-1238, E-mail: info@teamwave.com, URL: http://www.teamwave.com/.

Archive Index Issue Table of Contents

Advanced search

# Automated Mail Purging for SMTP Mail

**Michael S. Keller**

Issue #47, March 1998

Mr. Keller gives us three scripts for cleaning out old mail files automatically.

If your Linux system actively handles electronic mail, especially for several users, you may discover that mail consumes a lot of space where it resides, usually in one of /var/mail (System V-compliant systems), /usr/mail (BSD), /usr/spool/mail or /var/spool/mail (where my Debian GNU/Linux system stores it). The collection of Bash scripts presented here provides a way to reduce the space used by mail files by purging old messages.

## Background

For several months I worked as a contractor at a company that used a Unix system to process all electronic mail for its employees. Several hundred users retrieved their mail from a single host that served as both Simple Mail Transfer Protocol (SMTP) server (for general mail transport) and Post Office Protocol (POP3) server (for clients to retrieve mail without logging into the host). While it had a home-grown mail purge system, users could inadvertently defeat the system, allowing files to grow without end.

The existing mail purge system ran as a nightly **cron** job. It would determine the date 60 days before and construct a string in SMTP mail header format. It would then loop through all the user mail files in /var/mail using the **grep** command to find that string and note the lines on which it appeared. For each file that contained the string, it would use the **tail** command to discard everything prior to the line containing the target date string.

If messages arrived undamaged, stayed intact, always appeared in date sequence and cron called this job daily, this method would usually work. However, sometimes cron doesn't run a job, messages do not always arrive in date order and other software run against mail files might reorder messages.

Because of the existing solution's all-or-nothing keep-or-discard method, it had the following problems:

1. It was possible to lose newer messages if an older date appeared after a newer one.
2. If a message body had an un-escaped string that matched the search string, part of a message could be lost.
3. If older messages appeared out of order, they might remain longer than desired, since the keep/discard decision was based on simply finding a string, rather than examining each message's date.
4. It depended on finding the exact date string instead of making a numeric comparison that could discard messages older than the target date. A message containing invalid headers and lying at the end of the message file might remain, even if it had aged beyond the retention period.
5. A further effect is that some mail readers might not handle damaged messages gracefully.

### A Solution

To find a more reliable solution, I sought an existing free program that would address my needs. I asked my peers, including those on Internet mailing lists, but got no useful response. A search of Internet Unix archives revealed nothing, either. Thus, I was left to create a solution.

To ensure acceptable handling of messages, I settled on these requirements:

1. Each message must undergo individual examination to keep or discard it based on its creation date.
2. A reliable way to determine where messages began and ended was needed.
3. Because I discovered that date formats vary a bit (mail-handling programs only loosely follow the rules), I needed to convert each message date to a simple number and use that number for the keep/discard choice.

I discovered that **formail**, part of the **procmail** mail-handling package, can split an SMTP mail file into individual messages and repair damaged headers in one pass. This ability enables individual examination of each message to decide whether to keep or discard it. Since each message would undergo separate examination, date order would not be important.

### How It Works

The first script, mailrm.sh (see Listing 1), requires one command-line argument, naming the number of days of messages to preserve. When run, it sets needed

variables and starts checking each mail file in $MAILDIR. It uses formail, located in $FORMAIL, to verify, repair and split each mail file into individual messages.

Each message is then examined by the mailage.sh script (see Listing 2) to determine whether to keep it. First, it checks the message header's “From” line for the date, moving fields as necessary. (If formail has to repair a message date, the resulting date doesn't have a time zone in it.) Then it compares the message date with the value computed for today minus the number of days to retain messages. If the message is newer, the script concatenates the message onto STDOUT, saving it to a temporary file. If the message is older, the script exits with no output.

Afterward, if the new output file has a different number of lines from the original, it is moved into the original file's place, its ownership is reset and its permissions are restricted. If the original mail file is now zero length, mailrm.sh removes it. If a user has been removed from a system leaving his mail behind, this script deletes his mail file after all the messages in it have expired.

The third script, maildate.sh (see Listing 3), returns the integer number of days since 1900 of an input date in the form “MMM DD YYYY”. The returned integer is useful for calculating the difference between two dates.

## Installation

Copy the scripts to your favorite directory. I used /usr/local/bin. Edit mailrm.sh and mailage.sh to reflect your mail directory and the location of formail. Then just run the script with the number of days to retain messages as its argument. For example, to purge messages older than 60 days:

```
/usr/local/bin/mailrm.sh 60
```

## What Could Be Done Better

Is this mail purge solution perfect? No, it does nothing to lock mail files, which could pose a problem if a user's mail client polls frequently or this job runs during busy hours. Some possible steps to address this could include stopping **sendmail** while the scripts run, preventing the POP3 server from running and tightening permissions on the mail directory to prevent access from non-superusers. Since it would normally run in the early-morning hours via cron, the probability of collision would be low.

Also, this solution relies on external utilities that may not function as expected. **formail** might not properly handle all the mail headers, though I haven't encountered problems yet. **cat** might not like some characters that could

appear in messages, resulting in lost message text. I've had few problems with cat, but your experience may be different.

The date conversion logic in maildate.sh is simplistic. It's not accurate for the year 1900, and the leap year calculation will not work correctly after the year 2099. However, it works well for calculating the difference between two dates, and it's reasonably fast.

Since I use three scripts to get around parameter-passing limitations in Bash, this package runs more slowly than it might (because of having to fork processes repeatedly). Recoding the scripts into a single Perl script might help —my Perl skills are too limited for this project.

## Conclusion

If you have a need for automated mail purging, these scripts can help you reach your goal. At least, they may give you ideas for your own solution. If you create a more elegant solution, I'd like to hear about it.

