Advanced search

Linux Journal Issue #113/September 2003

PUT YOUR SUMP PUMP ON THE WEB
LINUX JOURNAL
This beat box is an on-line music portal
Control Your Own Wireless Network
Secure login for wireless users
Wi-Fi frenzy sweeps New York City
Mapping wireless networks with Kismet
Chessbrain: Can 646 Computers beat you at chess?
Dynamic Kernel Modules • Rapid development with pyGTK and Glade
Garbage Collection for C and C++ • Managing users with LDAP

Archive Index

Advanced search

Discovering Wireless Networks

Tony Steidler-Dennison

Issue #113, September 2003

Take a walk in your neighborhood and map the available wireless network connections with a free application.

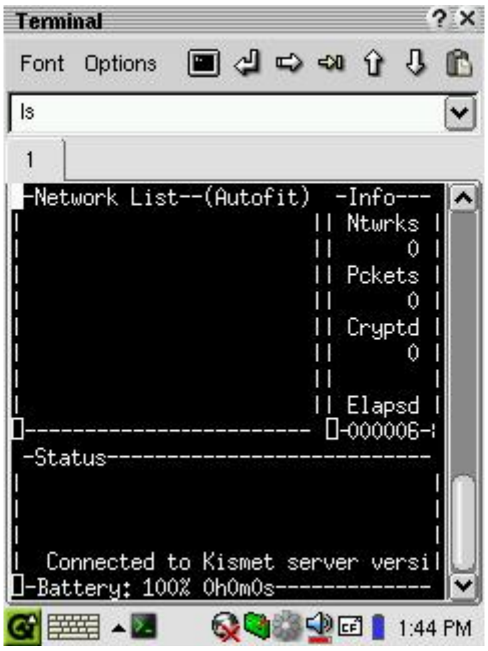
We're clearly on the upside of the wireless wave, with new installations happening across the country and around the world every day. Like many technologies in an early emergent phase, providers still seem to be at a loss as to how to get the word out that they're making this valuable service available. Discovering the wireless networks in your community on your own can be a challenge. If you live in an urban area, it's likely that hundreds of publicly accessible wireless networks are just waiting for you to log in. If you live in a rural area, they may be harder to come by, but they're there. What you need is a set of tools to help you find and use the networks made publicly available for your convenience.

The first tool, of course, is the computer itself. Because of the near ubiquity, increasing power and decreasing size of laptops, they've become the tool of choice for wireless discovery. Or, you can go with the PDA. Though classified as a PDA (or, in the marketing parlance of Sharp, a Personal Mobile Tool) the power of the Zaurus makes it a de facto desktop in a pocket size. Armed with the proper software tools, the Zaurus is much lighter on the wrists and arms for discovering networks, while sacrificing virtually nothing in power.

The software tool of choice for discovering wireless networks with the Zaurus is Kismet, available as a .tar.gz file from killefiz.de/zaurus. Kismet can be used in tandem with Kismet-Qt, a clean and extremely friendly GUI interface that provides all the information you need to sniff and connect to the wireless networks in your community. You don't need a Zaurus to use Kismet; it works under any Linux system.

For a quick taste of the Kismet command-line interface, slide your wireless CF card into the CF slot and, from the terminal mode, enter `kismet` as root. Your network card will begin to flash rapidly, indicating that Kismet is sniffing for wireless networks by analyzing all packets it encounters.

The Kismet command-line application indicates the number of networks in close proximity, as well as the number of packets received and how many of those packets are encrypted. To configure the command-line tool to your particular preferences, use vi to edit /home/root/usr/etc/kismet.conf. This is a well-commented configuration file, providing detailed instructions for each configurable element of Kismet. There are many, all of which are applicable to your use of Kismet-Qt.

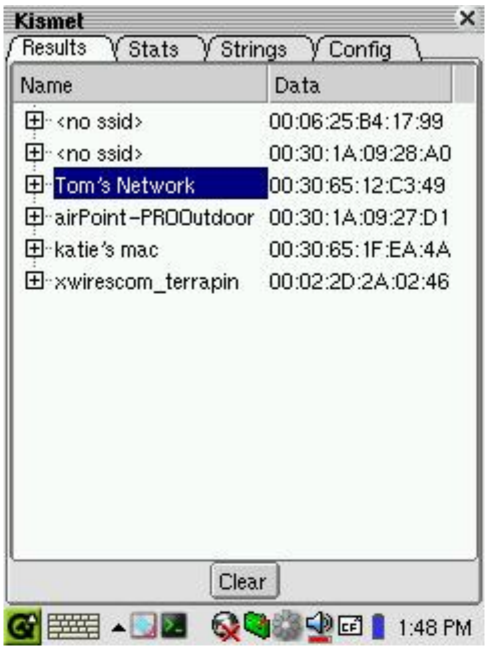


```
-Network List--(Autofit)  -Info---
|                        || Ntwrks |
|                        ||      0 |
|                        || Pckets |
|                        ||      0 |
|                        || Cryptd |
|                        ||      0 |
|                        ||        |
|                        || Elapsd |
-----------------------  -000006-
-Status---------------------------
|                                 |
|                                 |
|                                 |
|  Connected to Kismet server versi|
-Battery: 100% 0h0m0s-------------
```

Figure 1. The Kismet Command-Line Interface

Installation of the Kismet-Qt application is a bit more straightforward. With the .ipk file downloaded and transferred to your Zaurus, the file appears in the Add/Remove Software option within the Tools tab. By default, the file actually is stored at /home/zaurus/Documents/Install_Files. You can select the application from the Add/Remove Software window, select install and your install location (internal Flash or an external storage device), then let the Zaurus do the heavy lifting for you. If you'd prefer to install the file using the terminal application, change to this directory and follow the ipkg instructions above to install.

Command-line Kismet installs a server from which the Kismet-Qt application pulls its information. If the Kismet server isn't running, Kismet-Qt throws an error noting it was unable to connect to this server. So, you need to start the server from the terminal window as above, then open the Kismet-Qt interface by selecting it from the applications button in the lower-left corner of your Zaurus screen. With the Kismet-Qt package installed and Kismet running in your terminal window, you're ready to start discovering open wireless networks. Let's take a look at the Kismet-Qt interface.



Kismet

Results | Stats | Strings | Config

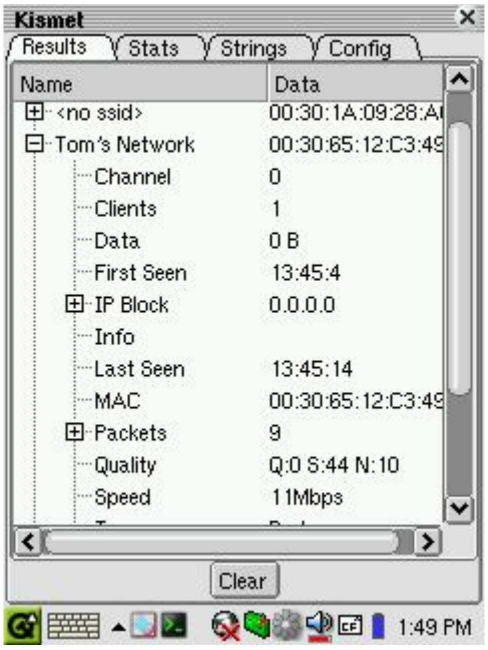
Name	Data
<no ssid>	00:06:25:B4:17:99
<no ssid>	00:30:1A:09:28:A0
Tom's Network	00:30:65:12:C3:49
airPoint-PROOutdoor	00:30:1A:09:27:D1
katie's mac	00:30:65:1F:EA:4A
xwirescom_terrapin	00:02:2D:2A:02:46

Clear

1:48 PM

Figure 2. The Kismet-Qt Interface

As shown in Figure 2, the Kismet-Qt interface is broken out into four main tabs. These tabs display the current network activity, a statistical summary of all current activity, a summary of all special strings detected and a configuration screen. These tabs provide more than enough detail for most users to detect and connect to community wireless networks. Let's take a look at some of these tabs individually.



Kismet

Results | Stats | Strings | Config

Name	Data
<no ssid>	00:30:1A:09:28:A
Tom's Network	00:30:65:12:C3:49
Channel	0
Clients	1
Data	0 B
First Seen	13:45:4
IP Block	0.0.0.0
Info	
Last Seen	13:45:14
MAC	00:30:65:12:C3:49
Packets	9
Quality	Q:0 S:44 N:10
Speed	11Mbps

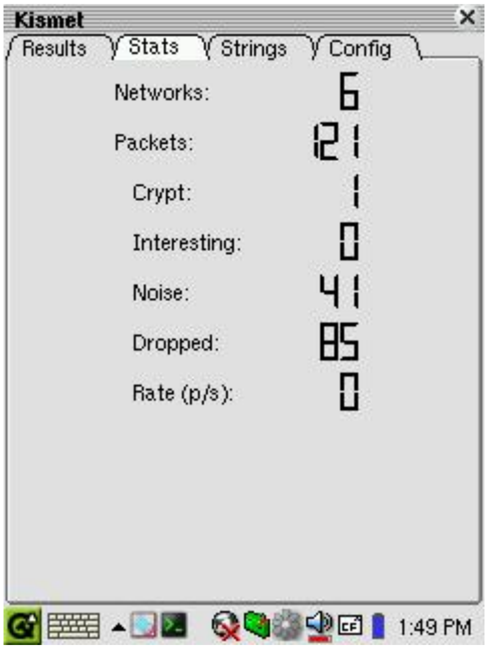
Clear

1:49 PM

Figure 3. The Kismet-Qt Results Tab

Kismet-Qt retains the data on all network activity detected during the current session. This data is presented in the Results tab in collapsible form, identified at the top level by the network name (or ESSID). Although the Results tab provides a wealth of information, a caveat or two is in order when viewing discovered networks. In cases where the wireless network does not utilize an ESSID, the default name is displayed within angle brackets. Seeing a name within the list such as "linksys" may be an indication that you've stumbled upon a private wireless network operating in the manufacturer's default configuration. Although the owner's intention may be to provide this service to the community, tread lightly. If there's any question about the purpose of the network, avoid using the network you've detected.

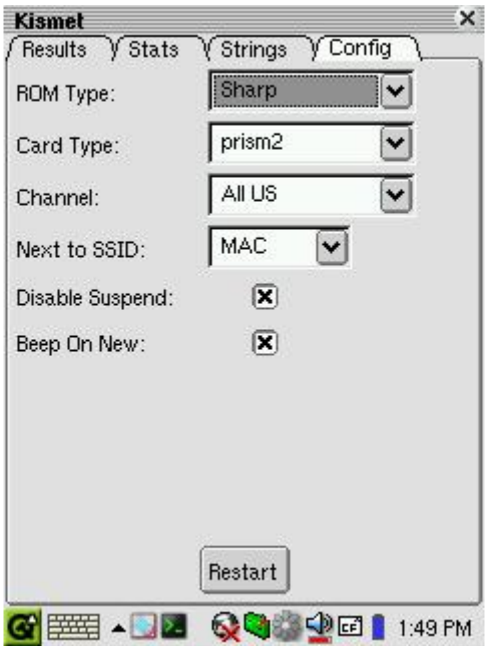
The Results tab provides virtually all the information you need to establish a connection with an open wireless network. We'll discuss these items in a bit. In particular, you need to note the ESSID, whether the signal is WAP-encrypted, the IP-address range and the channel on which the network is operating. These are your keys to the wireless kingdom. You also may want to pay attention to the signal strength figures and the time the last activity was detected ("last seen"). This data will provide wireless users on the move with an indication of their relative location to the hot spot. A weaker signal indicates you're on the fringe of the hot spot, moving either in or out.



```
Networks:       6
Packets:      121
  Crypt:        1
  Interesting:  0
  Noise:       41
  Dropped:     85
  Rate (p/s):   0
```

Figure 4. The Kismet-Qt Stats Tab

Kismet-Qt provides a clean interface to display the aggregate current wireless activity. This data includes the number of wireless networks currently within range, both the total number of packets received and the number that are encrypted, the signal noise level and the current packets-per-second receipt rate. Because this is an aggregate summary of the current activity, these numbers can be very high when encountering overlapping wireless zones. The rate-per-second figure, as with the signal strength and last seen figures above, can provide some indication of the user's movement into and out of the heart of a network.


Kismet

Results | Stats | Strings | Config

ROM Type: Sharp

Card Type: prism2

Channel: All US

Next to SSID: MAC

Disable Suspend: ☒

Beep On New: ☒

Figure 5. The Kismet-Qt Config Tab

The Config tab in Kismet-Qt contains the most critical data for making the Zaurus a wireless sniffer. You can select the system ROM type (Sharp or OpenZaurus) and the type of CF card you're using. Most common CF wireless cards are covered in the card type options, though you can modify the settings manually with the Other option if your card does not utilize one of the listed protocols. You also can select the operating channel from a list that provides the option of listening for specific individual channels or all US or international channels. The latter two options listen across the range of the US or international channels for any with current activity. Perhaps the most utilitarian option in the Config tab is the ability to beep when a new network is discovered.

The depth of the data provided by the Kismet/Kismet-Qt combination is quite rich. The tools provide all the data you need to sniff, connect to and utilize wireless networks in your community. Community wireless is coming to a town near you. With a Zaurus, Kismet and a wireless CF card, you can leverage the power of Linux to stay connected nearly anywhere.

Tony Steidler-Dennison is director of operations for Optical Mechanics, Inc. He builds observatory-grade robotic telescopes, configures the Linux systems that run them and installs the telescopes around the world. He can be reached at tony@steidler.net.

Linux-Powered Wireless Hot Spots

Mike Kershaw

Issue #113, September 2003

If you're setting up a wireless gateway for work or a public place, configure it to authenticate users and prevent abuse.

Wireless access in public areas is provided by 802.11 hot spots, with varying types of access depending on the desires of the hot spot operator. Many commercial hot spots exist in locations such as Starbucks or fast-food franchises, whereas libraries and conference halls might choose to provide free services to visitors and attendees. A public hot spot can be entirely altruistic, offering visitors free network access in the area; it can be a business opportunity, charging visitors for network access and services; or it could be a combination, allowing visitors restricted access and providing paying customers increased bandwidth or greater access.

Proprietary solutions are available for creating wireless hot spots, but why go for a closed-source solution when your favorite operating system and freely licensed tools can do it on a spare PC?

As the operator of a public access point, you have several options. The easiest, of course, is simply to connect a wireless access point (AP) to your network and allow all traffic. Unfortunately, the simplest route is not necessarily the safest one from a security standpoint. If your hot spot is designed to offer connectivity to the Internet at large or if it is connected to a segment of your private network, you almost certainly don't want to allow unfettered access to random passersby.

Many access points have controls to limit access by port and MAC address, but they don't offer any other tools for managing new users, logins or providing the user with information about the hot spot. The hot spots we discuss building here provide a captive portal, a system where users who have not logged in are forced to a single Web page with login, policy and, optionally, payment information.

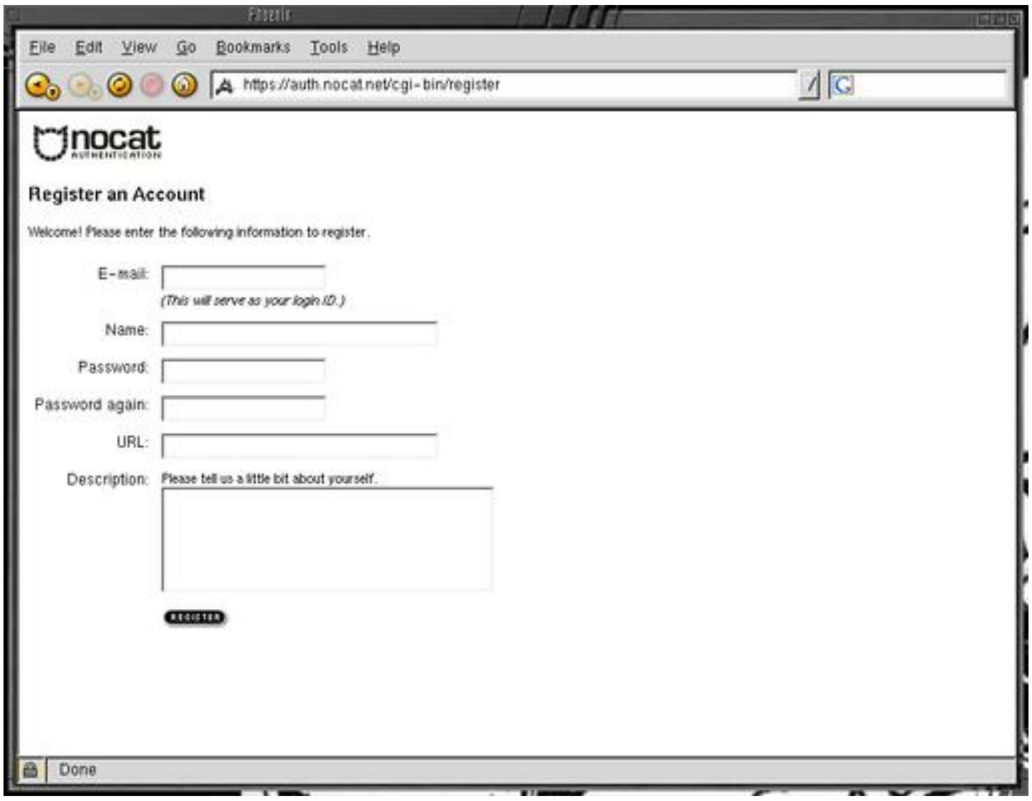
Hardware and Software

What do you need to get started providing a wireless hot spot? The list, fortunately, is short:

- Your favorite Linux distribution.
- NoCatAuth hot spot/portal software (free, open-source and available at www.nocat.net).
- A wireless access point, or several. Access points function as a bridge between the wired and wireless segments of your network. What type of wireless you choose depends on the users of your network. Currently, 802.11b is the most widely used; however, 802.11a offers higher data rates over shorter distances. 802.11g hardware, which provides higher data rates and is backward-compatible with 802.11b, is becoming more prominent (see the Sidebar 802.11a, b or g?).
- A moderately powered PC (Pentium or Pentium II class is more than sufficient for routing packets). NoCat suggests having at least two servers —one to act as a firewall and router and one to handle authentication— but a single server will do in a pinch.

NoCatAuth

NoCat builds a captive portal by assigning incoming users an IP address using DHCP and restricting network access until the user has validated, be it as a guest, paying customer or administrator. By rewriting the destination of all port 80 traffic in the firewall, any Web page the user attempts to visit before validating can be rerouted to the portal login page.


File Edit View Go Bookmarks Tools Help
https://auth.nocat.net/cgi-bin/register
nocat
REGISTER
Done

Register an Account

Welcome! Please enter the following information to register.

E-mail:
(This will serve as your login ID.)
Name:
Password:
Password again:
URL:
Description: Please tell us a little bit about yourself.

Figure 1. The NoCat Portal Login Page

Not all access points have the ability to control what MAC addresses are allowed on the network, and each manufacturer that does has a different method of configuring it, so NoCat uses the standard Linux iptables firewalling to control network access. It works with any access point. NoCat cannot prevent users from associating with the wireless network nor would you want it to; if a user can't associate, they can't log in. But it does prevent them from gatewaying to the outside network. Because NoCat dynamically rewrites the firewall rules to allow new users and deny disconnecting users access to the wired network, it's best to use a dedicated system that doesn't have other iptables rules on it already as a gateway.

NoCat consists of two main components: the gateway, which handles user logins and routing packets from the wireless network to the real network, and the authentication server, which stores user accounts and passwords. Typically, one gateway server is used per access point, and a single authentication server is used for a given installation.

The NoCat authentication system can use a simple flat-file password system, a MySQL database, a Radius or LDAP server or a Windows domain login over Samba to validate a user. The authentication server can be located on the local wired network or elsewhere on the Internet.

You can run your own authentication server on the same hardware as your gateway. However, it's more secure and easier for multiple gateways to use a single authentication server if you use separate machines for the authentication server and gateway.

Figure 2. Multiple NoCat gateways providing three hot spots linked to the same wired network using a single authentication server.

Before downloading and installing NoCat, you should begin planning what level of access you want users to have and what your acceptable use policies will be. Although the majority of your users likely will be honest, it's possible someone may attempt to cause mischief. Your port restrictions and acceptable use policies must strike a balance between being strict enough to prevent abuse and being permissive enough that the service is useful. Although any port can be used by a determined mischief maker to cause problems, many hot spots choose to allow SSH (port 22), HTTP (port 80) and HTTPS (port 443).

Other ports that may be useful to your users include POP3 (110), IMAP (143) and SMTP (25), but these carry their own risks to users and to your network. POP3 and IMAP traffic typically is not encrypted, which means users checking their e-mail risk having their passwords captured either in transit to their server or by someone sniffing wireless traffic in the area. Allowing SMTP, especially to unauthenticated users, can be dangerous because it could help a spammer connect to your network. The chances of someone sending massive bulk mailings through your hot spot probably are slim, but taking precautions always makes sense.

Building NoCat itself is a simple process: simply download the NoCat tarball from www.nocat.net, do `make gateway` and `make install` and edit the configuration file, /usr/local/nocat/nocat.conf. For full functionality, install a DHCP server and configure it to hand out private addresses for your wireless network.

NoCat is controlled by the nocat.conf file. A basic gateway needs:

- GatewayMode: controls the type of portal you run. An open portal allows anyone to use it once the terms on the splash page are accepted. A closed portal requires users to authenticate before they get access.
- IncludePorts and ExcludePorts: these control what TCP ports users are allowed to access. If you're running an open portal, you almost certainly want to restrict these to prevent abuse of the network. IncludePorts allows only the listed ports to be used, and ExcludePorts allows any ports but the listed ports to be used.
- InternalDevice and ExternalDevice: control the network interfaces that NoCat uses. The InternalDevice specifies the device to which the access point is connected, and the ExternalDevice specifies the wired network.

- LocalNetwork and DNSAddr: LocalNetwork should be set to the network used on your wireless network, and DNSAddr should be set only if you have a DNS server outside your wireless network. If you have a DNS server on your wireless segment, you won't need this option.
- AuthserviceAddr, AuthServiceURL and LogoutURL: control the type of authentication service the portal uses. By default, they are configured to use the NoCat authentication servers. If you're running a closed portal, though, you'll almost certainly want to set these to be your own authentication system.

Your portal can work with only the minimal configuration, but investigate the other configuration options to customize your portal's interface and rules.

Five Steps to a Simple Portal

Our example portal is a basic open portal. It needs only a single access point and server, because we don't need our own authentication system. We'll also lock it down so anonymous users can browse Web pages, use AIM and SSH to remote systems, but they won't have access to other TCP ports. This is a reasonably secure configuration for anonymous users, but you may wish to restrict it even further and allow only Web page access.

For our example portal, we use the following equipment:

- A generic access point, such as the Linksys WAP11, D-Link DWL900, the SMC 2655W or countless others. Normally, access points function as bridges connecting all users on a wired and wireless network, but in our setup we use the NoCat gateway to restrict traffic.
- A Linux server with at least 32MB of RAM, preferably a Pentium class processor or better; your favorite Linux distribution with Perl, Apache-SSL, iptables, Bind and DHCP; and two Ethernet cards. Alternatively, use one Ethernet card and one network card of whatever type is needed to connect to your external network. Once we have all the equipment, we can put the portal together in five easy steps:

1. Configure the Linux gateway to be on the real network and the wireless networks and to connect to server DHCP and DNS addresses. Plug the first Ethernet card (eth0) in to the access point and the second Ethernet card (eth1) in to the real network. Each distribution has its own method of configuring network cards, so here we use the command line. You'll need to configure your system to set up these interfaces at boot time. We're using 123.123.123.123 as an example address; you should change this to whatever your real address is:

   ```
   ifconfig eth0 192.168.1.1 \
   netmask 255.255.255.0 up
   ```

```
ifconfig eth1 123.123.123.123 \
netmask 255.255.255.0 up
```

2. Configure BIND (the DNS server) to be a basic caching DNS server listening to the local network. The default bind installation comes with an example caching configuration.
3. Configure DHCP to hand out addresses in the 192.168.1.*X* network by putting the following in /etc/dhcpd.conf. If your DNS server is not 192.168.1.1, change the domain-name-servers option line accordingly:

```
max-lease-time 120;
default-lease-time 120;
allow unknown-clients;

subnet 192.168.1.0 netmask 255.255.255.0 {
    option routers 192.168.1.1;
    option broadcast-address 192.168.1.255;
    option subnet-mask 255.255.255.0;
    option domain-name-servers 192.168.1.1;
    range 192.168.1.100 192.168.1.254;
}
```

4. Compile and install NoCat.
5. Configure NoCat:

```
InternalInterface       eth0
ExternalInterface       eth1
LocalNetwork            192.168.1.0/255.255.255.0
GatewayMode                     open
IncludePorts            22 80 443 5190
```

Things to Remember

When setting up and operating a wireless hot spot, you should remember a few things. First, protect your user information. If you allow users to sign up for your service over the wireless, and especially if you plan to take credit-card information, configure Apache for SSL encryption. Second, encourage users to use encryption wherever possible. POP and IMAP traffic can be encrypted with SSL if the remote site supports it, and any traffic can be tunneled over SSH if users have a remote account to which they can connect. Finally, learn as much as possible about wireless technology and security, and consider running monitoring programs. Many wireless security programs are available, but staying on the free software side of the fence, AirSnort (airsnort.shmoo.com), Kismet (www.kismetwireless.net) and Snort (www.snort.org) are good places to begin.

802.11a, b or g?

802.11a is nowhere near as widespread as 802.11b, and 802.11g boasts the same speeds plus backward compatibility. 802.11a also is extremely short-range.

At the time of this writing, existing 802.11g hardware on the market was released before the IEEE specification was completed and may not be compatible with other or future 802.11g implementations. Many reviewers have found backward-compatibility problems. Future 802.11g implementations hopefully will have these issues resolved. It may be more cost effective to set up a hot spot using 802.11b now and upgrade in the future.

Sam Leffler recently has developed drivers for Atheros 802.11a and 802.11g cards. However, the drivers require a proprietary hardware abstraction layer to restrict the frequencies and power the card uses.

Mike Kershaw currently works for a medium-sized college between Albany and New York City. He became interested in wireless in 2001 and hasn't looked back since. He also is the author of the Kismet wireless security program.

Linux Makes Wi-Fi Happen in New York City

Doc Searls

Issue #113, September 2003

Community groups, startup companies and even the phone company are using Linux to make New York City into one big happy wireless network hot spot. How is your town doing?

Public wireless networks are hacker community outreach. For hackers, it's a way to bring broadband Internet to public spaces. For users in streets and parks, it's a more civilized public life. Unwired from providers, public Wi-Fi makes the Net a gift—a civic grace akin to parks, sidewalks, boulevards and libraries.

In May 2003, when the FCC continued deregulating ownership of what we used to call the “public” air waves, the agency made a big deal about “saving” what was left of “free over-the-air” broadcasting. The Internet, however, needs no deregulation—or regulation—to make it free over the air. All it needs is generous technologists, citizens and civic organizations. That's what we have in New York City today. And, their work is remarkable to behold.

New York Unwired

Some public Wi-Fi efforts are largely municipal. That's the case with Long Beach, California, which provides a large public “hot zone” in its downtown and another at its airport. Other efforts are driven by technically savvy volunteers, such as in Austin, London, Perth, Seattle, San Francisco and many other places. Companies also are doing their part. In Asheville, North Carolina, Natural Communications offers a public hot spot called the BeamPost. New York, however, is different breed. It's all the above.

Although New York ranks 27th on Intel's list of “most unwired” cities (Portland is first), it is perhaps the best example of a working consensus between hackers, businesses, government and nonprofits about the need for free public Wi-Fi. This consensus is what gave rise to NYCwireless, a self-described “loose collection of interested minds”. NYCwireless has two missions: to provide free public wireless Internet access and to provide a forum for wireless technology development.

The founders of NYCwireless are Anthony Townsend and Terry Schmidt, partners in Emenity, the company that has been building out the new NYCwireless infrastructure in New York. Both NYCwireless and Emenity are products of public and private symbiosis—same with its customers, which include publicly funded neighborhood associations created for the purpose, among other things, of building out infrastructural improvements, such as public Wi-Fi in the parks.

In May, New York's City Council issued a staff report that recommends a restructuring of the city's fractured broadband procurement methods, a new fiber/wireless metropolitan area network (MAN) and public Wi-Fi networks. As an example of the latter, it says a potential Prospect Park Wi-Fi network would cost $192,000 to create but little to maintain. That report opens with special thanks to Anthony Townsend, who is also a Research Scientist at NYU's Taub Urban Research Center, where he has produced a pile of wise and seminal papers about the growth of the Internet in urban settings. Terry Schmidt's job is turning Anthony's vision into reality. Terry is Emenity's CTO and the hacker behind Pebble Linux, the stripped-down Debian used in NYCwireless access points.

Pebble Linux: Debian for Wi-Fi

An Inexpensive Single-Board Computer from Soekris Engineering

If you want to set up a public access point (AP), however, you'll need something that gives you a high level of functionality and control, in a compact and reliable system. Something, of course, that runs on a form of Linux. Pebble Linux is a tiny Debian-based Linux that's the basis of a load-and-go, fully-featured, (relatively) easy-to-customize, no-moving-parts AP. Created by Terry Schmidt of NYCwireless and maintained by a pack of user/hackers, Pebble is Debian, stripped down to a size and shape that fits cozily on a 128MB Flash card. Because it's Debian, adding and removing packages is relatively easy. Here's how Terry Schmidt says he did it:

> I stripped out all the documentation, all the Perl stuff, a lot of the binaries, all the packages I didn't think were necessary. I got it down to 44MB. I wanted the functionality of a real distro like Debian in a size that would fit in a CompactFlash in something like a Soekris box. I could do `apt-get install apache` and bang, we'd have Apache. So the full package manager is there, with all the ease and functionality you'd expect.

Terry's README (www.nycwireless.net/pebble/pebble.README) file adds:

> Its biggest advantage is that it mounts read-only. You don't have to worry as much about wearing down the CompactFlash, and you don't have to worry about doing proper shutdowns. Unplug and plug in as much as you want.

There are two packages in a base Pebble image that aren't installed as Debian packages:

- HostAP: a driver for Prism-based 802.11 cards that provides the best support for running an AP, as opposed to a client. This is available as a Debian package, but Pebble uses the latest version from CVS, because it associates better with 802.11g clients.
- NoCatAuth: more details below.

Three optional packages also are available:

- “Pebble mesh” support, which allows multiple Pebble machines to form a transparent mesh. This means that a user can roam without changing IP addresses or losing network connectivity. AP mesh capability (which is, IMNSHO, unbelievably cool) is the most interesting add-on for people building *public* wireless networks. You can build an arbitrarily large wireless hot zone, and—best of all—the devices autoconfigure, so adding

or removing a node doesn't require modifying the configurations of the other nodes.
- Support for an ELAN SC520 watchdog timer. In particular, this addresses the built-in watchdog timer on the Soekris. This allows for automatic reboot in case of software glitches. It's particularly useful when the AP is mounted someplace that's hard to access (like in a kiosk in a public park), or the AP isn't actively monitored (like almost every AP). This, along with the read-only filesystem, makes a Pebble system close to zero-maintenance.
- Support for running as a bridging firewall.

Pebble runs well on a 486 processor or better and requires no more than 32MB of RAM and 128MB of “disk” storage. It probably will run on that old 486 in your closet, but for less than $300, you can buy the very cool and very tiny Soekris 4511-20 and a wireless card—and be up and running in no time. If you're hard-core, you can buy the Soekris with no power supply or case and build your AP for less than $250.

Pebble is designed to work out of the box with any Intersil Prism2 or Prism2.5-based 802.11b card, such as the Linksys WPC11, the D-Link DWL-650 or the Compaq WL100 and WL200. With some simple configuration, it should work with any Linux-supported 802.11b card.

When you're ready to get started with Pebble, see the project site. As an alternative to Pebble for the /truly/ minimal-minded, you might consider WISP-Dist (Wireless ISP Distribution—leaf.sourceforge.net). WISP is *incredibly* tiny; it fits on an 8MB Flash ROM and 16MB of RAM. It's not nearly as full-featured as Pebble (it's a really-vanilla AP) nor is it as easy to customize.

To set up your own public AP, all you need is an ISP that doesn't care if you share bandwidth, an AP, a target service area, a directional antenna and motivation.

Some ISPs, like Bway.net in New York, are happy to let you share the bandwidth for which you pay. Others, like Time-Warner Cable and AT&T Broadband, crack down on users sharing bandwidth. A local public Wi-Fi organization can help you locate an ISP with suitable terms of service or help you lobby your ISP to change their terms of service. Freenetworks.org can help you find your nearest public Wi-Fi group.

NYCwireless' mission is to target outdoor public spaces, such as parks. Placement means everything. As Doc and Britt discovered when trying to reach Tudor City's park from high in a building half a block away, distance is a problem. A highly directional antenna can help by concentrating energy in a narrow beam, but a nearby omnidirectional (omni) antenna outperforms a distant directional antenna nearly every time. Bryant Park is served from a number of points by a combination of omni and sector (directional) antennas on the tops of kiosk buildings. City Hall Park is much better served by a sector antenna on the store across the street. Verizon gets great curbside service from simple omni antennas on public phone booths.

Antennas are not commodities, but they don't have to be expensive, either. And, sometimes simply putting an AP in a window does the job. Ben Hammersly did exactlly that for Kynance Mews in London and served a whole street, including two outdoor cafes. Motivation, of course, is up to you.

—Kurt Starsinic

Going Signal Fishing

Wi-Fi range is low on purpose. It operates on a tiny wedge of unlicensed microwave frequencies divided into 14 channels between 2.412 and 2.484GHz. Here in the US, we use only 1–11. In Europe they use 1–13, except for France, where they use 10–13. Japan runs from 1–14. The default transmission power of most access points (also known as APs, WAPs and base stations) is 30mw, about one-tenth the power of a cell phone but on a higher frequency, where the energy attenuates more rapidly with distance through air and has trouble penetrating many objects, including tinted windows and leaves full of microwave-absorbing water.

Wi-Fi range tends to run less than the average cordless phone, which sports a more powerful signal. With such a handy service delivered by such a short-range signal, it's only natural to find the best signal where the population is both dense and conveniently arranged, such as New York City, where people live and work on top of one another.

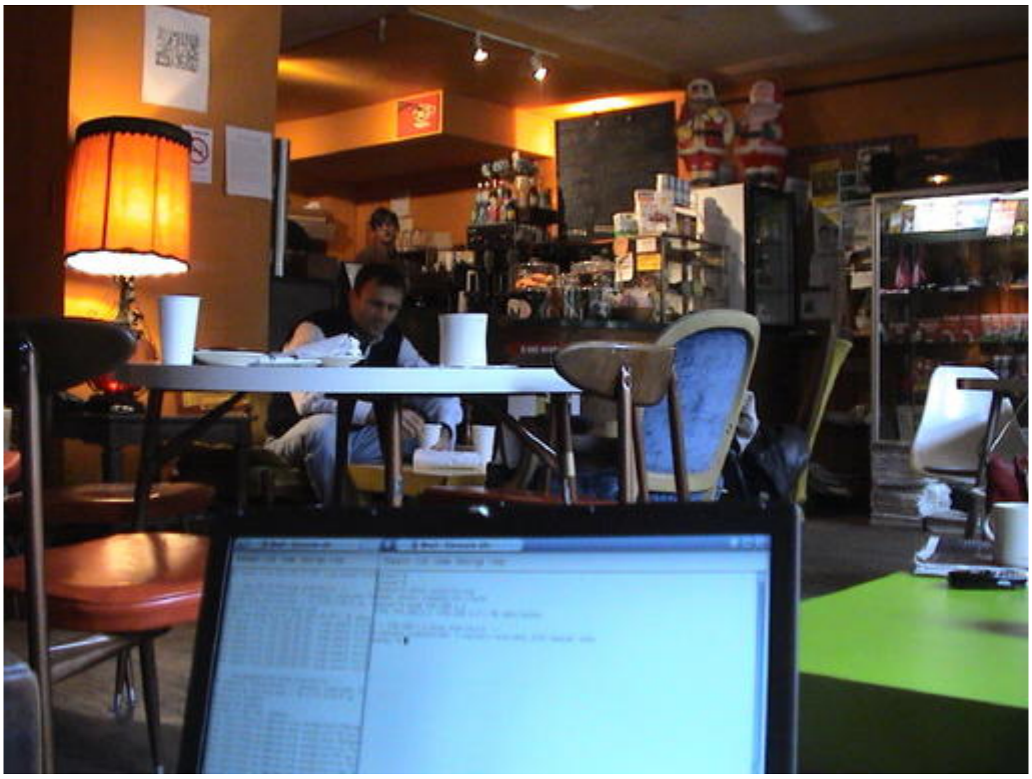
I ran six wardriving sessions, which included nine taxi rides, nearly all in Manhattan. The last session ended with a highway ride to LaGuardia Airport through signal-free parts of Queens. Each session recorded basic data about every detected signal, including ESSID (Extended Service Set Identifier—the access point's name). On the nine rides, I logged a total of 1,548 open access points.