## Resources

All listings referred to in this article are available by anonymous download in the file ftp://ftp.linuxjournal.com/lj/listings/issue47/2118.tgz.

**Michael S. Keller** (mskeller@paranet.com) is a Technical Analyst with Paranet, Inc., a nation-wide network services provider owned by Sprint. He has used computers for twenty years and Unix variants for seven. Paranet's virtual home is at http://www.paranet.com/.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL



# Networking with the Printer Port

**Alessandro Rubini**

Issue #47, March 1998

One of the strengths of Linux is its ability to serve both as engine for powerful number-crunchers and as effective support for minimal computer systems. The PLIP implementation is an outstanding resource in the latter realm, and this article shows its internals at the software level.

PLIP means “Parallel Line Internet Protocol”. It is a protocol used to bring IP traffic over a parallel cable. It works with any parallel interface and is able to transfer about 40KB per second. With PLIP you can connect any two computers at virtually no cost. Although nowadays ISA network cards are readily found and installed, you will still enjoy PLIP as soon as you get a laptop, unless you can afford a PCMCIA network card.

PLIP allows you to connect your computer to the Internet wherever there is a networked Linux box with a parallel port available, as long as you have root access on both systems (only root can load a module or configure an interface).

I find the internal design of PLIP quite interesting on three levels:

1. It shows how to use simple I/O instructions.
2. It lets you look at how network interfaces fit in the overall kernel.
3. It shows in practice how kernel software uses the task queues.

## Hardware Handling

Before showing any PLIP code, I'd like to describe the workings of the standard parallel port, so that you'll be able to understand how the actual data transfer takes place.

The parallel port is a simple device; its external connector exposes 12 output bits and 5 input bits. Software has direct access to the bits by means of three 8-bit ports: two ports for writing and one for reading. Moreover, one of the input

signals can trigger an interrupt; this ability is enabled by setting a bit in one of the output ports. Figure 1 shows how the three ports are mapped to the 25-pin connector. The **base_addr** of a parallel port (the address of its data port) is usually 0x378, 0x278 or 0x3bc. The vast majority of the parallel ports uses 0x378.

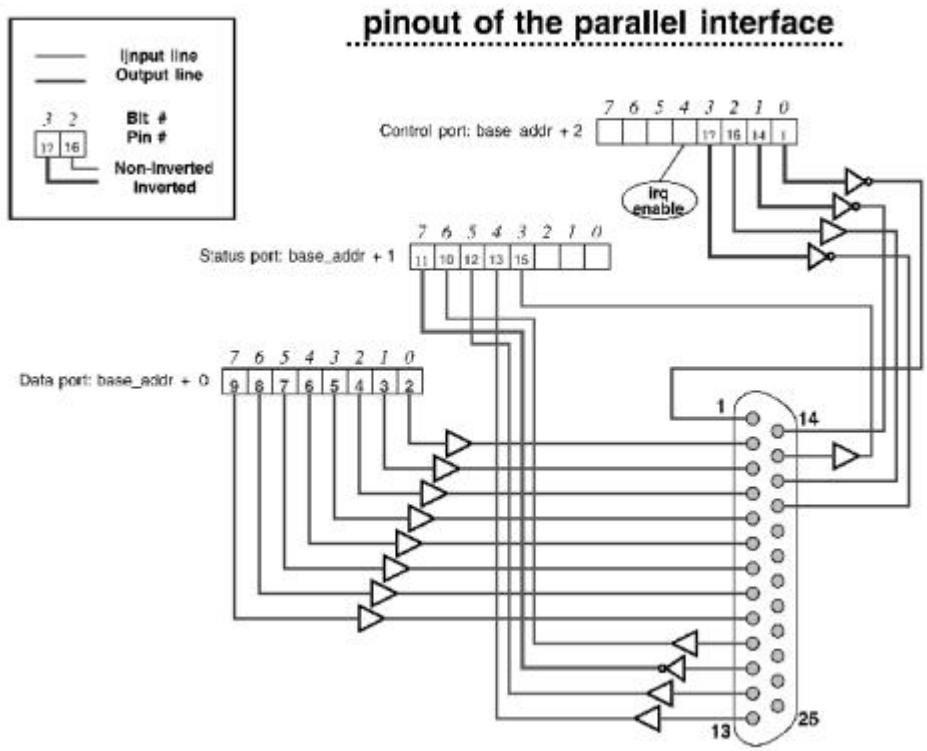


Figure 1. Mapping of Three Ports to Connector

Physical access to the ports is achieved by calling two C-language functions defined in the kernel headers:

```
#include <linux/io.h>
unsigned char inb(unsigned short port);
void outb(unsigned char value, unsigned short port);
```

The “b” in the function name means “byte”. Linux also offers *inw* (word, 16-bit), *inl* (long, 32-bit) and their output counterparts, but they are not needed to use the parallel port. The functions just shown are in fact macros, and they expand to a single machine instruction on most Linux platforms. Their definition relies on *extern inline* functions; this means you must turn optimization on when compiling any code using them. The reason behind this is somehow technical, and it is well explained in the gcc man page.

You don't need to be in kernel space to call **inb** and **outb**. If you want to access I/O ports from a shell script, you can compile inp.c and outp.c and play games with your devices (and even destroy the computer). For this reason, only root can access ports. The source code for inp.c and outp.c is available by

anonymous download in the file ftp://ftp.linuxjournal.com/lj/listings/issue47/2662.tgz.

### Communicating

Based on the description of the parallel port I provided in the previous section, it should be clear that two parties that communicate using PLIP must exchange five bits at a time, at most. The PLIP cable must be specially wired in order to connect five of the outputs of one side to the five inputs at the other side, and vice-versa. The exact pin out of the PLIP cable is described in the source file drivers/net/plip.c and in several places elsewhere, so I won't repeat the information here.