On city streets in Manhattan, I found there was nearly always an access point in range. And I'm sure the numbers above would have been much higher if I'd had an antenna outside each taxi instead of on my lap in the back seat. Although lots of commercial hot spots exist, the vast majority appear to belong to individuals. “Linksys” is the default ESSID for the company's popular inexpensive access points.

The willingness of individuals to share bandwidth is amazing. Although the number of wide-open APs was lower than the WEP numbers above suggest, because quite a few were password-protected, plenty of usable signals still were available. More than once I was able to pick up and send e-mail while a cab was stopped at a light.

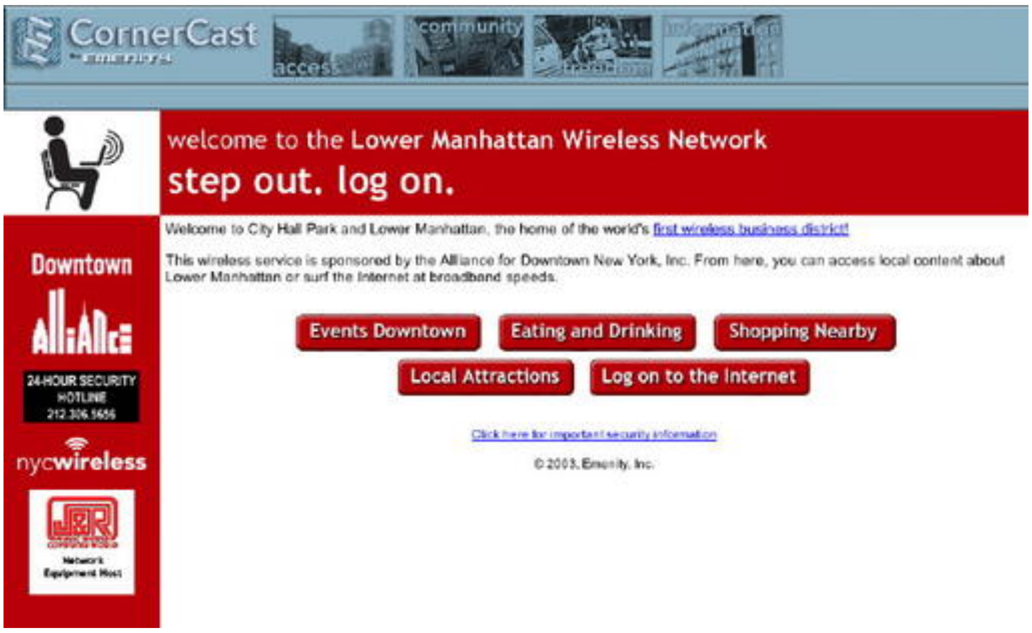
The ideal way to go signal fishing with a Linux or BSD laptop is with Kismet, a wireless network sniffer so full-featured it even does neat stuff with GPS, precisely associating signals with locations [see page XX].

Fellow Travelers



Alt.coffee from behind a Linux Laptop

Kurt Starsinic, Guide to the Wireless Scene



welcome to the Lower Manhattan Wireless Network

step out. log on.

Welcome to City Hall Park and Lower Manhattan, the home of the world's first wireless business district!

This wireless service is sponsored by the Alliance for Downtown New York, Inc. From here, you can access local content about Lower Manhattan or surf the Internet at broadband speeds.

Click here for important security information

© 2003, Emenity, Inc.

The City Hall Park welcome screen shows nearby attractions.

Right before the trip, I mentioned in a SuitWatch newsletter that I'd be coming to New York to check out the Wi-Fi situation and that I could use some local help. The first reply came from Kurt Starsinic who quickly became my Wi-Fi docent for warwalking and wardriving through lower Manhattan. I hooked up with Kurt at Alt.Coffee on Avenue A across from Tompkins Square Park. Alt.Coffee is both a comfortably run-down coffee house and a reliquary for dead computers. Kaypros, ARCnet hubs, early-vintage PCs and other antiques are scattered on tables and piled up in corners—worth a visit.

As it happened, the APs for both Alt.Coffee and NYCwireless were down while we were there, but when we walked around Tompkins Square Park, we still found at least one home node with an open and usable connection. And yes, *of course* we used it.

Our next stop was City Hall Park where the NYCwireless signal is clear and strong. There I was able to sample NYCwireless' local fare while Kurt briefed me on technology issues and both of us waited for my old friend Stephen Lewis to show up.

Steve, who carries US and Dutch passports, is a European telco industry veteran who was highly curious about what was happening with Wi-Fi in his home town. Walking around Steve's old haunts in the Lower East Side, we were impressed by the density of Wi-Fi, from both public and private sources: Verizon public phones, McDonalds restaurants and Starbucks coffee shops, in addition to private homes.

The ability to get on the Web almost anywhere in an outdoor urban setting was especially impressive to Steve, a two-time Fulbright Scholar with a hearty appetite for information. As a result, he began to develop ambitious plans to carry the lessons of New York neighborhood Wi-Fi (including Linux technologies) to Bulgaria, where he has lived for much of the last decade.

One of the most interesting figures in the New York Wi-Fi movement is Drazen Pantic. A former mathematics professor at the University of Belgrade, Drazen ran the Internet service of B92, a radio station that was a thorn in the side of the Milosevic regime. After the station's transmitter was shut down mysteriously, Drazen made sure the station's news and information continued to come out on the station's Web site and through streams that were picked up and rebroadcast in the UK, Netherlands, the US, and, most significantly, Yugoslavia. Stations there picked up and rebroadcast the analog signals relayed by satellite from the Netherlands. As a result, B92 quickly became the primary source of news from, and about, Yugoslavia and the conflicts there. Hearing him tell the story of his life, it was clear that Drazen was a hero of several revolutions at once.

Drazen is also involved with Dyne.org, a Vienna-based group of free software hackers devoted to producing GPL'd freeware for real-time video processing, media streaming and other cool stuff. The coolest of Dyne's tools, Drazen explained, is HasciiCam, a neat little hack that captures video from a TV card, renders it into ASCII and outputs it in a variety of ways—as HTML with a refresh tag, as a live ASCII window or as a simple text file.

On the downstream side, Drazen is excited about both the Dyne:bolic Linux distro and MPEG4IP. Dyne:bolic is a multimedia-oriented distro that can run from a CD and recognize sound, video, TV, network cards and other peripherals. MPEG4IP is a streaming package that obviates the need to use proprietary streaming systems. Drazen says, “After downloading Dynebolic, you can burn a CD, boot in to Linux and stream high quality MPEG4.”

Drazen believes all these open-source efforts will finish liberating audio and video authoring, production and distribution from the corporate chokeholders that still hold our ambitions and imaginations in check. He sees Linux as the public OS platform and Wi-Fi as the public network commons. Together they'll support a new form of (literally) public TV and radio. Between Wi-Fi, HasciiCam, digital camcorders, cheap hardware, free software, Dyne:bolic and MPEG4IP, Drazen expects the threshold of reporting and broadcasting to drop about as far as it can go. When it gets there, watch out.

Luggables and Wearables

On Sunday I returned to Alt.Coffee to meet with Ahmi Wolf. He and Mark Argo are the creators of the Bass-Station, a turn-of-the-80s suitcase-size ghetto blaster that also happens to be a digital juke box and a Wi-Fi hot spot. Ahmi and Mark removed the radio and cassette components of this funky old thing and replaced them with a variety of modern portable Wi-Fi goods: Via mini-ITX motherboard, wireless interface card hooked to an antenna, Debian (Woody) loaded onto a CompactFlash card, and a 120GB hard drive. They left the amplifier and speakers and hooked them up to the board's audio output.

Boom-Box: the Guts behind the Glory

The Bass-Station's guts consist of a mini-ITX motherboard using an 800MHz processor (www.viatech.com), 256MB of RAM, a Prism-based PCI wireless interface card and a 120GB IDE hard drive. It runs Debian Linux (Woody 3.01), which uses the HostAP drivers (hostap.epitest.fi) to put the Wi-Fi card into an access-point mode so the machine appears as a managed node as opposed to an ad hoc client mode. We have a DHCP server for dishing out IP addresses to wireless clients. It is the standard ISC DHCP server that comes with almost all Linux distros, configured in the standard manner. The Bass-Station also runs a DNS server configured to serve as the top-most authoritative DNS server on the Net—the so-called dot (.) domain, which resolves all domains to the IP address of the Bass-Station. This way any URL a user points to takes that user to the Bass-Station's Web server.

There also are alternative ways to take users to a specific Web page. Using active portal software like NoCat (www.nocat.net) will do this, but the purpose of such software is to be a portal or entryway to a network. The problem with this software is that it tries to resolve the intended URL *before* it shows you the portal page. Because the Bass-Station is not connected to or associated with any other network, there is no means to resolve an external intended address, so the program tries to resolve and resolve and doesn't show anything. So here's the hack-around. Start with a clean installation of the DNS system Bind (we used version 9). Then, in /etc/bind/named.conf change the zone "." entry to the following:

```
zone "." {
type master;
file "/etc/bind/db.root";
notify no;
};
```

Then, replace the default db,root file (back it up first) with a file that contains only the following:

```
;-----------Beginning of file------------;
$TTL 604800
@ IN SOA . root.localhost. (
1 ; Serial
604800 ; Refresh
86400 ; Retry
2419200 ; Expire
604800 ) ; Negative Cache TTL
@ IN NS .
* IN A 192.168.23.1
;-----------End of File----------------;
```

Replace the IP address 192.168.23.1 with the IP address to which you want all domains to be resolved.

Data lives in a MySQL database and is displayed through the Apache Web server. Together these provide the interface to all functionalities. For now these include:

- Uploading files (we're using HTTP, so all interaction can happen through a browser).
- Browsing/viewing/downloading of files located on the Bass-Station.
- Controlling the playback of music from the stereo.

We use mpg123 for media file playback. I also wrote the back-end program for control of mpg123 and interaction with our databases. C++ source should be available on our site some time in the future.

—Ahmi Wolf

The result is the social and aesthetic opposite of an iPod: a big ugly stereo that's also a Linux-based Wi-Fi access point, plus a juke box with a big-ass hard drive. The idea was to create a juke box for all kinds of convivial settings—from parties in parks to hang-outs on college campuses. Everyone connected by Wi-Fi to the Bass-Station gets to contribute music and play disk jockey, so it rewards cooperation as well.

The Bass-Station belongs to the neighborhood extranet—not to an individual and not to the whole world. Ahmi explains:

> The Bass-Station is not connected to or a part of another network. It creates it's own network that exists only within the range of the Bass-Station itself. On one hand, the range of Wi-Fi is limited, but the limited range makes it special. Users of the network are all in close proximity to each other, making them members of a community—be it a stable, persistent community or a spontaneous and mobile one like the Bass-Station's network.

Ahmi's Bulgarian friend Milena Iossifova, a fellow student at NYU's Interactive Telecommunications Program, has a way-cool Wi-Fi creation of her own called Wi-Fisense, which she calls “a wearable scanner for wireless networks”. It's a handbag with 64 LEDs in three different colors, each turned on by Wi-Fi activity on a different channel.

The optimism and energy of all this reminded me of what Silicon Valley felt like back in the 80s and 90s but without the corrupting context of other people's money. Ahmi, Milena and Dave already have produced enabling goods in this new culture.

Building a New Infrastructure, One Pebble at a Time

I met with Terry Schmidt at Emenity's offices near Wall Street, where he briefed me on the challenges of deploying public Wi-Fi in New York's peculiar urban settings. The first big project for both NYCwireless and Emenity was Bryant Park, which shares a midtown block with the New York's Public Library. Terry explains:

> We overbuilt that one with two omni antennas, one sector antenna and two point-to-point links within the park itself. But it was a big success, so it became clear that there was a need for free wireless networks. A volunteer organization like NYCwireless can't easily do service level agreements and stuff like that, so that's what we provide with Emenity.

Terry sees Emenity as a midway organization between the purely voluntary and the purely self-reliant. Bryant Park, for example, originally was built by NYCwireless, then maintained by Emenity and now is run entirely in-house by the park itself.

Emenity's biggest customer is the Downtown Alliance, a business improvement district (BID) organized to “create and promote a safe, clean, live-work, totally wired community”. BIDs throughout the city are supported by a small additional local sales tax. Improvements to Bryant Park—which are nothing less than spectacular, considering the no-mans-land it used to be—are examples of a BID at work. Because the alliance serves landowners, it also can approach them with requests to use their roofs or windows for wireless antennas aimed down at public spaces.

A Rooftop Antenna Serves City Hall Park

Location, Location, Location

At City Hall Park, the rooftop across the street at J&R Music and Computer World proved to be the ideal access point location. A square white sector antenna with a beam width of about 40°, angles down at the park and provides a signal footprint that serves the park itself and little else. At the far edge of the park by City Hall it fades away. A fairly precise footprint also graciously yields to other access points at the local Starbucks, City Hall, the Woolworth Building and elsewhere in the neighborhood.

Terry Schmidt says NYCwireless encourages local citizens operating free access points to label them “NYCwireless” and register with NYCwireless so they appear on the organization's node list. End user licensing runs the gamut from locked-down to free. Time-Warner, for example, aggressively denies users the right to share bandwidth. At the other extreme, Verizon sells Wi-Fi access points to its DSL customers.

The black bump on this pay phone is a Wi-Fi antenna.

Verizon Gets It

Verizon, which has thousands of phone booths on the streets of New York, has seen the same writing on the own wall, and come up with a brilliant plan: turn phone booths into access points. The first 150 were fired up on May 13, and the company has plans to add the service to 500 or more throughout the city and beyond.

At the time of this writing, the service is available and free, to Verizon business and residential DSL customers only. But there's nothing in the deployment that prevents the company from opening up to other customers or from opening up completely—it was designed that way. In fact, it was designed to be as easily deployable and modifiable as possible, which is why the company made use of Linux and open-source tools. Sean Byrnes, an architect with Verizon, explained it this way:

> What Linux let us do was deploy extremely quickly. So, rather than setting up large servers in one of our data centers, we were able to create Linux clusters and

> build initial versions that supported the hot spot service extremely quickly, using a wide variety of open-source software—much more quickly than if we had been waiting for licenses, etc. We couldn't have moved it into the data center if Linux didn't allow us to develop with platform independence and with open-source technologies that are implemented across multiple operating systems. We're working to have Linux qualified for the data centers, but it isn't there yet.

When I said it sounded to me like Verizon was an example of a company that found it easier to roll their own solutions than depend on vendors for help, Sean Byrnes replied, “That would be an understatement, actually.” He explained:

> If you think of very large companies, more often than not, when you're rolling out a new service or application, the argument can be made that the majority of it is glue. Because you already have so many systems and applications out there you have to glue them together somehow, so you're forced to be agile. It's never a question of being able to buy a package from a vendor and use it on day one.

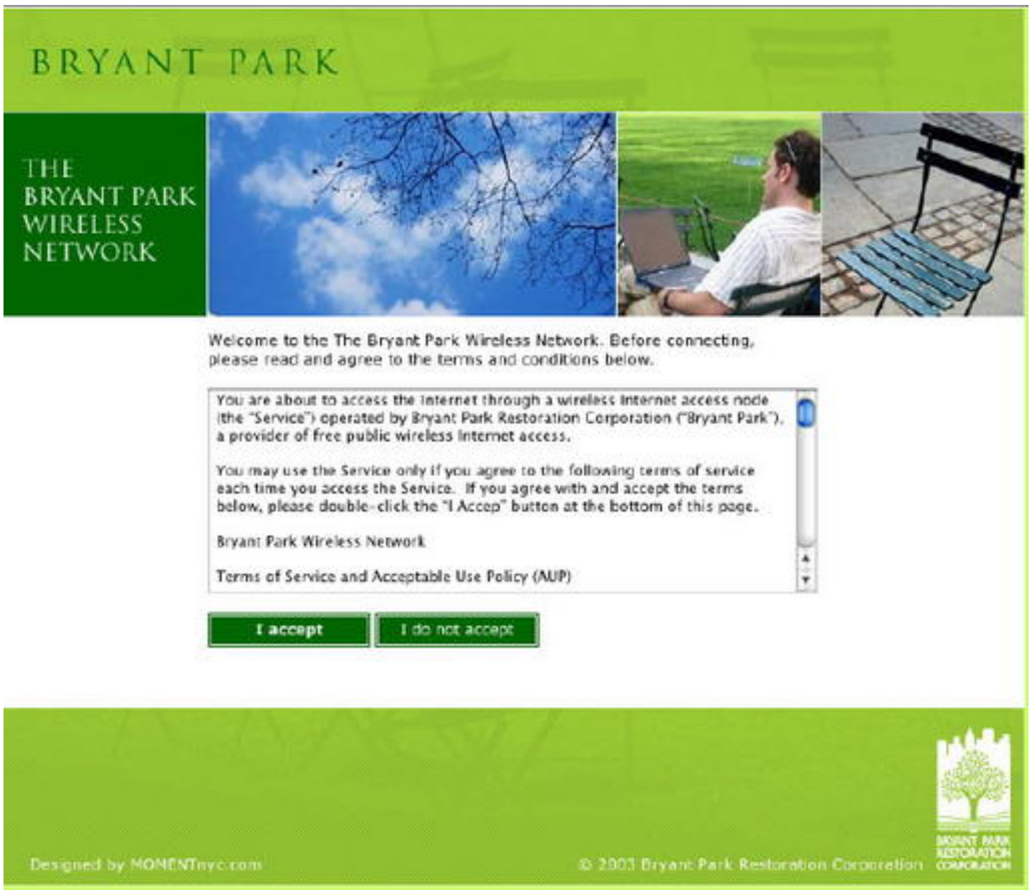
With that many managed access points on the street, the Verizon people have been gaining some valuable experience with Wi-Fi in the real world. Terry Schmidt isn't optimistic about nonfree business models for Wi-Fi. He says, “We don't think that a lot of the for-pay wireless stuff has a sustainable business model. Companies like T-Mobile, with all those Starbucks locations, are hemorraging money, and almost nobody's using them.”

Meanwhile, plenty of people are taking advantage of free Wi-Fi in places like Bryant Park and Alt.Coffee. “Free wireless is good for business”, Terry says.

> That's the model. Local business owner says, “I'm going to make my business and my surrounding market more valuable by providing free wireless. It's an attractive thing to do. It enhances the environment and attracts customers.”

Does Verizon's service, free for existing customers, serve as a conditional flower box? I believe so. Verizon is the incumbent local phone company in New York. It has a lot of home and business DSL customers. Flower boxes that appear magically for those customers are a nice bonus to existing service. It's a way for Verizon to say “Take that laptop out of here. Go sit in a cafe somewhere”.

This kiosk at Bryant Park hosts an access point.


BRYANT PARK
THE BRYANT PARK WIRELESS NETWORK
I accept
I do not accept
BRYANT PARK RESTORATION CORPORATION

Welcome to the The Bryant Park Wireless Network. Before connecting, please read and agree to the terms and conditions below.

You are about to access the Internet through a wireless Internet access node (the "Service") operated by Bryant Park Restoration Corporation ("Bryant Park"), a provider of free public wireless Internet access.

You may use the Service only if you agree to the following terms of service each time you access the Service. If you agree with and accept the terms below, please double-click the "I Accep" button at the bottom of this page.

Bryant Park Wireless Network

Terms of Service and Acceptable Use Policy (AUP)

Designed by MOMENTnyc.com

© 2003 Bryant Park Restoration Corporation

The Bryant Park Welcome Page

Enlarging Civic Life

Wi-Fi adds a new and practical feature to civic life. For two decades, most personal computing happened indoors, attached to printers, networks, servers and phone lines. If we used our laptops outdoors, it was usually in the same disconnected way we still use them on airplanes. With public Wi-Fi, we bring the networked knowledge of the world out into the open air, and that changes things.

For all the years I used to visit the New York Public Library, I completely ignored the wasteland that was Bryant Park. This last trip was my first exposure to Bryant Park, because it was completely re-done in the fashion of the great parks of Europe's cultural capitals. With its lawns, fountains, shaded pavilions and chairs scattered on sidewalks outside restaurants with open doors, it seemed to me the height of civilization. It also made me love civilization and the graces that increase it. That's saying a lot, too. It is the public places that civilize our cities. Perhaps public Wi-Fi will civilize the Net as well.

Resources

New York City Wireless Organizations

Downtown Alliance: www.downtownny.com

Emenity: emenity.com

New York's City Council Staff Report Network NYC: Building the Broadband City: www.council.nyc.ny.us/pdf_files/reports/broadbandcity.pdf

NYCwireless: www.nycwireless.net

NYU's Interactive Telecommunications Program: www.itp.nyu.edu

Wireless in Other Cities

“Antenna to the East: Linux and Wi-Fi in Sofia, Bulgaria”: www.linuxjournal.com/article/6954

Asheville, North Carolina, Beampost: www.blaserco.com/blogs/2003/02/20.html#a95

Austin, Texas: www.austinwireless.net/cgi-bin/index.cgi

Intel's list of “Most Unwired” Cities: www.intel.com/products/mobiletechnology/unwiredcities.htm

London: www.consume.net

Long Beach, California: www.longbeachportals.com

Paris, France: www.iht.com/articles/95233.html

Perth's WAfreenet: www.nodedb.com/australia/wa/perth/?

Portland, Oregon: www.personaltelco.net/index.cgi/PersonalTelco

San Francisco, California: www.bawug.org

Seaside, California: www.ezgoal.com/hotspots/wireless/f.asp?fid=57748

Seattle, Washington: www.seattlewireless.net

Winston-Salem, North Carolina: www.ezgoal.com/hotspots/wireless/f.asp?fid=65372

Free Software Projects

Bass-Station: bass-station.net

Dyne:bolic Linux: dynebolic.org

Dyne.org: dyne.org

HasciiCam: ascii.dyne.org

Kismet: www.kismetwireless.net

MPEG4IP: mpeg4ip.sourceforge.net

NoCatAuth: see p. 52 of this issue

Open Source Streaming Alliance: www.streamingalliance.org

Pebble Linux: www.nycwireless.net/pebble

WiFisense: wifisense.com

Products

Lindows MobilePC: info.lindows.com/mobilepc/mobilepc.htm

Media Box: www.ituner.com/products.htm

Soekris Engineering: www.soekris.com

Wireless Broadcast Public Wi-Fi, Network 2 Cable Network: open4all.info/laika

Miscellaneous

Alt.Coffee: www.altdotcoffee.com

EFF list of wireless-friendly ISPs: www.eff.org/Infra/Wireless_cellular_radio/wireless_friendly_isp_list.html#list

"It's All about Height": www.linuxjournal.com/article/6955

Warchalking: www.warchalking.org

Wardriving: www.personaltelco.net/index.cgi/WarDriving

Warwalking: www.personaltelco.net/index.cgi/WarWalking

Doc Searls is senior editor of *Linux Journal*.

Scripting for X Productivity

Marco Fioretti

Issue #113, September 2003

Bend X to your will with scripting tools, keyboard customizations and dialog boxes.

Linux offers graphical user interfaces (GUIs) with high resolution, multiple windows and menus. But with a GUI, human/computer interaction is the real speed bottleneck: as long as doing something requires ten mouse clicks in ten different points, it takes the same amount of time, regardless of the hardware. At the command line, the traditional UNIX answer is the toolbox philosophy. You can automate anything with a script and connect small specialized programs with pipes. A third solution combines the two approaches and is easy and powerful enough to suit most needs. Any stock X environment, regardless of the window manager or desktop environment, can become much faster and more flexible than it usually is.

Making Your Own Mouse and Keyboard

We have more fingers than hands and 101 keys to one mouse. Although drawing and similar tasks are quicker with mice or tablets, often our fingers must stay on the keyboard anyway. If a command or choice requires one single user action, pressing a key or two is much faster than clicking mouse buttons, not to mention RSI issues. Also, when we do touch the mouse, it should do what we need immediately.

Keys and mouse buttons can be shuffled around or mapped to specific meanings (accented vowels) or GUI actions (maximize window) using **xmodmap**. One of its beneficiaries has been left-handed people, who use it to reverse the order of the mouse buttons. Add this line in .xinitrc, or put the quoted part in a .xmodmaprc file:

```
xmodmap -e "pointer = 3 2 1"
```

Lately, xmodmap has been used to make mouse wheels work correctly under Linux (see Resources). It also can make wheels work faster by binding your most frequent actions to those extra keys available on the latest keyboards. For example, the magic line for an IntelliMouse Explorer is:

```
xmodmap -e "pointer = 1 2 3 6 7 4 5"
```

When speaking about keyboards, modifier means a key that changes the effect or state of other keys. The standard modifiers are Shift, Ctrl, Lock, Alt and five more simply called modN, with N=1,2..5. xmodmap can assign these meanings to any physical keys. The most frequent case is probably the swapping of the Ctrl and Caps Lock keys, described in the man page. On the same note, the meaning of the Windows key could be changed to mod4 in this way, assuming its keycode is 115:

```
xmodmap -e 'keycode 115 = Alt_R Meta_R'
```

Keys can be programmed to start applications too, but this is specific to window managers. In Blackbox 0.65, for example, keystrokes are mapped to programs with the bbkeys utility. A line like the following in $HOME/.bbkeysrc:

```
KeyToGrab(F1),WithModifier(None),WithAction(ExecCommand),\
DoThis(xterm -geometry 80x25 -e mutt)
```

would allow you to start mutt in an 80 × 25 xterm window by simply pressing F1. Many other window managers allow such bindings; check their documentation.

Point-and-Click Scripting

As already mentioned, any shell script can be given windows to enable communication with the user, and any graphical client (and X itself) can receive direct input from text programs. The first case happens when a shell script needs user interaction, but it isn't possible; say the user would rather die than type or there isn't a keyboard (Internet kiosks). The second scenario includes users who need to pass some unpredictable text to an X client from console programs without manual cut-and-paste capabilities. This *does* happen in real life: mutt and lynx may be all we need to read e-mail and surf the Net, but what if some e-mail or Web page says “Check out this movie preview”, and you want to start Mozilla on that page with one click.

Giving Windows to Shell Scripts

Starting an X window from a script has been possible for at least ten years, with tools like the now defunct Xscript. A modern solution is Xdialog, a GTK+-based widget generator. The script in Listing 1 shows an almost real-life example; it lets the user choose the best ISP, the account to be used and where to log the connection report. It then starts a traditional pppd/chat script (called netconn in the example), passing to it all the parameters collected graphically. Let's examine the GUI side in detail.

Listing 1. GUI Front End for an Internet Connection Script

```
#! /bin/bash

/bin/rm -f /tmp/netsettings

echo -n "PROVIDER=" > /tmp/netsettings
Xdialog --menubox "ISP selection menu" 20 40 5 \
"ISP_1"  "Lowest price in business hours"  \
"ISP_2" "More economic on weekends"       \
"ISP_3"     "Fastest ftp server" 2>> /tmp/netsettings

RETVAL=$?
# add control code here

echo -n "ACCOUNT=" >> /tmp/netsettings
Xdialog --inputbox "Enter account name"  10 25 \
"son" 2>> /tmp/netsettings

echo -n "LOGFILE=" >> /tmp/netsettings
Xdialog -fselect /tmp 20 80  2>> /tmp/netsettings

source /tmp/netsettings
netconn $PROVIDER $ACCOUNT $LOGFILE
```

The script starts removing the old version of the netsettings file, which stores all the user choices. For each variable needed by the netconn script, an assignment line in bash syntax is written to /tmp/netsettings. The left part simply is echoed without newline (`echo -n`). The second part, the actual value chosen by the user, is captured by Xdialog.

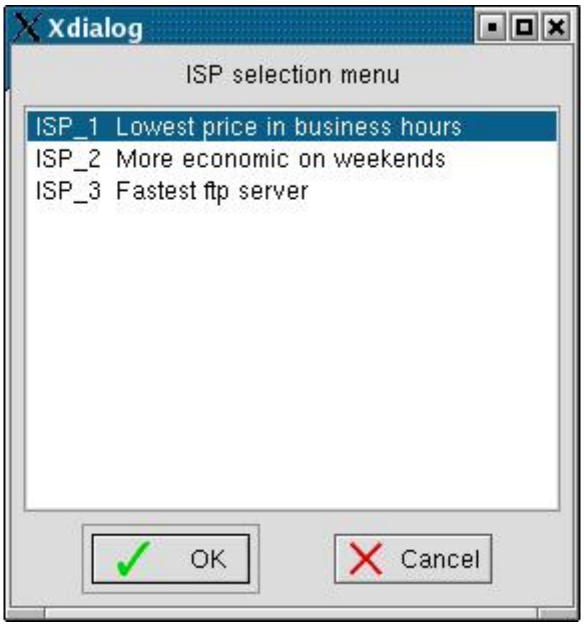

Xdialog
ISP selection menu
ISP_1 Lowest price in business hours
ISP_2 More economic on weekends
ISP_3 Fastest ftp server
OK
Cancel

Figure 1. Invoking Xdialog

The first invocation (see Figure 1) allows the user to choose the best ISP. We also added some explanatory text (ISP selection menu) and specified the height (20) and width (40) of the window, as well as the height of each entry (5). In the case of Figure 1, after the user presses the OK button, /tmp/netsettings will contain this one line:

```
PROVIDER=ISP_1
```

Error checking can and should happen by saving the exit code of *every* call to Xdialog in a variable (RETVAL in Listing 1) and checking it to know what the user really did. We omitted this code for brevity, but keep in mind that a RETVAL of 1 means “Cancel pressed”; 255 means the user closed the box, and 0 means a choice actually was made.

The second Xdialog command allows the user to type in an account name or to accept the default value (Figure 2). The last one (Figure 3), displays a file selection window in which one can type or select with the mouse the name and location of the current log file. At this point, all the user choices are in /tmp/netsettings, so we simply must source that file and launch the connection.

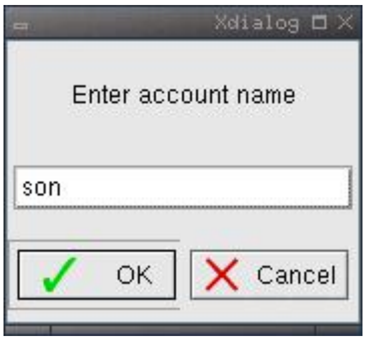


Figure 2. Entering the Account Name

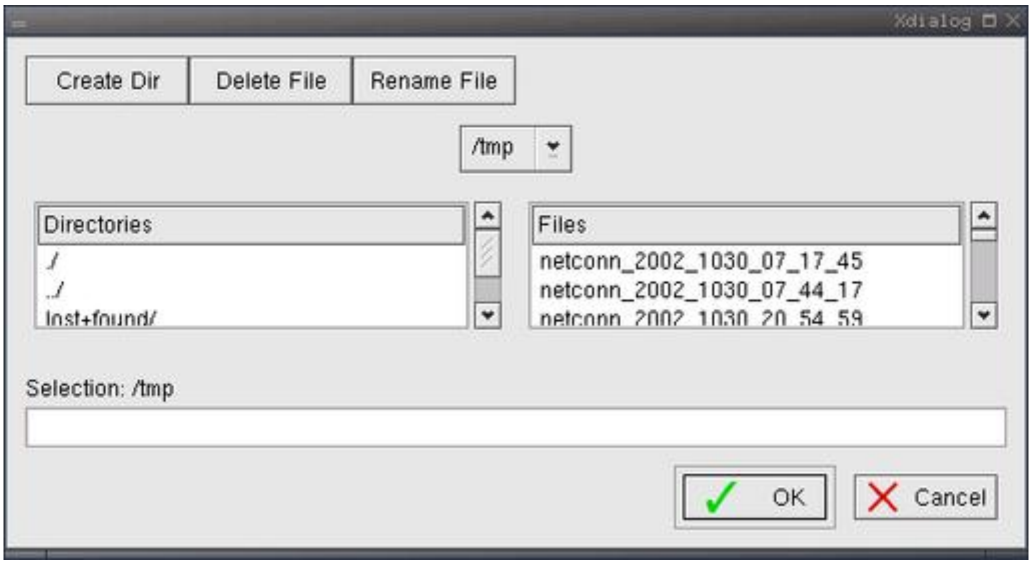


Figure 3. Finding the Log File

Xdialog offers many more types of input widgets, from radio buttons to range slides and calendars. Quite a few of these provide feedback to the user. The author uses the `--msgbox` option, for example, to pop up a window listing how many messages have been downloaded by fetchmail, sorted by account. Other possibilities include gauge and progress bars, info boxes with timeouts and file display windows. Consequently, Xdialog can be attached to scripts doing CD burning, audio and video playback, backups—you name it.

What if the same script must be used when X is not running? No problem; menus and boxes can be drawn in character terminals, too. Simply use the character-oriented equivalent, dialog.

Passing Text to X

Returning to the URL example, we can pass a URL to a browser without typing by using xclip, which captures the last mouse selection and echoes it. We put it together with the utility gnome-moz-remote, which either starts a new instance of Mozilla or opens the given URL with your already-running Mozilla. Put the following command in a script:

```
gnome-moz-remote --newwin "'xclip -o'" \
&> /dev/null &
```

Call it start_browser.sh, and bind it to a macro to run it from the application that must be enabled to launch external clients. In mutt, such a macro could be:

```
macro pager \cn "!start_browser.sh\n" 'open URL'
```

At this point, whenever we see a URL inside mutt, we'll simply highlight it with the mouse and press Ctrl-N. Inside the script, xclip echoes the selected text, and everything is as if we had started the browser by hand and then typed that text in the Location window.

xclip has a couple of distant cousins, xclipboard and xcutsel, that manage the X clipboard and cut buffer. These two programs are useful when it is necessary to see the clipboard content and to move the selection between applications that don't support them. Check the related man pages for more information.

Click Emulation

How do we make X believe we are moving the cursor with the mouse and using its buttons when we actually are pressing or releasing keys? The xbut program performs both these tasks. You can set up a simple configuration file to make keys work like mouse movements or clicks.

It is possible to go even farther with xwit, the X Window Interactive Tool, which directly *places* the X pointer to an absolute screen position. Besides warping the pointer, xwit moves and iconifies windows. To iconify the current xterm, sleep, and then pop it back up, use:

```
xwit -iconify; sleep 2; xwit -pop
```

This tool can be handy for long-running tasks; substitute the program name for the sleep command, and it will iconify the window, do the work and get back in your face when it's done.

Finally, a flexible X-automation tool is xte, part of the xautomation package. Use `xte -h` to see a list of the X events it supports. This example goes to a point 320 pixels from the left edge of the screen and 50 pixels down and then fakes a click there with button 1:

```
xte 'mousemove 320 50' 'mousedown 1'
'mouseup 1'
```

The mousedown and mouseup events are separate, so you can drag as well as click.

A Password-Aware Desktop

A fast and effective environment is one in which you can start any authorized process (whether in the background or interactive, text or GUI-based) on every other machine to which you have access as quickly, securely and transparently as possible. Of course, the right way to do it is through OpenSSH. But even if you're using RSA or DSA authentication for security—and to save having to remember a password per host—you have to type your SSH passphrase for every connection. Luckily, a valuable SSH partner, ssh-agent, remembers your SSH private key or keys and takes over the job of performing any authentication that requires the private key. Practically speaking, this means if you start X as a child process of ssh-agent, every local X client started later is able to use ssh-agent for authentication.

You can start your window manager as a child of ssh-agent in .xinitrc or .Xsession; put ssh-agent before the window manager. If you have `sawfish`, change it to `ssh-agent sawfish`, and everything that runs in your X session, including any SSH programs, will be able to use ssh-agent.

GNOME starts ssh-agent by default. To add it to KDE, find your KDE startup script and add ssh-agent to it as you would for a window manager.

The ssh-add command is the one that actually makes available identities known to the agent. You can make sure ssh-agent is running and knows your identity with `ssh-add -L`.

If the computer is left unattended, everybody walking by has immediate access, no questions asked, to all hosts where you can log in. Log out, or use `ssh-add -D` to delete your identity from the agent.

Finding Events

We explained how the numerical keycodes corresponding to extra keys can be remapped to mean other events, but how does one come to know them in the first place? The solution is the diagnostic tool xev, which opens an Event Tester window and reports in the original terminal everything that happens to that window. On the author's system, pressing the left Windows key returns (notice the keycode value and comments):

```
KeyRelease event, serial 23, synthetic NO, window
↪0x1000001,
 root 0x46, subw 0x0, time 1108438536, (175,176),
 root:(627,425), state 0x40, keycode 115
 ↪(keysym 0xffeb,
 Super_L), same_screen YES, XLookupString gives
 ↪0 characters:
 ""
```

Last but not least, screenshots are necessary to show off your shell script GUIs, aren't they? Even in this case, plain X and ImageMagick suffice, without needing fancier front ends installed. The images for this article all were grabbed and converted to PNG format with the following standard commands, all properly documented in their man pages:

```
xwd -out temp_image -frame
xwdtopnm temp_image  > fig1.pnm
convert fig1.pnm fig1.png
```

The first command dumps the window selected with the cursor, frame included, into temp_image, and the second and the third convert that file first to “portable anymap”, then to PNG format. It goes without saying that these three commands may be inserted easily in a shell script that asks, through Xdialog, which to grab (screen or window) and in which file to save the result.