One of the deficiencies of the parallel port is the unavailability of any timing resources in hardware. Compare this with the serial port which contains its own clock. The communication protocol can't exploit any advanced hardware functionality, and any handshake must be performed in software. The chosen protocol uses one of the bits as a strobe signal to signal the availability of four data bits; the receiver must acknowledge receipt of the data by toggling its own strobe line. This approach to data transmission is very CPU intensive. The processor must poll the strobe signal to send its data, and system performance degrades severely during PLIP data transfers.

The time line of a PLIP transmission is depicted in Figure 2, which details the steps involved in the transmission of a single byte of information.

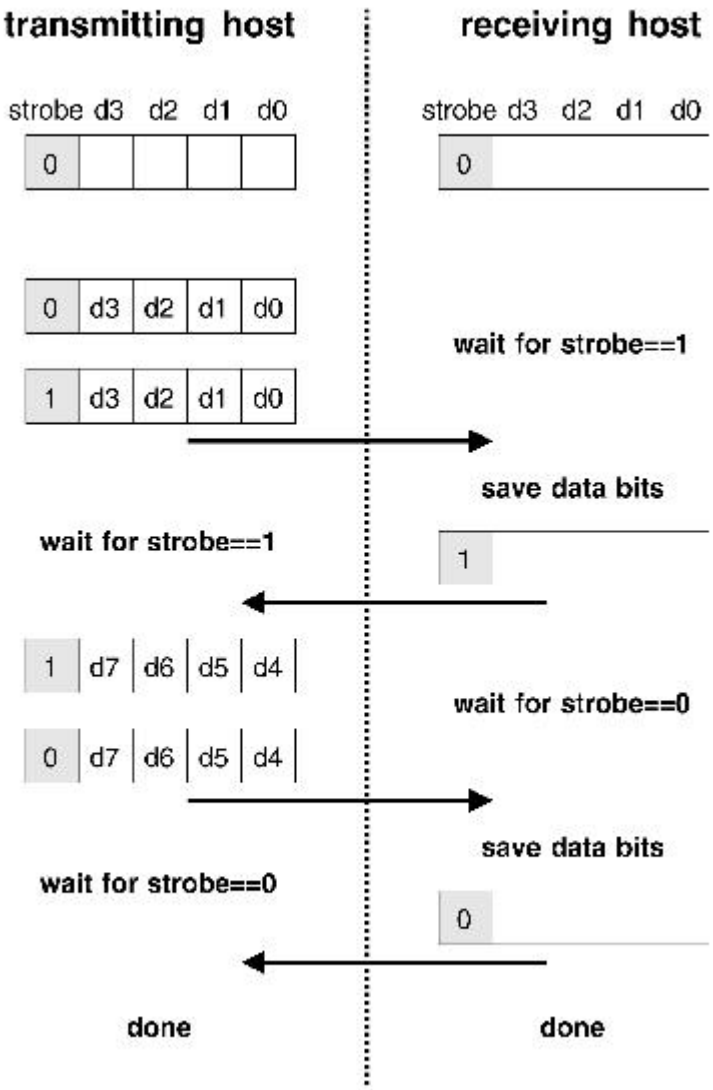


Figure 2. Time Line of PLIP Transmission

## Internet Datagrams

As far as the kernel proper is concerned, the PLIP device is just like any other network device. More specifically, it is like any other Ethernet interface even though its name is **plip***x* instead of **eth***x*.

When a datagram must be transmitted through a network interface, it is passed to the transmission function of the device driver. The driver receives a socket-buffer argument (**struct sk_buff**) and a pointer to itself (**struct device**).

With PLIP, transmission occurs by encapsulating the IP datagram into a "hardware header" for delivery, not unlike what happens for any other transmission medium. The difference with PLIP is that although it receives a data packet that already includes an Ethernet header, the driver adds its own

header. A packet encapsulated by PLIP, as it travels over the parallel cable, is made up of the following fields:

- **Packet length**: Transmission is headed by the length of the data packet in bytes, least significant byte first. The count is transmitted as a 16-bit number that doesn't include the count itself or the final checksum byte.
- **Ethernet header**: PLIP uses Ethernet encapsulation, so that programming the first implementation (in the PC world, when Linux didn't exist) was simplified. These fourteen bytes are almost useless, but they are still present for backward compatibility.
- **IP data**: IP data directly follow the header, just as with Ethernet interfaces.
- **Checksum byte**: The trailing byte of a PLIP transmission is a checksum byte, which must equal the sum modulo 256 of every data byte in the packet, excluding the leading two bytes (the length) and the checksum itself.

Whenever a packet is transmitted, all of the bytes are sent through the cable using the 5-bit protocol described earlier. This is quite straightforward and works flawlessly, unless something goes haywire during transmission.

### Asynchronous Operation

The interesting part of a PLIP communication channel is how asynchronous operations are handled. Transmission and reception of data packets must fit with other system operations and must be fault tolerant as much as possible. This involves several kernel resources and is quite interesting to anyone interested in kernel internals.

There are three problems to overcome to achieve reliable PLIP transmission. Outgoing packets must be transmitted asynchronously with respect to the rest of the system; even if transmission is CPU bound, it should happen outside of the normal computational flow. Incoming packets must also be received asynchronously, and they must be able to notify the PLIP device driver about their arrival. The last problem is fault tolerance; if one of the parties locks up transmission for any reason, we don't want the peer host to freeze while it waits for a strobe signal.

Asynchronous operation is achieved in PLIP by using the kernel task queues (which were introduced in the June 1996 issue of *Linux Journal*). Fault tolerance and timeouts are implemented using a state-machine implementation that interleaves PLIP transmission/reception with other computational activities without losing track of the internal status of the transmitter.

Let's look at PLIP cable connecting machines called Tanino and Romeo (Tanino = Tx, Romeo = Rx). The following paragraphs explain what happens when Tanino sends a packet to Romeo.

Tanino sends the signal to interrupt Romeo, disables interrupt reporting for itself and initiates the transmission loop by registering **plip_bh** in the immediate task queue and returning. When plip_bh runs, it knows that the interface is sending data and calls **plip_send_packet**.

Romeo, when interrupted (**plip_interrupt**), registers plip_bh in the immediate task queue. The plip_bh function dispatches computation to **plip_receive_packet**, which disables interrupt reporting in the interface and starts receiving bytes.

Tanino's loop is built on **plip_send** (which transmits a single byte) while Romeo's loop is built on **plip_receive** (which receives a single byte). These two functions are ready to detect a timeout condition, and in this case, they return the TIMEOUT macro to the calling function, which returns TIMEOUT to plip_bh.

When the callee aborts the loop by returning TIMEOUT, the plip_bh function registers a function in the timer task queue, so that the loop will be entered again at the next clock tick. If timeout continues after a few clock ticks, transmission or reception of this packet is aborted, and an error is registered in the **enet_statistics** structure; these errors are shown by the **ifconfig** command.

If the timeout condition doesn't persist at the next clock tick, data exchange goes on flawlessly. The state machine implemented in the interface is responsible for restarting communication at exactly the place where the timeout occurred.

As you see, the PLIP interface is fairly symmetrical.

### Plugging the Driver in Linux

As far as a network driver is concerned, being able to transmit and receive data is not the whole of its job. The driver needs to interface with the rest of the kernel in order to fit with the rest of the system. The PLIP device driver devotes more or less one quarter of its source code to interface issues, and I feel it worth introducing here.

Basically, a network interface needs to be able to send and receive packets. Network drivers are organized into a set of “methods”, as character drivers are (see *Dynamic Kernels: Discovery*, *LJ* April 1996). Sending a packet is easy; one of the methods is dedicated to packet transmission, and the driver just implements the right method to transmit data to the network.

Receiving a packet is somehow more difficult, as the packet arrives through an interrupt, and the driver must actively manage received data. Packet reception for any network interface is managed by exploiting the so-called “bottom halves”.

In Linux, interrupt handling code is split into two halves. The top half is the hardware interrupt, which is triggered by a hardware event and is executed immediately. The bottom half is a software routine that gets scheduled by the kernel to run without interfering with normal system operation. Bottom halves are run whenever a process returns from a system call and when “slow” interrupt handlers return. When a slow handler runs, all of the processor registers are saved and hardware interrupts are not disabled; therefore, it's safe to run the pending bottom halves when such handlers return. It's interesting to note that new kernels in the 2.1 hierarchy no longer differentiate between fast and slow interrupt handlers.

A bottom-half handler must be “marked”; this consists of setting a bit in a bit-mask register, so that the kernel will check whether any bottom half is pending or not. The immediate task queue, used by the PLIP driver, is implemented as a bottom half. When a task is queued, the caller must call **mark_bh(IMMEDIATE_BH)**, and the queue will be run as soon as a process is done with a system call or a slow handler returns.

Getting back to network interfaces, when a driver receives a network datagram, it must make the following call:

```
netif_rx(struct sk_buff *skb)
```

where **skb** is the buffer hosting the received packet; PLIP calls **netif_rx** from plip_receive_packet. The netif_rx function queues the packet for later processing and calls:

```
mark_bh(NET_BH)
```

Then, when bottom halves are run, the packet is processed.

In practice, something more is needed to fit a network interface into the Linux kernel; the module must register its own interfaces and initialize them. Moreover, an interface must export a few house-keeping functions that the kernel will call. All of this is performed by a few short functions, listed below:

- **plip_init**: This function is in charge of initializing the network device; it is called when **init_module** registers its devices. The function checks to see if the hardware is installed in the system and assigns fields in the **struct device** that describes the interface.

- **plip_open**: Whenever an interface is brought up, its **open** function is called by the kernel. The function must prepare to become operative (similar to the **open** method for character devices).
- **plip_close**: This function is the reverse of plip_open.
- **plip_get_stats**: This function is called whenever statistical information is needed. For example, the printout of **ifconfig** shows values returned by this function.
- **plip_config**: If a program changes the hardware configuration of the device, this function is called. PLIP allows you to specify the interrupt line at run time, because probing can't be safely performed when a module is loaded. Most of the parallel ports are configured to use the default interrupt line.
- **plip_ioctl**: Any interface that needs to implement device-specific **ioctl** commands must have an ioctl method. PLIP allows changing its timeout values, although I never needed to play with these numbers. The **plipconfig** program allows changing the timeouts.
- **plip_rebuild_header**: This function is used to build an Ethernet header in front of the IP data. Ethernet interfaces that use ARP don't need to implement this function, as the default one for the Ethernet interface does all of the work.
- **init_module**: As you probably already know, this is the entry point to the modularized driver. When a network interface is loaded to a running system, its **init_module** should call **register_netdev**, passing a pointer to **struct device**. Such a structure should be partly initialized and should include a pointer to an **init** function which completes initialization of the structure. For PLIP, such a function is plip_init.

These functions, along with **hw_start_xmit**, the one function responsible for actual packet transmission, are all that's needed to run a network interface within Linux. Although I admit there's more to know in order to write a real driver, I hope the actual sources can prove interesting to fill the holes.