Conclusion

For many users, the standard, full-blown desktop environments have either too many features, which slow the PC down, or too few to fulfill their specific needs. The tools and techniques described here can help users greatly improve their productivity and also can be a lifesaver whenever the same PC is shared by CLI and mouse addicts.

Resources

Remote Control of Mozilla: www.mozilla.org/unix/remote.html

ssh-agent: www.cvrti.utah.edu/~dustman/no-more-pw-ssh, www.arches.uga.edu/~pkeck/ssh, www.linuxgazette.com/issue67/nazario2.html and www.linuxgazette.com/issue64/dellomodarme.html

xautomation (includes xte): hoopajoo.net/projects/xautomation.html

xclip: people.debian.org/~kims/xclip

xclipboard: www.modperldev.com/portfolio/samples/perl/irc/xchat_clipboard.html and sandklef.com/software/xbut

xdialog: jan.netcomp.monash.edu.au/SW.html#XScript

xev: www.ditch.org/kbstick

xmodmap: www.deadman.org/X/xbuttons.html, koala.ilog.fr/colas/mouse-wheel-scroll, www.tldp.org/HOWTO/Keyboard-and-Console-HOWTO-15.html, www.sandklef.com//software/xikbd and www.fedu.uec.ac.jp/ZzzThai/xio

xwit: softcity.libero.it/debian/pool/main/x/xwit

Marco Fioretti is a hardware systems engineer interested in free software both as an EDA platform and (as the current leader of the RULE Project) as an efficient desktop. Marco lives with his family in Rome, Italy.

Archive Index Issue Table of Contents

Advanced search

A Beginner's Guide to Using pyGTK and Glade

Dave Aitel

Issue #113, September 2003

pyGTK and Glade allow anyone to create functional GUIs quickly and easily.

The beauty of pyGTK and Glade is they have opened up cross-platform, professional-quality GUI development to those of us who'd rather be doing other things but who still need a GUI on top of it all. Not only does pyGTK allow neophytes to create great GUIs, it also allows professionals to create flexible, dynamic and powerful user interfaces faster than ever before. If you've ever wanted to create a quick user interface that looks good without a lot of work, and you don't have any GUI experience, read on.

This article is the direct result of a learning process that occurred while programming Immunity CANVAS (www.immunitysec.com/CANVAS). Much of what was learned while developing the GUI from scratch was put in the pyGTK FAQ, located at www.async.com.br/faq/pygtk/index.py?req=index. Another URL you no doubt will be using a lot if you delve deeply into pyGTK is the documentation at www.gnome.org/~james/pygtk-docs. It is fair to say that for a small company, using pyGTK over other GUI development environments, such as native C, is a competitive advantage. Hopefully, after reading this article, everyone should be able to put together a GUI using Python, the easiest of all languages to learn.

As a metric, the CANVAS GUI was written from scratch, in about two weeks, with no prior knowledge of pyGTK. It then was ported from GTK v1 to GTK v2 (more on that later) in a day, and it is now deployed to both Microsoft Windows and Linux customers.

The Cross-Platform Nature of pyGTK

In a perfect world, you never would have to develop for anything but Linux running your favorite distribution. In the real world, you need to support several versions of Linux, Windows, UNIX or whatever else your customers need. Choosing a GUI toolkit depends on what is well supported on your customers' platforms. Nowadays, choosing Python as your development tool in any new endeavor is second nature if speed of development is more of a requirement than runtime speed. This combination leads you to choose from the following alternatives for Python GUI development: wxPython, Tkinter, pyGTK and Python/Qt.

Keeping in mind that I am not a professional GUI developer, here are my feelings on why one should chose pyGTK. wxPython has come a long way and offers attractive interfaces but is hard to use and get working, especially for a beginner. Not to mention, it requires both Linux and Windows users to download and install a large binary package. Qt, although free for Linux, requires a license to be distributed for Windows. This probably is prohibitive for many small companies who want to distribute on multiple platforms.

Tkinter is the first Python GUI development kit and is available with almost every Python distribution. It looks ugly, though, and requires you to embed Tk into your Python applications, which feels like going backward. For a beginner, you really want to split the GUI from the application as much as possible. That way, when you edit the GUI, you don't have to change a bunch of things in your application or integrate any changes into your application.

For these reasons alone, pyGTK might be your choice. It neatly splits the application from the GUI. Using libglade, the GUI itself is held as an XML file that you can continue to edit, save multiple versions of or whatever else you want, as it is not integrated with your application code. Furthermore, using Glade as a GUI builder allows you to create application interfaces quickly—so quickly that if multiple customers want multiple GUIs you could support them all easily.

Version Issues with GTK and pyGTK

Two main flavors of GTK are available in the wild, GTK versions 1 and 2. Therefore, at the start of a GUI-building project, you have to make some choices about what to develop and maintain. It is likely that Glade v1 came installed on your machine. You may have to download Glade v2 or install the development packages for GTK to compile the GTK v2 libglade. Believe me, it is worth the effort. GTK v2 offers several advantages, including a nicer overall look, installers for Windows with Python 2.2 and accessibility extensions that allow applications to be customized for blind users. In addition, version 2 comes installed on many of the latest distributions, although you still may need to install development RPMs or the latest pyGTK package.

GTK v2 and hence pyGTK v2 offer a few, slightly more complex widgets (Views). In the hands of a mighty GUI master, they result in awesome applications, but they really confuse beginners. However, a few code recipes mean you can treat them as you would their counterparts in GTK v1, once you learn how to use them.

As an example, after developing the entire GUI for CANVAS in GTK v1, I had to go back and redevelop it (which took exactly one day) in GTK v2. Support was lacking for GTK v1 in my customers' Linux boxes, but installing GTK v2 was easy enough. The main exception is Ximian Desktop, which makes pyGTK and GTK v1 easy to install. So, if your entire customer base is running that, you may want to stay with GTK v1. One thing to keep in mind though—a Python script is available for converting projects from Glade v1 to Glade v2, but not vice versa. So if you're going to do both, develop it first in Glade v1, convert it and then reconcile any differences.

An Introduction to Glade v2

The theory behind using Glade and libglade is it wastes time to create your GUI using code. Sitting down and telling the Python interpreter where each widget goes, what color it is and what the defaults are is a huge time sink. Anyone who's programmed in Tcl/Tk has spent days doing this. Not only that, but changing a GUI created with code can be a massive undertaking at times. With Glade and libglade, instead of creating code, you create XML files and code links to those files wherever a button or an entry box or an output text buffer is located.

To start, you need Glade v2 if you don't have it already. Even if you do, you may want the latest version of it. Downloading and installing Glade v2 should be easy enough once you have GTK v2 development packages (the -devel RPMs) installed. However, for most people new to GUI development, the starting window for Glade is intimidatingly blank.

To begin your application, click the Window Icon. Now, you should have a big blank window on your screen (Figure 1).

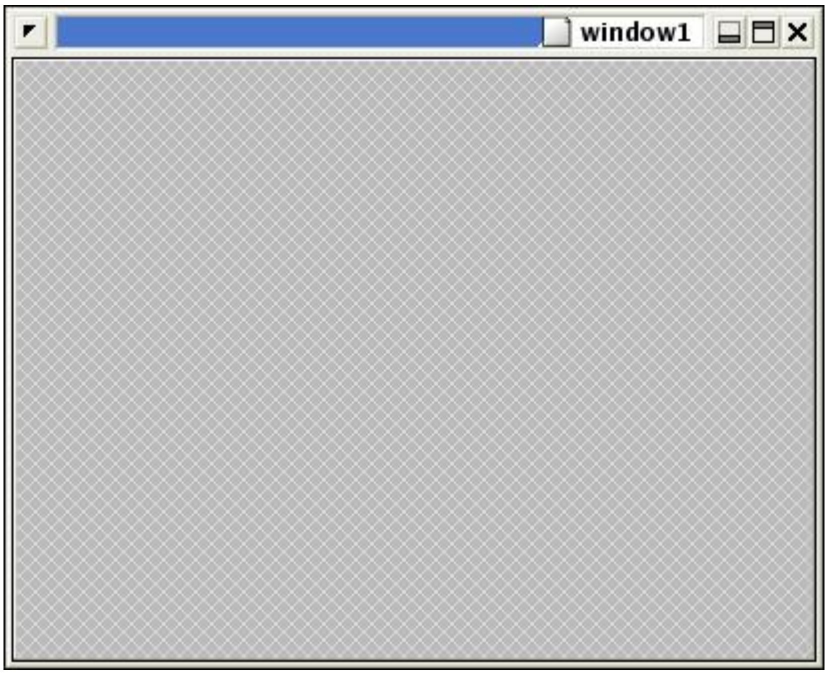


Figure 1. The cross-hatched area in the starting window is a place to put another widget.

The important thing to learn about GUI development is there are basically two types of objects: widgets, such as labels and entry boxes and other things you can see, and containers for those widgets. Most likely, you will use one of three kinds of containers, the vertical box, the horizontal box or the table. To create complex layouts, its easiest to nest these containers together in whatever order you need. For example, click on the horizontal box icon. Clicking on the hatched area in window1 inserts three more areas where you can add widgets. Your new window1 should look like Figure 2.

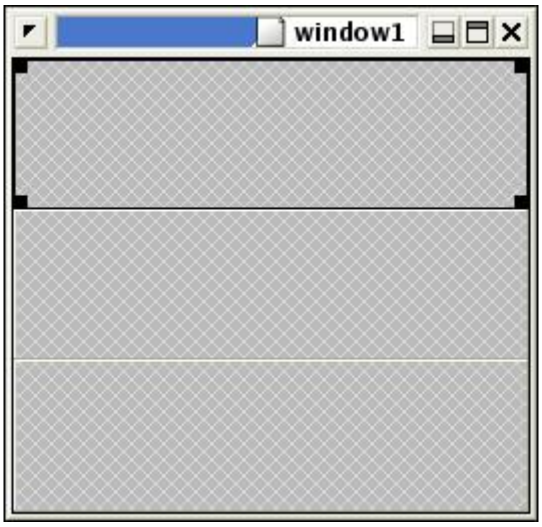


Figure 2. A basic three-pane vbox with the top pane selected.

You now can select any of those three areas and further divide it with a vertical box. If you don't like the results, you always can go back and delete, cut and paste or change the number of boxes from the Properties menu (more on that later).

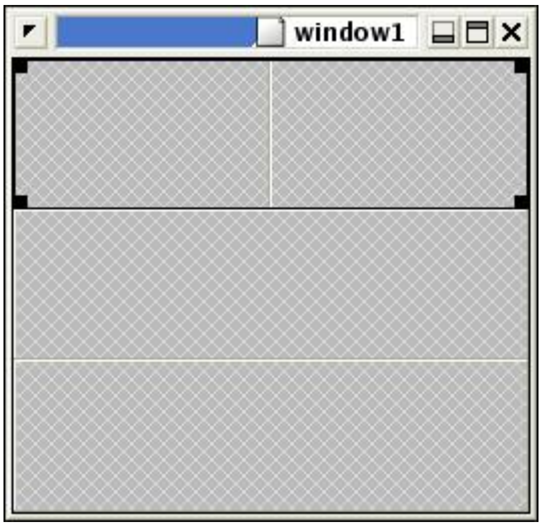


Figure 3. The top pane has been split by a two-pane hbox, which is selected.

You can use these sorts of primitives to create almost any sort of layout. Now that we have a beginning layout, we can fill it with widgets that actually do something. In this case, I'll fill them with a label, a text entry, a spinbutton and a button. At first this looks pretty ugly (Figure 4).

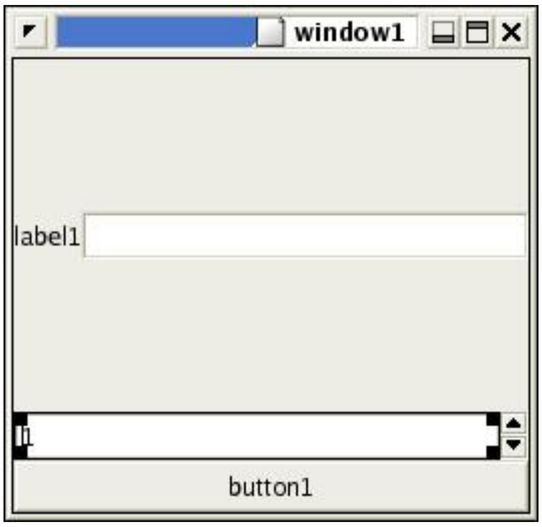
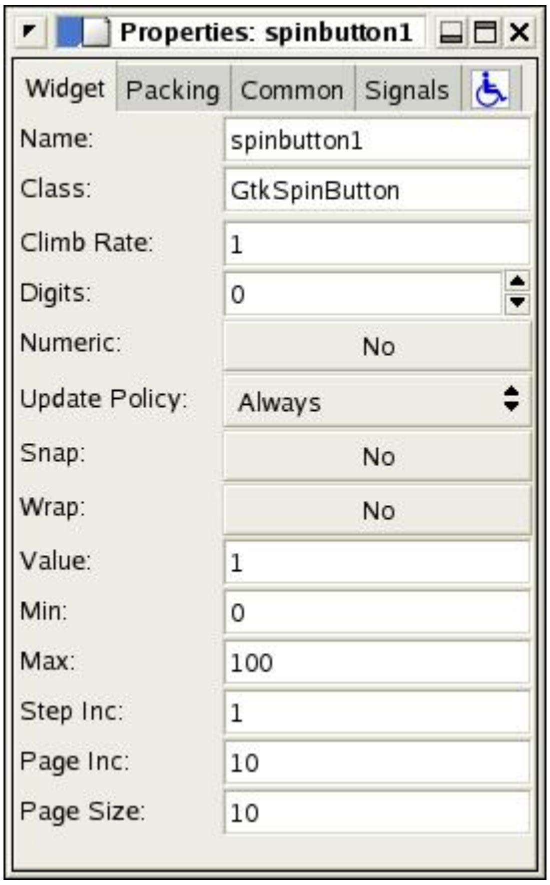


Figure 4. The initial window filled in with widgets.

Remember that GTK auto-adjusts the sizes of the finished product when it is displayed, so everything is packed together as tightly as possible. When the user drags the corner of the window, it's going to auto-expand as well. You can adjust these settings in the Properties window (go to the main Glade window and click View→Show Properties). The Properties window changes different values for different kinds of widgets. If the spinbutton is focused, for example, we see the options shown in Figure 5.



Properties: spinbutton1

Widget | Packing | Common | Signals

Name:	spinbutton1
Class:	GtkSpinButton
Climb Rate:	1
Digits:	0
Numeric:	No
Update Policy:	Always
Snap:	No
Wrap:	No
Value:	1
Min:	0
Max:	100
Step Inc:	1
Page Inc:	10
Page Size:	10

Figure 5. The Glade interface for changing a widget's properties is customized for each type of widget.

By changing the Value option, we can change what the spinbutton defaults to when displayed. Also important is to change the Max value. A common mistake is to change the Value to something high but forget the Max, which causes the spinbutton initially to display the default but then revert to the Max value when it is changed, confusing the user. In our case, we're going to use the spinbutton as a TCP port, so I'll set it to 65535, the minimum to 1 and the default to 80.

Then, focus on the label1 and change it to read Host:. By clicking on window1 in the main Glade window, you can focus on the entire window, allowing you to change its properties as well. You also can do this by bringing up the widget tree window and clicking on window1. Changing the name to serverinfo and the title to Server Info sets the titlebar and the internal Glade top-level widget name appropriately for this application.

If you go to the widget tree view and click on the hbox1, you can increase the spacing between Host: and the text-entry box. This may make it look a little nicer. Our finished GUI looks like Figure 6.

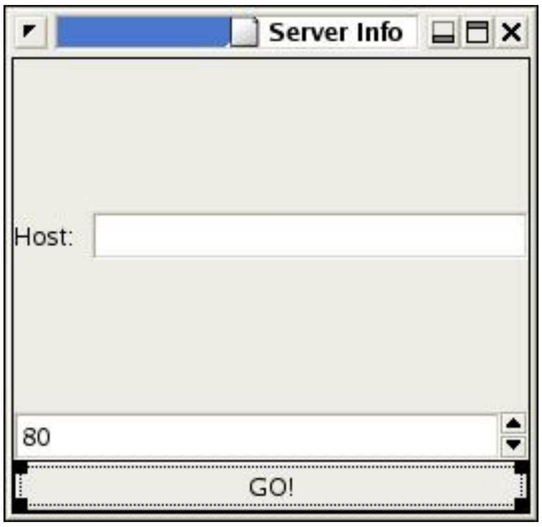


Figure 6. The GUI in Glade does not look exactly like it does when rendered, so don't worry about the size of the Host: area.

Normally, this would take only a few minutes to put together. After a bit of practice you'll find that putting together even the most complex GUIs using Glade can be accomplished in minutes. Compare that to the time it takes to type in all those Tk commands manually to do the same thing.

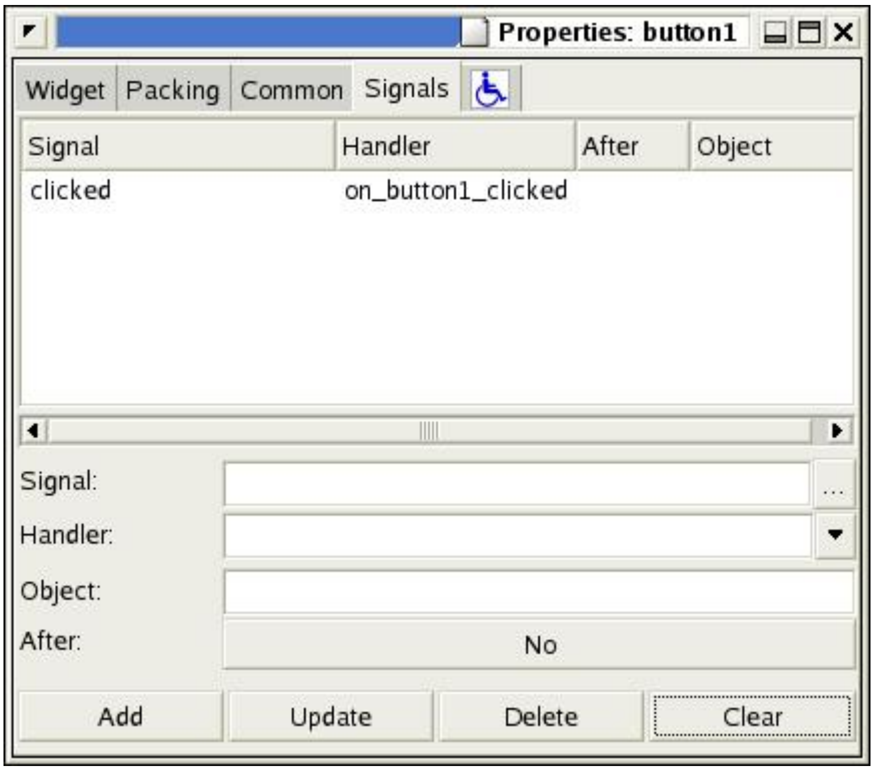
This GUI, of course, doesn't do anything yet. We need to write the Python code that loads the .glade file and does the actual work. In fact, I tend to write two Python files for each Glade-driven project. One file handles the GUI, and the other file doesn't know anything about that GUI. That way, porting from GTK v1 to GTK v2 or even to another GUI toolkit is easy.