### More Information

My choice to discuss PLIP is motivated by the easy availability of such a network connection, and the do-it-yourself approach that might convince someone to build their own infrared Ethernet link. If you really are going to peek in the sources to learn how a network interface works, I'd suggest starting with loopback.c, which implements the **o** interface, and skeleton.c, which is quite detailed about the problems you'll encounter when building a network driver.

If you are more keen to use PLIP than to write device drivers, you can refer to the PLIP-HOWTO in any LDP mirror, and to /usr/doc/HOWTO in most Linux installations.

*Alessandro is always wondering why laptops have a floppy drive instead of a second parallel port. He reads e-mail as rubini@linux.it, where you can bug him about Linux in general and device drivers in particular. He has written a book called* Linux Device Frivers for O'Reilly and Associates.

Archive Index Issue Table of Contents

Advanced search

# Writing HTML with m4

**Bob Hepple**

Issue #47, March 1998

Ease your creation and maintenance of web pages using this handy pre-processor called m4.

It's amazing how easy it is to write simple HTML pages—and the availability of WYSIWYG (what you see is what you get) HTML editors like Netscape Gold lulls one into a mood of “don't worry, be happy”. However, managing multiple, inter-related pages of HTML rapidly gets very difficult. I recently had a slightly complex set of pages to put together, and I started thinking, “there has to be an easier way.”

I immediately turned to the WWW and looked up all sorts of tools—but quite honestly I was rather disappointed. Mostly, they were what I would call “typing aids”—instead of having to remember arcane incantations like **<a href="link"7gt;text</a>** text, you are given a button or a magic keychord like **alt-ctrl-j** which remembers the syntax and does all the typing for you.

Linux to the rescue—since HTML is built as ordinary text files, the normal Linux text management tools can be used. This includes revision control tools such as **rcs** and the text manipulation tools like **awk**, **Perl**, etc. These tools offer significant help in version control and managing development by multiple users as well as automating the process of displaying information from a database (the classic **grep |sort |awk** pipeline).

The use of these tools with HTML is documented elsewhere, e.g., Jim Weirich's article in *Linux Journal* Issue 36, April 1997, “Using Perl to Check Web Links”. I highly recommend this article as yet another way to really flex those Linux muscles when writing HTML.

What I will cover here is work I've done recently using the pre-processor **m4** to maintain HTML. The ideas can very easily be extended to the more general SGML case.

## Using m4

I decided to use m4 after looking at various other pre-processors including **cpp**, the C front-end, which is perhaps a little too C-specific to be useful with HTML. **m4** is a generic and clean macro expansion program, and it's available under most Unices including Linux.

Instead of editing *.html files, I create *.m4 files with my favourite text editor. These files look something like the following:

```
m4_include(stdlib.m4)
_HEADER(`This is my header')
<P>This is some plain text<P>
_HEAD1(`This is a main heading')
<P>This is some more plain text<P>
_TRAILER
```

The format is just HTML code, but you can include files and add macros rather like in C. I use a convention that my new macros are in capitals and start with an _ character to make them stand out from HTML language and to avoid name-space collisions.

The m4 file is then processed as follows to create an .html file using the command:

```
m4 -P <file.m4 >file.html
```

This process is especially easy if you create a makefile to automate these steps in the usual way. For example:

```
.SUFFIXES: .m4 .html
.m4.html:
        m4 -P <$*.m4 >$*.html
DEFault:        index.html
*.html: stdlib.m4
all:    default PROJECT1 PROJECT2
PROJECT1:
        (cd project2; make all)
PROJECT2:
        (cd project2; make all)
```

Some of the most useful commands in m4 are listed here with their cpp equivalents shown in parentheses:

- **m4_include:** includes a common file into your HTML (#include)
- **m4_define:** defines an m4 variable (#define)
- **m4_ifdef:** a conditional (#ifdef)
- **m4_changecom:** change the m4 comment character (normally #)
- **m4_debugmode:** control error diagnostics
- **m4_traceon/off:** turn tracing on and off
- **m4_dnl:** comment

- **m4_incr**, **m4_decr:** simple arithmetic
- **m4_eval:** more general arithmetic
- **m4_esyscmd:** execute a Linux command and use the output
- **m4_divert(i):** This is a little complicated, so skip on first reading. It is a way of storing text for output at the end of normal processing. It will come in useful later, when we get to automatic numbering of headings. It sends output from m4 to a temporary file number *i*. At the end of processing, any text which was diverted is then output, in the order of the file number *i*. File number **-1** is the bit bucket and can be used to comment out chunks of comments. File number 0 is the normal output stream. Thus, for example, you can use m4_divert to divert text to file 1, and it will only be output at the end.

### Sharing HTML Elements Across Several Pages

In many "nests" of HTML pages, each page shares elements such as a button bar containing links to other pages like this:

```
  [Home]  [Next]  [Prev]  [Index]
```

This is fairly easy to create in each page. The trouble is that if you make a change in the "standard" button-bar then you have the tedious job of finding each occurrence of it in every file and manually making the changes. With m4 we can more easily do this job by putting the shared elements into an m4_include statement, just like C.

Let's also automate the naming of pages by putting the following lines into an include file called button_bar.m4:

```
m4_define(`_BUTTON_BAR',
 <a href="homepage.html">[Home]</a>
 <a href="$1">[Next]</a>
 <a href="$2">[Prev]</a>
 <a href="indexpage.html">[Index]</a>)
```

and then these lines in the document:

```
m4_include button_bar.m4
_BUTTON_BAR(`page_after_this.html',
  `page_before_this.html')
```

The $1 and $2 parameters in the macro definition are replaced by the strings in the macro call.

### Managing HTML elements that often change

It is troublesome to have items change in multiple HTML pages. For example, if your e-mail address changes, you need to change all references to it to the new

address. Instead, with m4 you can put a line like the following in your stdlib.m4 file:

```
m4_define(`_EMAIL_ADDRESS', `MyName@foo.bar.com')
```

and then just put **_EMAIL_ADDRESS** in your m4 files.

A more substantial example comes from building strings with multiple components, any of which may change as the page is developed. If, like me, you develop on one machine, test out the page and then upload to another machine with a totally different address, then you could use the m4_ifdef command in your stdlib.m4 file (just like the **#ifdef** command in cpp). For example:

```
m4_define(`_LOCAL')
...
m4_define(`_HOMEPAGE',
 m4_ifdef(`_LOCAL',
  `//127.0.0.1/~YourAccount',
  `http://ISP.com/~YourAccount'))
m4_define(`_PLUG', `<A HREF="http://www.ssc.com/linux/">
<IMG SRC="_HOMEPAGE/gif/powered.gif"
ALT=<"[Linux Information]"> </A>')
```

Note the careful use of quotes to prevent the variable **_LOCAL** from being expanded. **_HOMEPAGE** takes on different values according to whether the variable **_LOCAL** is defined or not. This definition can then ripple through the entire project as you build the pages.

In this example, **_PLUG** is a macro to advertise Linux. When you are testing your pages, use the local version of **_HOMEPAGE**. When you are ready to upload, remove or comment out the **_LOCAL** definition in this way:

```
m4_dnl m4_define(`_LOCAL')
```

... and then re-make.

### Creating New Text Styles

Styles built into HTML include things like **<EM>** for emphasis and **<CITE>** for citations. With m4 you can define your own new styles like this:

```
m4_define(`_MYQUOTE',
 <BLOCKQUOTE><EM>$1</EM></BLOCKQUOTE>)
```

If, later, you decide you prefer **<STRONG>** instead of **<EM>**, it is a simple matter to change the definition. Then, every **_MYQUOTE** paragraph falls into line with a quick **make**.

The classic guides to good HTML writing say things like “It is strongly recommended that you employ the logical styles such as **<EM>...</EM>** rather than the physical styles such as **<I>...</I>** in your documents.” Curiously, the WYSIWYG editors for HTML generate purely physical styles. Using the m4 styles may be a good way to keep on using logical styles.

### Typing and Mnemonic Aids

I don't depend on WYSIWYG editing (having been brought up on **troff**) but all the same I'm not averse to using help where it's available. There is a choice (and maybe it's a fine line) to be made between:

```
<BLOCKQUOTE><PRE><CODE>Some code you want to display.
</CODE></PRE></BLOCKQUOTE>
```

and:

```
_CODE(Some code you want to display.)
```

In this case, you would define **_CODE** like this:

```
m4_define(`_CODE',
<BLOCKQUOTE><PRE><CODE>$1</CODE></PRE></BLOCKQUOTE>)
```

Which version you prefer is a matter of taste and convenience although the m4 macro certainly saves some typing. Another example I like to use, since I can never remember the syntax for links, is:

```
m4_define(`_LINK', <a href="$1">$2</a>)
```

Then, instead of typing:

```
<a href="URL_TO_SOMEWHERE">Click here to get to SOMEWHERE
</a>
```

I type:

```
_LINK(`URL_TO_SOMEWHERE', `Click here to get to SOMEWHERE')
```

### Automatic Numbering

**m4** has a simple arithmetic facility with two operators **m4_incr** and **m4_decr**. This facility can be used to create automatic numbering, perhaps for headings, for example:

```
m4_define(_CARDINAL,0)
m4_define(_H, `m4_define(`_CARDINAL',
 m4_incr(_CARDINAL))<H2>_CARDINAL.0 $1</H2>')
_H(First Heading)
_H(Second Heading)
```

This produces:

```
<H2>1.0 First Heading</H2>
<H2>2.0 Second Heading</H2>
```

### Automatic Date Stamping

For simple date stamping of HTML pages, I use the m4_esyscmd command to maintain an automatic timestamp on every page:

```
This page was updated on m4_esyscmd(date)
```

which produces:

```
This page was last updated on Fri May 9 10:35:03 HKT 1997
```

### Generating Tables of Contents

Using m4 allows you to define commonly repeated phrases and use them consistently. I hate repeating myself because I am lazy and because I make mistakes, so I find this feature an absolute necessity.

A good example of the power of m4 is in building a table of contents in a big page. This involves repeating the heading title in the table of contents and then in the text itself. This is tedious and error-prone, especially when you change the titles. There are specialised tools for generating a table of contents from HTML pages, but the simple facility provided by m4 is irresistible to me.

### Simple To Understand TOC

The following example is a fairly simple-minded table of contents generator. First, create some useful macros in stdlib.m4:

```
m4_define(`_LINK_TO_LABEL',
 <A HREF="#$1">$1</A>)
m4_define(`_SECTION_HEADER',
 <A NAME="$1"><H2>$1</H2></A>)
```

Then define all the section headings in a table at the start of the page body:

```
m4_define(`_DIFFICULTIES',
 `The difficulties of HTML')
m4_define(`_USING_M4', `Using
 <EM>m4</EM>')
m4_define(`_SHARING', `Sharing HTML
 Elements Across Several Pages')
```

Then build the table:

```
<UL><P>
 <LI> _LINK_TO_LABEL(_DIFFICULTIES)
 <LI> _LINK_TO_LABEL(_USING_M4)
 <LI> _LINK_TO_LABEL(_SHARING)
<UL>
```

Finally, write the text:

```
 ...
_SECTION_HEADER(_DIFFICULTIES)
...
```

The advantages of this approach are twofold. If you change your headings you only need to change them in one place, and the table of contents is then automatically regenerated. Also, the links are guaranteed to work.