Creating the Python Program

First, we need to deal with any potential version skew. I use the following code, although a few other entries mentioned in the FAQ do similar things:

```
#!/usr/bin/env python

import sys

try:
 import pygtk
  #tell pyGTK, if possible, that we want GTKv2
  pygtk.require("2.0")
except:
  #Some distributions come with GTK2, but not pyGTK
  pass

try:
  import gtk
  import gtk.glade
except:
  print "You need to install pyGTK or GTKv2 ",
  print "or set your PYTHONPATH correctly."
  print "try: export PYTHONPATH=",
  print "/usr/local/lib/python2.2/site-packages/"
  sys.exit(1)

#now we have both gtk and gtk.glade imported
#Also, we know we are running GTK v2
```

Now are going to create a GUI class called appGUI. Before we do that, though, we need to open button1's properties and add a signal. To do that, click the three dots, scroll to clicked, select it and then click Add. You should end up with something like Figure 7.



Signal	Handler	After	Object
clicked	on_button1_clicked		

Signal:
Handler:
Object:
After: No

Figure 7. After Adding the Event (Signal) Handler

With this in place, the signal_autoconnect causes any click of the button to call one of our functions (button1_clicked). You can see the other potential signals to be handled in that list as well. Each widget may have different potential signals. For example, capturing a text-changed signal on a text-entry widget may be useful, but a button never changes because it's not editable.

Initializing the application and starting gtk.mainloop() gets the ball rolling. Different event handlers need to have different numbers of arguments. The clicked event handler gets only one argument, the widget that was clicked. While you're at it, add the destroy event to the main window, so the program exits when you close the window. Don't forget to save your Glade project.

```
class appgui:
  def __init__(self):
    """
    In this init we are going to display the main
    serverinfo window
    """
    gladefile="project1.glade"
    windowname="serverinfo"
    self.wTree=gtk.glade.XML (gladefile,windowname)
    # we only have two callbacks to register, but
    # you could register any number, or use a
    # special class that automatically
    # registers all callbacks. If you wanted to pass
    # an argument, you would use a tuple like this:
    # dic = { "on button1_clicked" : \
             (self.button1_clicked, arg1,arg2) , ...

dic = { "on_button1_clicked" : \
            self.button1_clicked,
            "on_serverinfo_destroy" : \
            (gtk.mainquit) }
    self.wTree.signal_autoconnect (dic)
    return

#####CALLBACKS
  def button1_clicked(self,widget):
    print "button clicked"

# we start the app like this...
app=appgui()
gtk.mainloop()
```

It's important to make sure, if you installed pyGTK from source, that you set the PYTHONPATH environment variable to point to /usr/local/lib/python2.2/site-packages/ so pyGTK can be found correctly. Also, make sure you copy project1.glade into your current directory. You should end up with something like Figure 8 when you run your new program. Clicking GO! should produce a nifty button-clicked message in your terminal window.

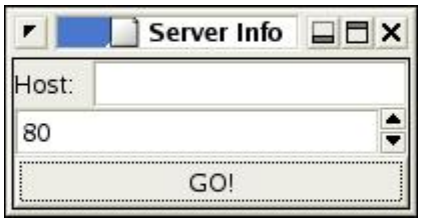


Figure 8. The Initial Server Info GUI

To make the application actually do something interesting, you need to have some way to determine which host and which port to use. The following code fragment, put into the button1_clicked() function, should do the trick:

```
  host=self.wTree.get_widget("entry1").get_text()
  port=int(self.wTree.get_widget(
    "spinbutton1").get_value())
  if host=="":
    return
  import urllib
  page=urllib.urlopen(
    "http://"+host+":"+str(port)+"/")
  data=page.read()
  print data
```

Now when GO! is clicked, your program should go off to a remote site, grab a Web page and print the contents on the terminal window. You can spice it up by adding more rows to the hbox and putting other widgets, like a menubar, into the application. You also can experiment with using a table instead of nested hboxes and vboxes for layout, which often creates nicer looking layouts with everything aligned.

TextViews

You don't really want all that text going to the terminal, though, do you? It's likely you want it displayed in another widget or even in another window. To do this in GTK v2, use the TextView and TextBuffer widgets. GTK v1 had an easy-to-understand widget called, simply, GtkText.

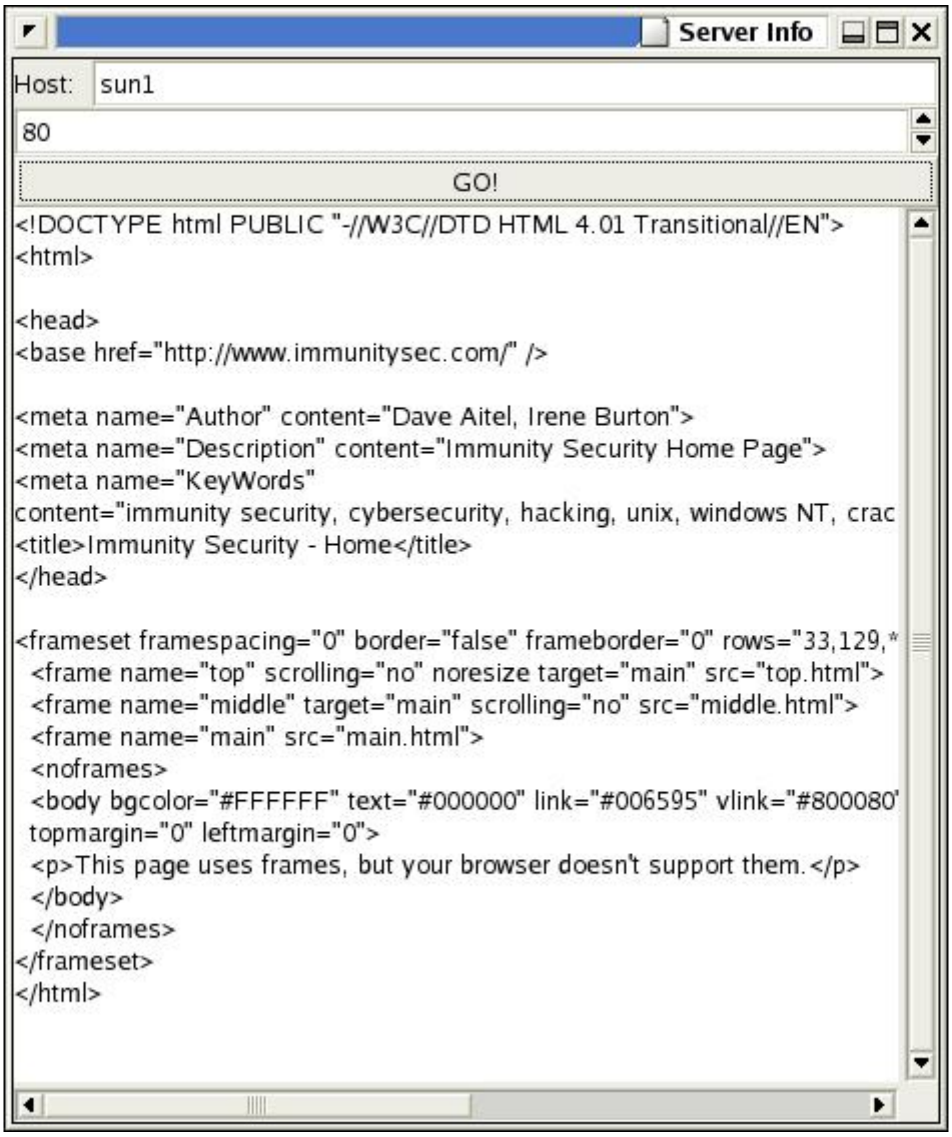
Add a TextView to your Glade project and put the results in that window. You'll notice that a scrolledwindow is created to encapsulate it. Add the lines below to your init() to create a TextBuffer and attach it to your TextView. Obviously, one of the advantages of the GTK v2 way of doing things is the two different views can show the same buffer. You also may want to go into the Properties window for scrolledwindow1 and set the size to something larger so you have a decent view space:

```
   self.logwindowview=self.wTree.get_widget("textview1")
   self.logwindow=gtk.TextBuffer(None)
   self.logwindowview.set_buffer(self.logwindow)
```

In your button1_clicked() function, replace the print statement with:

```
   self.logwindow.insert_at_cursor(data,len(data))
```

Now, whenever you click GO! the results are displayed in your window. By dividing your main window with a set of vertical panes, you can resize this window, if you like (Figure 9).



```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>

<head>
<base href="http://www.immunitysec.com/" />

<meta name="Author" content="Dave Aitel, Irene Burton">
<meta name="Description" content="Immunity Security Home Page">
<meta name="KeyWords"
content="immunity security, cybersecurity, hacking, unix, windows NT, crac
<title>Immunity Security - Home</title>
</head>

<frameset framespacing="0" border="false" frameborder="0" rows="33,129,*
  <frame name="top" scrolling="no" noresize target="main" src="top.html">
  <frame name="middle" target="main" scrolling="no" src="middle.html">
  <frame name="main" src="main.html">
  <noframes>
  <body bgcolor="#FFFFFF" text="#000000" link="#006595" vlink="#800080'
  topmargin="0" leftmargin="0">
  <p>This page uses frames, but your browser doesn't support them.</p>
  </body>
  </noframes>
</frameset>
</html>
```

Figure 9. Clicking GO! loads the Web page and displays it in the TextView.

TreeViews and Lists

Unlike GTK v1, under GTK v2 a tree and a list basically are the same thing; the difference is the kind of store each of them uses. Another important concept is the TreeIter, which is a datatype used to store a pointer to a particular row in a tree or list. It doesn't offer any useful methods itself, that is, you can't ++ it to step through the rows of a tree or list. However, it is passed into the TreeView methods whenever you want to reference a particular location in the tree. So, for example:

```
import gobject
self.treeview=[2]self.wTree.get_widget("treeview1")
self.treemodel=gtk.TreeStore(gobject.TYPE_STRING,
                             gobject.TYPE_STRING)
self.treeview.set_model(self.treemodel)
```

defines a tree model with two columns, each containing a string. The following code adds some titles to the top of the columns:

```
self.treeview.set_headers_visible(gtk.TRUE)
renderer=gtk.CellRendererText()
column=gtk.TreeViewColumn("Name",renderer, text=0)
column.set_resizable(gtk.TRUE)
self.treeview.append_column(column)
renderer=gtk.CellRendererText()

column=gtk.TreeViewColumn("Description",renderer,
                          text=1)
column.set_resizable(gtk.TRUE)
self.treeview.append_column(column)
self.treeview.show()
```

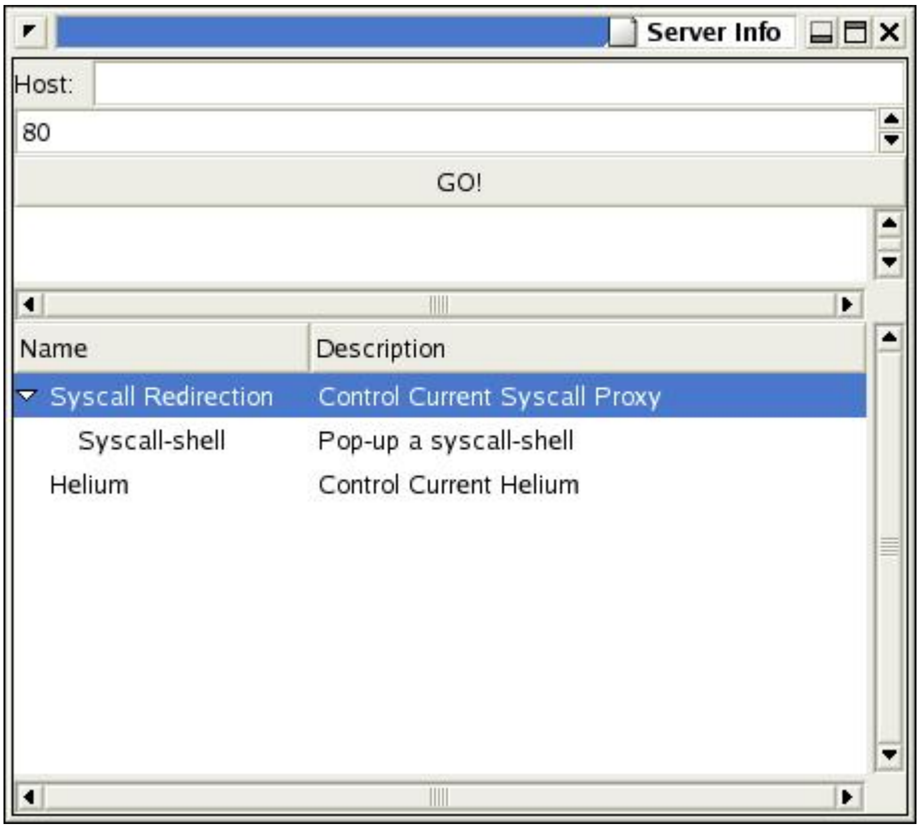
You could use the following function to add data manually to your tree:

```
def insert_row(model,parent,
               firstcolumn,secondcolumn):
    myiter=model.insert_after(parent,None)
    model.set_value(myiter,0,firstcolumn)
    model.set_value(myiter,1,secondcolumn)
    return myiter
```

Here's an example that uses this function. Don't forget to add treeview1 to your glade file, save it and copy it to your local directory:

```
model=self.treemodel
insert_row(model,None,'Helium',
           'Control Current Helium')
syscallIter=insert_row(model,None,
                    'Syscall Redirection',
                    'Control Current Syscall Proxy')
insert_row(model,syscallIter,'Syscall-shell',
           'Pop-up a syscall-shell')
```

The screenshot in Figure 10 shows the results. I've replaced the TextView with a TreeView, as you can see.



Name	Description
Syscall Redirection	Control Current Syscall Proxy
Syscall-shell	Pop-up a syscall-shell
Helium	Control Current Helium

Figure 10. An Example TreeView with Two Columns

A list is done the same way, except you use ListStore instead of TreeStore. Also, most likely you will use ListStore.append() instead of insert_after().

Using Dialogs

A dialog differs from a normal window in one important way—it returns a value. To create a dialog box, click on the dialog box button and name it. Then, in your code, render it with `[3]gtk.glade.XML(gladefile,dialogboxname)`. Then call get_widget(*dialogboxname*) to get a handle to that particular widget and call its run() method. If the result is gtk.RESPONSE_OK, the user clicked OK. If not, the user closed the window or clicked Cancel. Either way, you can destroy() the widget to make it disappear.

One catch when using dialog boxes: if an exception happens before you call destroy() on the widget, the now unresponsive dialog box may hang around, confusing your users. Call widget.destroy() right after you receive the response and all the data you need from any entry boxes in the widget.

Using input_add() and gtk.mainiteration() to Handle Sockets

Some day, you probably will write a pyGTK application that uses sockets. When doing so, be aware that while your events are being handled, the application isn't doing anything else. When waiting on a socket.accept(), for example, you are going to be stuck looking at an unresponsive application. Instead, use gtk.input_add() to add any sockets that may have read events to GTK's internal list. This allows you to specify a callback to handle whatever data comes in over the sockets.

One catch when doing this is you often want to update your windows during your event, necessitating a call to gtk.mainiteration(). But if you call gtk.mainiteration() while within gtk.mainiteration(), the application freezes. My solution for CANVAS was to wrap any calls to gtk.mainiteration() within a check to make sure I wasn't recursing. I check for pending events, like a socket accept(), any time I write a log message. My log function ends up looking like this:

```
def log(self,message,color):
"""
       logs a message to the log window
        right now it just ignores the color
        argument
        """
        message=message+"\n"
        self.logwindow.insert_at_cursor(message,
                                     len(message))
        self.handlerdepth+=1
        if self.handlerdepth==1 and \
        gtk.events_pending():
            gtk.mainiteration()
        self.handlerdepth-=1
        return
```

Moving a GUI from GTK v1 to GTK v2

The entry in the pyGTK FAQ on porting your application from GTK v1 to GTK v2 is becoming more and more complete. However, you should be aware of a few problems you're going to face. Obviously, all of your GtkText widgets need to be replaced with Gtk.TextView widgets. The corresponding code in the GUI also must be changed to accommodate that move. Likewise, any lists or trees you've done in GTK v1 have to be redone. What may come as a surprise is you also need to redo all dialog boxes, remaking them in GTK v2 format, which looks much nicer.

Also, a few syntax changes occurred, such as GDK moving to gtk.gdk and libglade moving to gtk.glade. For the most part, these are simple search and replaces. Use GtkText.insert_defaults instead of GtkTextBuffer.insert_at_cursor() and radiobutton.get_active() instead of radiobutton.active, for example. You can convert your Glade v1 file into a Glade v2 file using the libglade distribution's Python script. This gets you started on your GUI, but you may need to load Glade v2 and do some reconfigurations before porting your code.

Final Notes

- Don't forget you can cut and paste from the Glade widget tree. This can make a redesign quick and painless.
- Unset any possible positions in the Properties window so your startup doesn't look weird.
- If you have a question you think other people might too, add it to the pyGTK FAQ.
- The GNOME IRC server has a useful #pygtk channel. I couldn't have written CANVAS without the help of the people on the channel, especially James Henstridge. It's a tribute to the Open Source community that the principal developers often are available to answer newbie questions.

The finished demo code is available from ftp.linuxjournal.com/pub/lj/listings/issue113/6586.tgz.

Dave Aitel is the founder of Immunity, Inc., a New York-based security consulting company. CANVAS is Immunity's penetration testing and exploit development framework, written entirely in Python using pyGTK. More information on Immunity is available at www.immunitysec.com.

Archive Index Issue Table of Contents

Advanced search

From Vinyl to Digital

Tom Younker

Issue #113, September 2003

Nothing beats the sound quality of a good LP record, but if you need travel convenience, you can make a CD or Ogg Vorbis copy.