### Simple To Use TOC

The table of contents generator that I normally use is a bit more complex and requires a bit more study, but it is much easier to use. It not only builds the table, but it also automatically numbers the headings on the fly—up to four levels of numbering (e.g., section 3.2.1.3), although this can be easily extended. It is very simple to use as follows:

1. Where you want the table to appear, call **Start_TOC**.
2. At every heading use **_H1(`Heading for level 1')** or **_H2(`Heading for level 2')** as appropriate.
3. After the last line of HTML code (probably **</HTML>**), call **End_TOC**.

The code for these macros is shown in Listing 1. One restriction is that you should not use diversions (i.e., m4-divert) within your text, unless you preserve the diversion to file 1 used by this TOC generator.

### Simple Tables

Other than Tables of Contents, many browsers support tabular information. Here are some funky macros as a short cut to producing these tables. First, an example (see Figure 1) of their use:

```
<CENTER>
_Start_Table(BORDER=5)
_Table_Hdr(,Apples, Oranges, Lemons)
_Table_Row(England, 100,250,300)
_Table_Row(France,200,500,100)
_Table_Row(Germany,500,50,90)
_Table_Row(Spain,,23,2444)
_Table_Row(Danmark,,,20)
_End_Table
</CENTER>
```

| | Apples | Oranges | Lemons |
|---|---|---|---|
| England | 100 | 250 | 300 |
| France | 200 | 500 | 100 |
| Germany | 500 | 50 | 90 |
| Spain | | 23 | 2444 |
| Denmark | | | 20 |

Figure 1. Example Table

## m4 Gotchas

Unfortunately, m4 needs some taming. A little time spent on familiarisation will pay dividends. Definitive documentation is available (for example, in the Emacs info documentation system) but, without being a complete tutorial, here are a few tips based on my experiences.

## Gotcha 1—Quotes

**m4**'s quotation characters are the grave accent ` which starts the quote, and the acute accent ' which ends it. It may help to put all arguments to macros in quotes, for example:

```
_HEAD1(`This is a heading')
```

The main reason for using quotes is to prevent confusion if commas are contained in an argument to a macro, since m4 uses commas to separate macro parameters. For example, the line **_CODE(foo, bar)** would put the foo in the HTML output but not the bar. Use quotes in the line **_CODE(`foo, bar')**, and it works properly.

## Gotcha 2—Word Swallowing

The biggest problem with m4 is that some versions of it *swallow* key words that it recognises, such as *include*, *format*, *divert*, *file*, *gnu*, *line*, *regexp*, *shift*, *unix*, *builtin* and *define*. You can protect these words by putting them in single quotes, for example:

```
Smart people `include' Linux in their list
of computer essentials.
```

The trouble is, this is both inconvenient and easy to forget.

A safer way to protect keywords (my preference) is to invoke m4 with the **-P** or **--prefix-builtins** option. Then all built-in macro names are modified so that they all begin with the prefix **m4_** and ordinary words are left as is. For example, using this option, one would write m4_define instead of define (as shown in the examples in this article). One hitch is that not all versions of m4 support this option—most notably some PC versions under MS-DOS.

## Gotcha 3—Comments

Comment lines in m4 begin with the # character—everything from the # to the end of the line is ignored and output unchanged. If you want to use # in the HTML page, you must quote it like this: **`#'**. Another option (my preference) is to change the m4 comment character to something exotic with a line like this:

```
m4_changecom(`[[[[')
```

and not have to worry about # symbols in your text.

If you want to use comments in the m4 file but not have them appear in the final HTML file, use the **macro m4_dnl** (dnl = **D**elete to **N**ew **L**ine). This macro suppresses everything until the next newline character.

```
m4_define(_NEWMACRO, `foo bar')
m4_dnl This is a comment
```

Yet another way to have source code ignored is the m4_divert command. The main purpose of m4_divert is to save text in a temporary buffer for inclusion in the file later—for example, in building a table of contents or index. However, if you divert to "-1", it just goes to limbo-land. This option is useful for getting rid of the whitespace generated by the **m4_define** command. For example:

```
m4_divert(-1) diversion on
m4_define(this ...)
m4_define(that ...)
m4_divert diversion turned off
```

### Gotcha 4—Debugging

Another tip for when things go wrong is to increase the number of error diagnostics that m4 outputs. The easiest way to do this is to add the following to your m4 file as debugging commands:

```
m4_debugmode(e)
m4_traceon
...
buggy lines
...
m4_traceoff
```

### Conclusion

It should be noted that HTML 3.0 does have an include statement that looks like this:

```
<!--#include file="junk.html" -->
```

However, the HTML include has the following limitations:

- The work of including and interpreting the include is done on the server-side before downloading and adds overhead as the server has to scan files for include statements.
- Most servers (especially public ISPs) deactivate this feature because of the large overhead.

- Include is all you get—no macro substitution, no parameters to macros, no ifdef, etc., as with m4.

There are several other features of m4 that I have not yet exploited in my HTML ramblings so far, such as regular expressions. It might be interesting to create a "standard" stdlib.m4 for general use with nice macros for general text processing and HTML functions. By all means download my version of stdlib.m4 as a base for your own hacking. I would be interested in hearing of useful macros, and if there is enough interest, maybe a Mini-HOWTO could evolve from this article.

There are many additional advantages to using Linux to develop HTML pages, far beyond the simple assistance given by the typical typing aids and WYSIWYG tools. Certainly, I will go on using m4 until HTML catches up—I will then do my last **make** and drop back to using pure HTML. I hope you enjoy these little tricks and encourage you to contribute your own.

**Bob Hepple** has been hacking at Unix since 1981 under a variety of excuses and has somehow been paid for it at least some of the time. It's allowed him to pursue another interest—living in warm, exotic countries including Hong Kong, Australia, Qatar, Saudi Arabia, Lesotho and (presently) Singapore. His initial aversion to the cold was learned in the UK. Ambition—to stop working for the credit card company and tax man and to get a real job. Bob can be reached at bhepple@pacific.net.sg.

Archive Index Issue Table of Contents

Advanced search

# Best of Technical Support

**Various**

Issue #47, March 1998

Our experts answer your technical questions.

### Implementing Ballistic Mouse Control

I've been spoiled by operating systems that use “ballistic mouse” control (a fast motion of the mouse moves the mouse a lot while a slow motion moves the mouse very little). I'm using Metro-X but seem to recall the same limitation with XFree.

Is there a patch or parameter available which will enable a mouse to respond to the speed of the mouse movement? I'm running at a fairly high resolution and I have to pick my mouse up two or three times to move the cursor from one side of the screen to the other. —Mike Hall

You can do this using the **m** option of the **xset** command. This command lets you set parameters for both acceleration and a threshold for the number of pixels the mouse must travel before it accelerates. Check out the xset man page. —Samuel Ockman & Larry Augustin, VA Research ockman@varesearch.com and lma@varesearch.com

### How Do I Remove This File?

I have Red Hat 4.2, and I have noticed something strange at the root “/” directory. I have a *file* named “..”. This is in addition to the normal directories “.” and “..”. How do I delete a —Kevbo

The file cannot truly be called “..” because the “..” directory is present. Its name is probably “.. ” (with a trailing space) or something similar. The problem here is finding the name of the file so you'll be able to remove it. Try typing the command **ls -l > listfile** and then look in listfile with a binary capable editor like Emacs, or use the “set list” command in vi to show the end of line, and trailing blanks will be revealed. Then, using double quotes or wildcard characters

(expressions like “..?”), you'll be able to identify the file on the command line and remove it.

Other files that are often hard to remove are those beginning with a minus sign. (For example a file named “-test”). To remove these files just use a relative pathname, e.g., —Alessandro Rubini alessandro.rubini@pluto.linux.it

### Error During Red Hat Install

I can't install Red Hat 4.0 on my system. I have a mainboard with a P200 MMX and USB support. When I try to autoboot from a CD-ROM the system starts loading Linux but then suddenly reboots with error messages like “Unknown PCI-device”. I can only install Slackware from a boot disk when I press PAUSE a few times while loading the image. What can I do about this? —Mario Vos Red Hat 2.0.0

Most likely the kernel you are trying to boot can't deal with your PCI controller. Sometimes, though, a system prints what appears to be an error message but works anyway. Make sure you aren't just seeing what you think are errors and not giving it a chance to work.

If there truly are errors the best thing to do is to try a more recent version of Red Hat (4.2 currently). —Donnie Barnes, Red Hat Software redhat@redhat.com

### Does Linux Support Multicasting?

Can Linux support native IP multicasting over PPP? If so, what version(s) must I use and what are the configuration requirements? —Don Skillen Caldera 2.0.29

Make sure your kernel has been compiled with these options:

- *IP: multicasting*
- *IP: forwarding/gatewaying*
- *IP: multicast routing (experimental)*
- *IP: tunneling*

You should also check out the web page http://www.teksouth.com/linux/multicast/. It is a great resource for information on Linux multicasting. —Mark Bishop mark@vincent.silug.org

### Why Is My Sendmail Slow?

I have two machines connected via 10Base coaxial cable. One is running Linux Slackware 2.0.30 and the other is running Windows 95. I've configured **Sendmail** on my Linux machine. I use a Netscape mailer on the Win95 machine.

When I send mail between the two machines Sendmail takes a very long time to send the mail even though it's just a test mail. The mail coming from Netscape to the Linux box is turbo fast. —Ronneil Camara Slackware 3.3

Your machine is most likely trying to do a reverse DNS lookup to determine the host name of the machine from which you are connecting. The delay you are experiencing is the time waiting for the DNS response. On the Linux box, run the “host” command with the IP number of your Windows 95 machine. It should return the name of the Windows 95 machine. If it doesn't, your name server setup is not correct.

If you don't have an external name server (provided by your ISP), and don't want to create your own, make sure that the daemon **named** is not running on your Linux machine; confirm this by typing:

```
ps ax | egrep -i named
```

*Next add an entry for the Windows machine to the /etc/hosts file.*

*If you are running named, check its setup. You should run reverse DNS to map IP numbers to host names.* The Network Administrator's Guide by Olaf Kirch is freely available and is a good guide to setting up a name server.

If your Internet service provider is providing the name server, make sure your /etc/resolv.conf file correctly points to your ISP's name server, and talk to your ISP to make sure they are correctly providing reverse DNS. —Larry Augustin, VA Research lma@varesearch.com

### Getting Started With Linux

What files do I need to get started? Where do I get them? How do I install them? —Michael Hall

Linux is best installed from a distribution, which is a collection of usable kernels, software and utilities. Distributions generally have decent installation programs that allow you to set up and install the packages that come with them.

There are many distributions, such as Slackware, Red Hat, Debian and others. If you are new to Linux and have no Unix experience, you may wish to buy a book on Linux since that book will come with a Linux CD and easy installation instructions.

If you want to do this on your own, visit the primary Linux FTP site, http://sunsite.unc.edu/, or a mirror of this site. Look in the /pub/Linux/distributions

directory. Each distribution has different requirements and installation procedures, so you will have to look for README (and other) files there.

For experienced computer users who want to get into Linux without a book, I recommend Slackware. It can be installed from almost any media, even DOS-formatted floppy disks, and it's somewhat easier to figure out than other distributions when you go it alone. —Chad Robinson, BRT Technologies chadr@brt.com

Archive Index Issue Table of Contents

Advanced search