Dory Previn inspired me to pursue this project. She recorded a string of intelligent, literate albums in the 1970s, but by the early 1990s they still were not available on CD. With a little searching, I found a package named gramofile by Anne Bezemer and Ton Le designed to capture the sound of vinyl and prepare it for burning to CD. Then, along came xmcd2make by C. R. Johnson, which extends gramofile's functionality. Look at the possibilities:

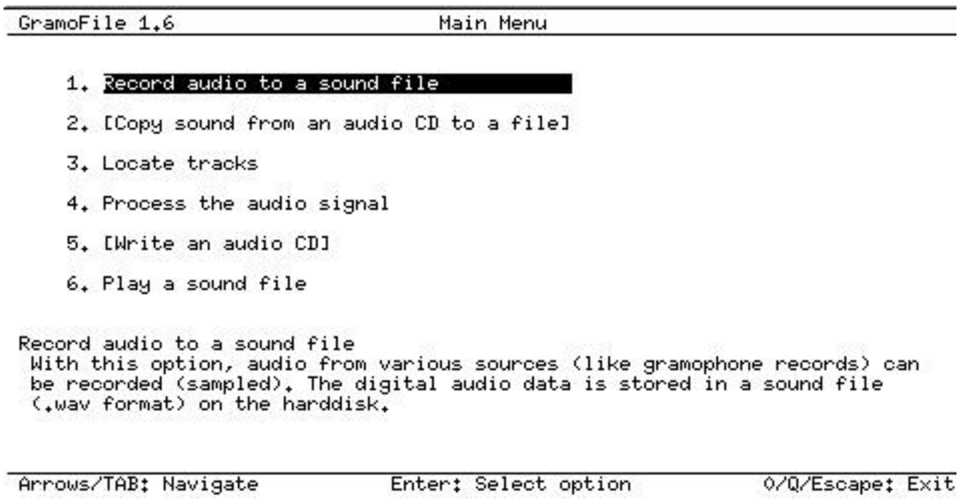
- Preserve the music on inexpensive, durable media.
- Have each track individually accessible with timing details.
- Encode to Ogg or MP3.
- Apply click reduction filters to all or selected tracks.
- Leave out unwanted songs and rearrange the order.
- Fit two albums onto one disk.

Preparing the Hardware

Sound quality depends on many factors, as explained in the “Linux Audio Quality HOWTO”, but with a quality sound card and well-supported drivers. Try to keep the card away from other cards to minimize induced noise. Because I'm using a spare PC solely for this project, I have only two cards installed: video in the first PCI slot and a SoundBlaster Live! 5.1 in the last PCI slot. If you plan to use the PC for other tasks while capturing the LP to disk, use a kernel from the 2.4 series, apply the preempt-kernel and lock-break patches (see Resources), select the new choices under Processor type, then build and install it. I was pleasantly surprised by the quality of sound on the CD from the first LP I processed.

Minimal Software

If you simply want to burn a CD, gramofile alone does the job. Install it from an RPM or deb file. Figure 1 shows the main menu. I discuss each step further later in this article. gramofile expects a mixer called xmixer, which I found in a Debian package called mctools-lite, but in the RPM world, it is in a package called multimedia. However, there's no problem using whatever mixer you have in another window or console.



```
 GramoFile 1.6                      Main Menu

      1. Record audio to a sound file

      2. [Copy sound from an audio CD to a file]

      3. Locate tracks

      4. Process the audio signal

      5. [Write an audio CD]

      6. Play a sound file

 Record audio to a sound file
  With this option, audio from various sources (like gramophone records) can
  be recorded (sampled). The digital audio data is stored in a sound file
  (.wav format) on the harddisk.

 Arrows/TAB: Navigate          Enter: Select option          0/Q/Escape: Exit
```

Figure 1. gramofile's Main Menu

Deluxe Software

For the encoding of songs in Ogg or MP3 format, consider taking the extra steps to install xmcd2make and its dependencies also. A bit of tedium up front will simplify the processing of every album. The xmcd2make scripts are simply that—Perl scripts whose installation amounts to `make install`. However, they won't work until swig, oggenc (for Ogg encoding) or lame (for MP3), mpgtx and Perl module Getopt::Long are installed. In addition, xmcd2make expects the special version of gramofile that has perl-swig extensions, with a P in the version name. So let's get to work. Because this machine is dedicated to one task, I'll do all installs as root and untar each package from /usr/local.

swig has progressed, but gramofile hasn't. Using the latest version 1.3.17, the gramofile `make perl-swig` failed. Using Debian's older version 1.1.p883-4 (`apt-get install swig`), the make completed. Here's the manual equivalent:

```
tar xvzf swig1.1-883.tar.gz
cd  SWIG1.1-883
./configure
make
make install
```

oggenc should be available as RPM or deb packages, though the name may be elusive. For Debian, I used `apt-get install vorbis-tools libvorbis0`. Lame may be harder to find due to patent issues, and Ogg is the politically correct choice.

mpgtx, a command-line MPEG toolbox, is a simple `apt-get install mpgtx` in Debian, but I installed version 1.3 from source with the classic `tar`, `configure`, `make`, `make install` as in swig above. Though pages of warnings scrolled by, it installed without complaint.

The Perl module Getopt::Long is part of Debian's 5.6.1 package, and I hope it's in yours too. On my system it's installed in /usr/share/perl/5.6.1/Getopt/Long.pm.

A manual install of gramofile with perl-swig extensions is not for the timid. It requires ncurses5-dev and doesn't install itself. You'll need to know where your Perl CORE resides, so try:

```
cd /usr/lib
find -name CORE
./perl/5.6.1/CORE
```

This reveals that my Perl CORE is at /usr/lib/perl/5.6.1/CORE. After untarring gramofile, edit the Makefile in the perl-swig subdirectory, changing the line `PERLCORE = -I/usr/...` to match your installation:

```
tar xvzf gramofile-1.6P
cd gramofile-1.6P
cd perl-swig
```

(edit the Makefile here)

```
cd ..
make
make perl-swig
```

Now, let's copy the executables into a directory that's in your $PATH:

```
cp gramofile bplay_gramo brec_gramo /usr/bin
```

Finally, `cd` into the perl-swig subdirectory, and copy two files from there into a directory in the Perl path. But first, check your Perl path for a suitable directory:

```
perl -e 'print join("\n",@INC), "\n"'
```

So, in my Debian setup:

```
cd perl-swig
mkdir /usr/local/lib/site_perl
cp Gramofile.pm Gramofile.so /usr/local/lib/site_perl
```

Finally, we're ready for xmcd2make:

```
tar xvzf xmcd2make-0.4.tar.gz
cd xmcd2make-0.4
make install
```

xmcd2make installs with a default bitrate of 128, but I prefer a higher bitrate (at the expense of larger files and longer encoding time), so I edited the file /usr/local/bin/xmcd2make and changed the bitrate to 224:

```
# $bitrate = 128;
$bitrate = 224;
```

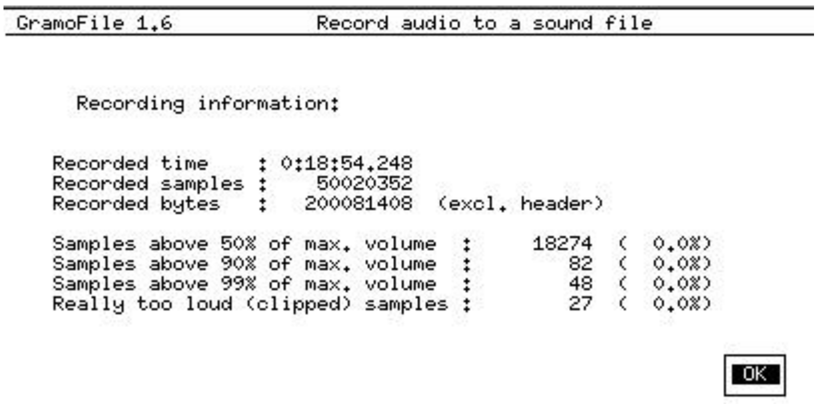
If you can stand one more item, I recommend a mixer called umix, because it has a console version, offers numeric levels to set levels accurately and repeatably, and the ability to save or restore all settings with a single keystroke. This means the whole LP to CD process can be done on a low-end PC without even installing X. The default mixer path is /dev/sound/mixer, which you may want to adjust as follows: `./configure --with-mixer-dev=/dev/mixer`. To load and save levels as an ordinary user, start umix with a config filename, like `umix -f $HOME/umixrc`. Press S to save the current settings and L to reload the last saved settings.

Basic Recording Procedure

Here are the steps to capture Dory's *On My Way To Where* album to the hard drive, then process it to arrive at a set of wav files suitable for burning an audio CD, plus a set of Ogg files for use by computers and portable players. Eventually, we'll tell xmcd2make the basename is “where”, so we name the files appropriately.

- Position the computer near your stereo, and connect stereo line out to your sound card line in with a quality, shielded cable. You probably will need a dual RCA-to-stereo mini-plug cable to make this connection.
- Load your mixer in one console or xterm. Load gramofile in another after changing to an empty directory on a partition with a lot of free space.
- Use the mixer to set “line in” to record mode and to mute all other channels. You may need to bring up input gain (igain) too. This reduces background noise.
- Use gramofile's Record audio to a sound file to capture a sample, and use the mixer to adjust levels to bring the peaks on the gramofile level meter near the top.

- Stop sampling, and verify that a reasonable percent of samples were above 50% of max volume, a few above 90%, but that few or none were above 99% (Figure 2).
- Set gramofile to capture side one of the LP by giving it a descriptive name, in our example, where1.wav; start playing the LP, then start capturing.
- When side one is completed, stop gramofile and verify that the sample levels are reasonable. If not, use your mixer to adjust the levels. Some of the volume spikes are caused by clicks, so a few clipped samples are acceptable.
- Capture side two to a file called where2.wav.



```
 GramoFile 1.6              Record audio to a sound file

       Recording information:

     Recorded time     : 0:18:54.248
     Recorded samples :    50020352
     Recorded bytes    :   200081408  (excl. header)

     Samples above 50% of max. volume  :     18274  (  0.0%)
     Samples above 90% of max. volume  :        82  (  0.0%)
     Samples above 99% of max. volume  :        48  (  0.0%)
     Really too loud (clipped) samples :        27  (  0.0%)

                                                            OK
```

Figure 2. gramofile's Recording Information

Now we have two wav files of about 200MB each, digital representations of the sound in the vinyl grooves. This is a good time to decide whether this LP has significant surface noise—those clicks for which vinyl is famous. If the whole album is noisy, run both sides through the filter(s) gramofile offers. If only selected songs are noisy, typically the first track on each side, wait until the tracks are split.

The gramofile documentation includes a fascinating discussion (Signproc.txt) of each filter and the theory behind it. You'll notice when choosing Process the audio signal that Conditional Median Filter II already is selected. It's the most sophisticated, and I had good results with it. Clicking noises aren't obliterated, but they're greatly reduced. Multiple filters can be used, or you can use the same filter twice. I stick with one pass because when I used Conditional Median Filter II twice, there was a noticeable degradation in the music. However, the process takes only a few minutes, so feel free to experiment. The original file is preserved, and you have the opportunity to give the filtered file a meaningful name. Listen to this new wav file. If you like the result, delete the original and rename the filtered file to be the same as the original.

Before encoding any tracks, you'll want a listing of the artist name, album name and the title of every track. You can save some typing by searching freedb.org. If your album is found, click on the IDs link above the first track title, then save the page that comes up as a text file. If you prefer, simply copy the listing from this xmcd page and paste it into a text editor, ignoring any lines starting with #. Be sure to save the file as plain text into the directory where your wav files are, and name it to match. For our example, that would be where.xmcd. In either case, all we'll be using are DTITLE and TTITLE lines. For obscure LPs, I keep a copy of an xmcd file with only DTITLE= and TTITLE0= thru TTITLE10= in my home directory. It only takes a minute to copy it with a name to match the current LP, then type in the titles from the album jacket. Notice that the track listings start with zero. Here's a partial sample:

```
DTITLE=Dory Previn / On My Way To Where
TTITLE0=Scared To Be Alone
TTITLE1=I Ain't His Child
```

Our task now is to split each side into individual songs. Straight gramofile users can choose Locate tracks from the menu, then choose side one (where1.wav), click Next, click Start computation and wait for the song count to come up. If the count is not correct, try again but adjust one or more of the options before Start computation. Try reducing intertrack silence to 12 or less. Repeat for side two. When you're happy with the track counts, choose Process the audio signal; choose the side one wav file again; click Next; then, tab through to the “Next” screen if you want to use the default click filter. To use no filter, stop at the Available filters box, highlight Copy only and press the Enter key. Now tab to the Selected filters box, highlight Conditional Median Filter II and press the R key to remove it. Use the same procedure to choose a different (or additional) filter, but you must select either a filter or Copy only to process the files. Incidentally, pressing the Enter key while a filter is highlighted in the Selected filters box allows you to change specific filter settings. Finally, tab to Start and press Enter. When it completes, you should have a wav file for each track, ready for test listening and then burning.

xmcd2make users should exit gramofile, as xmcd2make's findtracks script is a wrapper around gramofile's findtracks function. Run `findtracks where1.wav` from the prompt to scan album side one, then compare its output with the official track listing. Type `less *.tracks`, and you'll see a plain-text file with start and stop timings for each track. If you see two tracks lumped together, go back to the prompt and try again with one or more of the parameters adjusted, for example:

```
findtracks where1.wav --min-silence-blocks 12
```

It's possible, though tedious, to split the songs manually. A crude alternative is to burn the whole side as a single track. In this case, use gramofile's filter, but choose Copy only (as explained above but unchecking the Split tracks option), which would do no filtering but would add timing information to the wav file, so the CD burner would know the true running time. If you find a song split in two, it's easy to edit the tracks file to merge the two, then renumber the rest of the tracks on that side and edit the `Number_of_tracks=` line to match. Finally, repeat the whole process for side 2.

Now, we're ready to create a Makefile, which automates the rest of the process:

```
xmcd2make --basename where --counts 5,5 > Makefile
```

This creates a Makefile that includes the timing for splitting the tracks and the album, artist and trackname information for Ogg or MP3 creation. The counts must match those in the track's files. To adjust the bitrate, add `--rate 192`, for example. Now simply typing `make` copies each track's content to a separate wav file and encodes it to an Ogg file with a descriptive name. Here's an abbreviated directory listing from our example:

```
Makefile                    9.5k
where1.wav                  196M
where1.wav.tracks           1.2k
where2.wav                  191M
where2.wav.tracks           1.2k
where_processed_101.wav      52M
101_Scared_To_Be_Alone.ogg  8.2M
```

Use `make mp3` to split tracks and create MP3 files instead of Ogg files. Typing `make proc` simply splits the tracks, allowing you to apply a filter to selected tracks. By deleting the original unfiltered file and renaming the filtered file to the original name, `make` creates an Ogg file from the filtered wav file. For more options, try `xmcd2make --help`.

There you have it—wav files to burn to an audio CD-R and Ogg files to play back on any capable device or burn to a data CD-R. If you're new to CD burning, there are many fine HOWTOs available. Once you've set up the system, you can repeat it for stacks of albums and enjoy the results. By the way, in 2002 the last of Dory's albums finally were re-issued on CD, but I already have my Linux-made disks.

Resources

CD Burning: www.tldp.org/HOWTO/CD-Writing-HOWTO.html

gramofile with or without extensions: panic.et.tudelft.nl/~costar/gramofile

gramofile with perl-swig extensions and xmcd2make: ftp.freeengineer.org/pub/xmcd2make

Linux Audio-Quality HOWTO: www.linuxdj.com/audio/quality

Kernel Patches: www.tech9.net/rml/linux

mpgtx: mpgtx.sourceforge.net

Perl Module Installation: www.perldoc.com/perl5.8.0/lib/CPAN.html

swig: www.swig.org

umix: umix.sourceforge.net

Tom Younker (tom@darecomputer.com) lives in smoggy Atlanta, Georgia with his Mac-loving wife and a basement full of Linux boxen. He also runs a consulting business.

Garbage Collection in C Programs

Gianluca Insolvibile

Issue #113, September 2003

LISP and Java programmers take garbage collection for granted. With the Boehm-Demers-Weiser library, you easily can use it in C and C++ projects, too.

The first word that came to mind when I heard about introducing Garbage Collection techniques into a C or C++ program was “nonsense”. As with any other decent C programmer who loves this language, the thought of leaving the management of my own memory to others seemed nearly offensive. I had a similar feeling 15 years ago, when I first heard about compilers that would generate assembly code on my behalf. I was used to writing my code directly in 6510 opcodes, but that was Commodore 64—and a totally different story.

What Is Garbage Collection?

Garbage Collection (GC) is a mechanism that provides automatic memory reclamation for unused memory blocks. Programmers dynamically allocate memory, but when a block is no longer needed, they do not have to return it to the system explicitly with a free() call. The GC engine takes care of recognizing that a particular block of allocated memory (heap) is not used anymore and puts it back into the free memory area. GC was introduced by John McCarthy in 1958, as the memory management mechanism of the LISP language. Since then, GC algorithms have evolved and now can compete with explicit memory management. Several languages are natively based on GC. Java probably is the most popular one, and others include LISP, Scheme, Smalltalk, Perl and Python. C and C++, in the tradition of a respectable, low-level approach to system resources management, are the most notable exceptions to this list.

Many different approaches to garbage collection exist, resulting in some families of algorithms that include reference counting, mark and sweep and copying GCs. Hybrid algorithms, as well as generational and conservative variants, complete the picture. Choosing a particular GC algorithm usually is not a programmer's task, as the memory management system is imposed by the adopted programming language. An exception to this rule is the Boehm-Demers-Weiser (BDW) GC library, a popular package that allows C and C++ programmers to include automatic memory management into their programs. The question is: Why would they want to do a thing like this?

The Boehm-Demers-Weiser GC Library

The BDW library is a freely available library that provides C and C++ programs with garbage collection capabilities. The algorithm it employs belongs to the family of mark and sweep collectors, where GC is split into two phases. First, a scan of all the live memory is done in order to mark unused blocks. Then, a sweep phase takes care of putting the marked blocks in the free blocks list. The two phases can be, and usually are, performed separately to increase the general response time of the library. The BDW algorithm also is generational; it concentrates free space searches on newer blocks. This is based on the idea that older blocks statistically live longer. To put it another way, most allocated blocks have short lifetimes. Finally, the BDW algorithm is conservative in that it needs to make assumptions on which variables are actually pointers to dynamic data and which ones only look that way. This is a consequence of C and C++ being weakly typed languages.

The BDW collector comes as a static or dynamic library and is installed easily by downloading the corresponding package (see Resources) and running the traditional configure, make and make install commands. Some Linux distributions also come with an already-made package. For example, with Gentoo you need to type only `emerge boehm-gc` to install it. The installed files include both a shared object (libgc.o) and a static library (libgc.a).

Using the library is a fairly straightforward task; for newly developed programs, you simply call GC_alloc() to get memory and then forget about it when you do not need it anymore. “Forget about it” means setting all the pointers that reference it to NULL. For already existing sources, substitute all allocation calls (malloc, calloc, realloc) with the GC-endowed ones. All free() calls are replaced with nothing at all, but do set any relevant pointers to NULL.

GC_alloc() actually sets the allocated memory to zero to minimize the risk that preexisting values are misinterpreted as valid pointers by the GC engine. Hence, GC_alloc() behaves more like calloc() than malloc().

Using GC in Existing C Programs

If you want to try GC in an existing application, manually editing the source code to change mallocs and frees is not necessary. In order to redirect those calls to the GC version, you basically have three options: using a macro, modifying the malloc hooks and overriding glibc's malloc() with libgc's malloc(). The first approach is the easiest one; you simply need to insert something like:

```
#define malloc(x) GC_malloc(x)
#define calloc(n,x) GC_malloc((n)*(x))
#define realloc(p,x) GC_realloc((p),(x))
#define free(x) (x) = NULL
```

into the appropriate include files. This code substitutes only the explicit calls contained in your code, leaving startup and library allocations to traditional malloc/free calls.

A different approach is to hook malloc and friends to functions of your own, which in turn would call the GC versions. Listing 1 shows how to do it, and it can be linked directly to an existing program. See my article “Advanced Memory Allocation” [*LJ*, May 2003] for details on these hooks. With this method, any heap allocation is guaranteed to go through libgc, even if it is not performed directly by your code.

Listing 1. Using glibc's malloc and free Hooks to Enable Garbage Collection

```
/*
 * Using the malloc hooks to substitute GC functions
 * to existing malloc/free.
 * Similar wrapper functions can be written
 * to redirect calloc() and realloc()
 */

#include <malloc.h>
#include <gc.h>

static void gc_wrapper_init(void);
static void *gc_wrapper_malloc(size_t,const void *);
static void gc_wrapper_free(void*, const void *);

__malloc_ptr_t (*old_malloc_hook)
    __MALLOC_PMT((size_t __size,
                  __const __malloc_ptr_t));
void (*old_free_hook)
    __MALLOC_PMT ((__malloc_ptr_t __ptr,
                   __const __malloc_ptr_t));


/* Override initializing hook from the C library. */
void (*__malloc_initialize_hook)(void) =
                                    gc_wrapper_init;

static void gc_wrapper_init() {
    old_malloc_hook = __malloc_hook;
    old_free_hook = __free_hook;
    __malloc_hook = gc_wrapper_malloc;
    __free_hook = gc_wrapper_free;
}

void *
gc_wrapper_malloc(size_t size, const void *ptr) {
    void *result;
    /* Restore all old hooks */
    __malloc_hook = old_malloc_hook;
    __free_hook = old_free_hook;
```

```
        /* Call the Boehm malloc */
        result = GC_malloc(size);


        /* Save underlying hooks */
        old_malloc_hook = __malloc_hook;
        old_free_hook = __free_hook;

        /* Restore our own hooks */
        __malloc_hook = gc_wrapper_malloc;
        __free_hook = gc_wrapper_free;

        return result;
    }

    static void
    gc_wrapper_free(void *ptr, const void *caller)
    {
      /* Nothing done! */
    }
```

As a third alternative, you can pass --enable-redirect-malloc to **configure** before compiling the libgc library. Doing so provides the library with wrapper functions that have the same names as the standard glibc malloc family. When linking with your code, the functions in libgc override the standard ones, with a net effect similar to using malloc hooks. In this case, though, the effect is system-wise, as any program linked with libgc is affected by the change.

Do not expect to endow huge programs with GC easily using any of these methods. Some simple tricks are needed in order to exploit GC functions and help the collector algorithm work efficiently. For example, I tried to recompile gawk (version 3.1.1) using GC and obtained an executable ten times slower than the original. With some adjustments, such as setting each pointer to NULL after having freed it, the execution time improved significantly, even if it was still greater than the original time.

Garbage Collection in New Programs

If you are developing a new program and would like to take advantage of automated memory management, all you need to do is use the GC_malloc() family in place of the plain malloc() one and link with libgc. Memory blocks no longer needed simply can be disposed of by setting any referencing pointers to NULL. Alternatively, you can call GC_free() to free the block immediately.

Always remember that your whole heap is scanned periodically by the collector to look for unused blocks. If the heap is large, this operation may take some time, causing the performance of the program to degrade. This behavior is suboptimal, because large blocks of memory often are guaranteed never to contain pointers, including buffers used for file or network I/O and large strings. Typically, pointers are contained only in fixed positions within small data structures, such as list and tree nodes. Were C and C++ strongly typed languages, the collector could have decided whether to scan a memory block, based on the type of pointer. Unfortunately, this is not possible because it is perfectly legal in C to have a char pointer reference a list node.

For optimal performance, the programmer should try to provide some basic runtime type information to the collector. To this end, the BDW library has a set of alternative functions that can be used to allocate memory. GC_malloc_atomic() can be used in place of GC_malloc() to obtain memory blocks that will never contain valid pointers. That is, the collector skips those blocks when looking for live memory references. Furthermore, those blocks do not need to be cleared on allocation. GC_malloc_uncollectable() and GC_malloc_stubborn() also can be used to allocate fixed and rarely changing blocks, respectively. Finally, it is possible to provide some rough type information by using GC_malloc_explicitly_typed() and building block maps with GC_make_descriptor(). See gc_typed.h on the *Linux Journal* FTP site for more information [available at ftp.linuxjournal.com/pub/lj/listings/issue113/6679.tgz].

The collector's behavior also can be controlled by the user through a number of function calls and variables. Among the most useful ones are GC_gcollect(), which forces a full garbage collection on the whole heap; GC_enable_incremental(), which enables incremental mode collection; and GC_free_space_divisor, which tunes the trade-off between frequent collections (high values, causing low heap expansion and high CPU overhead) and time efficiency (low values).

Heap status and debug information is available through a number of functions, including GC_get_heap_size(), GC_get_free_bytes(), GC_get_bytes_since_gc(), GC_get_total_bytes() and GC_dump(). Many of these parameters and functions are not documented at all, not even in the source code itself. As always, a good editor is your friend.

Performance and Behavior

A single best approach to memory management that is effective for any program does not exist. Given a specific application, the optimal solution must be found by compromising on a number of different factors, including CPU overhead, heap expansion, allocation latency and, last but not least, manageability and robustness of the code. Profiling the program and testing different memory strategies probably is the best solution for dealing with these issues.

Garbage Collection and Compiler Optimizations

One subtle point against GC is it requires extra care if compiler optimizations are switched on. The collector may wrongly assume a certain pointer has disappeared simply because references to it have been optimized out. Thus, the corresponding memory block might be freed even if it still is in use by the program, with the obvious consequences. Hence, it might be tempting to turn compiler optimizations off to be safe, losing part of the performance gain obtained by using GC.

In order to have an idea of the behavior and performance of GC vs. traditional memory management, we have experimented with a test program, gctest, which loops over the creation and destruction of a simple list. Simple as it may seem, the test raises some interesting points. The source code is not really instructive and is too long to be printed, so it is available for download on the FTP site (ftp.linuxjournal.com/pub/lj/listings/issue113/6679.tgz).

gctest can be controlled with a number of options that allow you to experiment with different list lengths and node sizes, as well as to change working parameters—enabling and disabling specific features of the GC library. Before we comment on the results we obtained with this test tool, it is important to point out that they were obtained in an artificial and not excessively realistic environment. So, again, you are invited to test the GC library yourself and evaluate it for your own code. Take the parameters presented here as possible indicators of the library suitability to a particular application.

The measurements we collected are overall execution time, the CPU load; heap expansion, how much memory is requested by the system with respect to how much actually was allocated by the program; and allocation latency average and standard deviation, how long it takes to allocate a single block of memory and how much this time varies across different allocations. The meaning of the first parameter is quite obvious and needs no further explanation. Heap expansion is a measure of how much of the allocated memory is fragmented and what amount of extra memory is requested by the library from the operating system. As we will see, the library may allocate a 1MB block ten times, freeing it after each allocation and requesting a total of 10MB of the system, as if freed memory was not put back on the free list. Although this sometimes is desired behavior, needed to optimize the allocation strategy, it can be annoying on systems with a limited availability of RAM. It can become a further source of CPU overhead if swap space is involved. Finally, allocation latency is important for real-time applications, which need the longest allocation time to be bounded. Typical cases are multimedia playback applications and specialized embedded systems that need to react to external events in a predictable amount of time.

Some Comments on the Results

Our test box A is a Pentium 4 2.53GHz system, with 1GB of RAM running Gentoo Linux (all code is optimized for the CPU architecture) and glibc 2.3, which has an improved memory management algorithm over glibc 2.2. Test box B is a K6-II 400MHz laptop with 128MB of RAM, running Slackware Linux with glibc 2.2.

Our first test consisted of allocating a list of 150,000 nodes, 16 bytes each, 30 times. On each loop, the allocated list was destroyed, that is, free()ed in case of traditional management, unlinked in case of GC. The test commands were:

```
gctest -tu -s 4 -n 150000 -l 30
```

and:

```
gctest -gu -s 4 -n 150000 -l 30
```

The overall execution time, on box A, was 3.80 seconds with traditional management and 2.43 seconds with GC, an improvement of about 35%. The same test performed on box B showed an even greater improvement, around 40%. This first test shows that, contrary to popular belief, GC actually can be quite faster than malloc/free. Heap expansion is rather limited and amounts to about 2 in both cases. What is even more surprising is that allocation latency is the same—6.7 microseconds—with a slightly larger deviation for the GC case. Also interesting is that by calling GC_gcollect() at each loop (option -G), the overall execution time decreases by 0.1 second. This result is counterintuitive, because we have one more function call in the loop.

Now, let's see what happens if we forget to destroy the list at the end of each loop. In the traditional management case, the test executes faster, 2.58 vs. 3.80 seconds, but the peak heap expansion is 140MB, which is twice the overall allocated memory. In the GC case, the test aborts due to memory exhaustion unless we call for an explicit collection (-G) at the end of each loop. By doing so, we obtain the lowest execution time, 2.32 seconds. This probably is quite far from what we could have imagined a priori—that's why actual experimentation is important for finding the optimal solutions.

The same test also has been performed on box B, but with an unoptimized Slackware distribution and glibc 2.2. It is interesting that although the improvement of GC over malloc/free still was around 40%, the test ran 27% faster under Gentoo.

The second test we made shows some limitations of the GC library. The test conditions actually were quite extreme: we tried to allocate five lists, each having 1,500,000 nodes, with each node being 16 bytes long. Although the malloc/free version ran correctly, the GC version did not complete the test because of memory exhaustion. The problem probably is due to the large number of blocks consecutively allocated.

The third test used larger nodes, 140 bytes each, and a shorter list length, 150,000 nodes. We ran:

```
gctest -tu -s 128 -n 150000 -l 5
```

and:

```
gctest -gu -s 128 -n 150000 -l 5
```

Under these conditions, the improvement of GC over malloc/free was around 20%, 0.85 vs. 0.60 seconds. Recall, though, the GC library always clears the allocated blocks, but malloc() does not. The overhead associated with this operation grows linearly with node size and thus is more important with larger nodes. To make a fair comparison, then, we need to substitute malloc() with calloc() at those points where GC_malloc() is called, as happens in:

```
gctest -tuc -s 128 -n 150000 -l 5
```

This test yields an execution time of 0.88 seconds and brings the GC improvement to 32%. Heap expansion is greater in the GC case, with a value of 1.7 vs. 1.0. Allocation latency is practically the same for both traditional and GC management, although a larger latency variation was experienced in the latter. Enabling incremental collection (-i option) did not lower the variation, although introducing calls to GC_free() (-F option) to explicitly free the list nodes explicitly actually did yield better results than the malloc/free case, on both execution time and latency. However, in this case we are not strictly using a real garbage collection approach.

Testing even larger memory blocks makes the difference between traditional and GC memory management quite noticeable. On 4KB nodes, GC performed quite poorly in comparison with malloc/free on execution time, 0.85 vs. 2 seconds; heap expansion, 2.75 vs. 1; and latency, 0.7 milliseconds vs. 1.6 microseconds. When compared with calloc/free performance, the execution time of GC still is quite competitive (40% faster). But, issues related to heap and latency remain.

Conclusion

GC techniques often are surrounded by myths and legends. In this article we have shown that GC actually can perform better than malloc/free. These advantages do not come for free, however, and the correct use of the library requires minimum knowledge on its internal mechanisms.

There is no final verdict on the suitability of GC for C programs. For now, the BDW library can be one more tool in your box, to be given serious consideration the next time you deal with a complex piece of software. Several open-source projects, like the Mono Project and GNU gcj Java runtime, have been using it for a while.

Pros and Cons

Before deciding that GC is for wimps and a hard-core C programmer would never need it, it may be beneficial to consider the actual advantages GC may offer with respect to the traditional C/C++ memory management schemes. As Ellis and Stroustrup say in *The Annotated C++ Reference Manual*, "C programmers think memory management is too important to be left to the computer. LISP programmers think memory management is too important to be left to the user."

That memory-related problems contain some of the most insidious and time-wasting bugs a C programmer can encounter is a well-known fact. Even an experienced programmer might have a hard time tracking down bugs caused by invalid accesses, overflowing writes, accesses to dead memory, memory leaks and the like. Furthermore, from a software design point of view, the need to prevent such situations often leads designers to unclean interfaces between modules that should be decoupled. Think of functions that return a dynamically allocated memory block that then must be freed by the caller or a pointer to an internal static buffer, spoiling threadsafety—strtok() is a typical example in this sense—not to mention problems arising from memory areas shared among several modules. Each module has to free such areas only if no other modules are or will be using it, resulting in a tighter coupling between the modules themselves. This problem usually is addressed by using reference counting, where an internal counter is kept for each active user of the area; multiple copies, where one copy of the area is kept for each module; or, for those who fancy C++, smart pointers.

GC is an effective way of dealing with memory-related problems in C, relieving the programmer of memory accounting burdens. Each memory block knows when it is in use, actually, the GC engine knows, and the block automatically disappears when it is not referenced anymore. GC eliminates a priori all premature frees and memory leak problems.

GC has some drawbacks, of course, the most annoying being the feeling of having lost control that afflicts C programmers. More concretely, drawbacks stem from the automated management of resources, which translates into:

- Not knowing when an unused memory block actually is freed and if it ever will be. This has further consequences when the memory block is a C++ object, whose destructor is called at an unspecified time.
- Not being able to determine in advance how much time an allocation takes (allocation latency and jitter), which often is an issue for real-time systems. It must be said, though, that traditional malloc() also presents this problem sometimes. Incremental GC can help alleviate the problem and provide limited constraints to allocation times.
- Longer execution time due to GC processing overhead. This often is not true, as sometimes the overhead associated with traditional free()s is equal to or even greater than that of GC. In this sense, only storage allocators written specifically for a given program might be faster. Even so, programmers often write their own memory management systems without a preliminary profiling activity, sometimes resulting in negative effects on performance.
- Higher memory fragmentation (heap expansion), caused by unused objects not being freed by the GC engine when more memory is allocated by the program. It is quite common to have a heap much larger than the amount of memory in actual use at a certain time. Even worse, the heap size sometimes is proportional to the total amount of allocation performed since the program started. This may be a serious issue on systems with scarce RAM availability, where frequent collections and explicit free() calls might be necessary to limit fragmentation.
- Reduced locality of reference, due to the garbage detection phase that has to traverse the whole addressable heap space to look for unused blocks. This results in cache and virtual memory misses more often than happens with traditional allocations. Generational GC algorithms limit the severity of this problem.

GC enthusiasts claim that these issues are not so relevant, considering the cleaner design and greater robustness that GC offers. Even if these advantages are difficult to be evaluated concretely, they sometimes provide a convenient trade-off between code manageability and loss of resources.

Finally, the BDW library also can be used proficiently as a leak detector; the Mozilla team uses it this way, for example. For more information, have a look at the included documentation.

Resources

The Boehm-Demers-Weiser home page at www.hpl.hp.com/personal/Hans_Boehm/gc provides the source code for the library and a wealth of documentation and links for more information on GC. Good starting points are Hans Boehm and David Chase, “A proposal for Garbage-Collector-Safe C Compilation”, 1992, available at www.hpl.hp.com/personal/Hans_Boehm/gc/papers/boecha.ps.gz; Paul Wilson, “Uniprocessor Garbage Collection Techniques”, University of Texas, 1992, available at ftp.cs.utexas.edu/pub/garbage/gcsurvey.ps; and Benjamin Zorn, “The Measured Cost of Conservative Garbage Collection”, Tech Report, University of Colorado at Boulder, 1992.

Gianluca Insolvibile has been a Linux enthusiast since kernel 0.99pl4. He currently deals with networking and digital video research and development. He can be reached at g.insolvibile@cpr.it.

Archive Index Issue Table of Contents

Advanced search

ChessBrain: a Linux-Based Distributed Computing Experiment

Carlos Justiniano

Issue #113, September 2003

If one computer already beats you at chess, wait until 646 of them gang up on you.

On May 27, 2003, 646 machines worked together to play a single game of chess. This was the first time such a feat had been accomplished, and it was made possible by the power of Linux, open-source software and hundreds of contributors from over 37 different countries.

ChessBrain (chessbrain.net) is a distributed computation project that uses the idle processing power of distributed machines to solve computationally intensive problems. ChessBrain is a system focused on playing chess, but the underlying system can be adapted for other games as well as for non-game-related applications.

Imagine playing a game against an opponent, except every time he moves, you grab the phone and start calling friends for help. You call Sue, describe the current position and ask her to call you back when she has an answer. Then you call Ryan to ask whether you should worry about a pending attack; again, you ask for a call back when he has an answer. After calling 20 other friends, you sit back and wait for replies. This is similar to how ChessBrain plays chess.

ChessBrain consists of a Linux-based server application, the SuperNode, and client software known as PeerNodes. The SuperNode connects to an on-line game server, which allows visiting members to play against one another, challenge ChessBrain to a game or watch ChessBrain play against its current opponent. While ChessBrain plays, it examines positions, dispatches hundreds of potential moves to remote PeerNodes for analysis, collects feedback from the PeerNodes, processes that information and makes its best move.

ChessBrain exists as an ever-changing pool of networked machines. Philosophically and scientifically, it's a beautiful thing.

I started ChessBrain as a distributed computing experiment in the summer of 2001. By the end of that year, I had a working prototype and needed a place to host the server. My longtime friend, Walter Howard, the webmaster of HackerWhacker (hackerwhacker.com), offered to host the server on his personal T1 line.

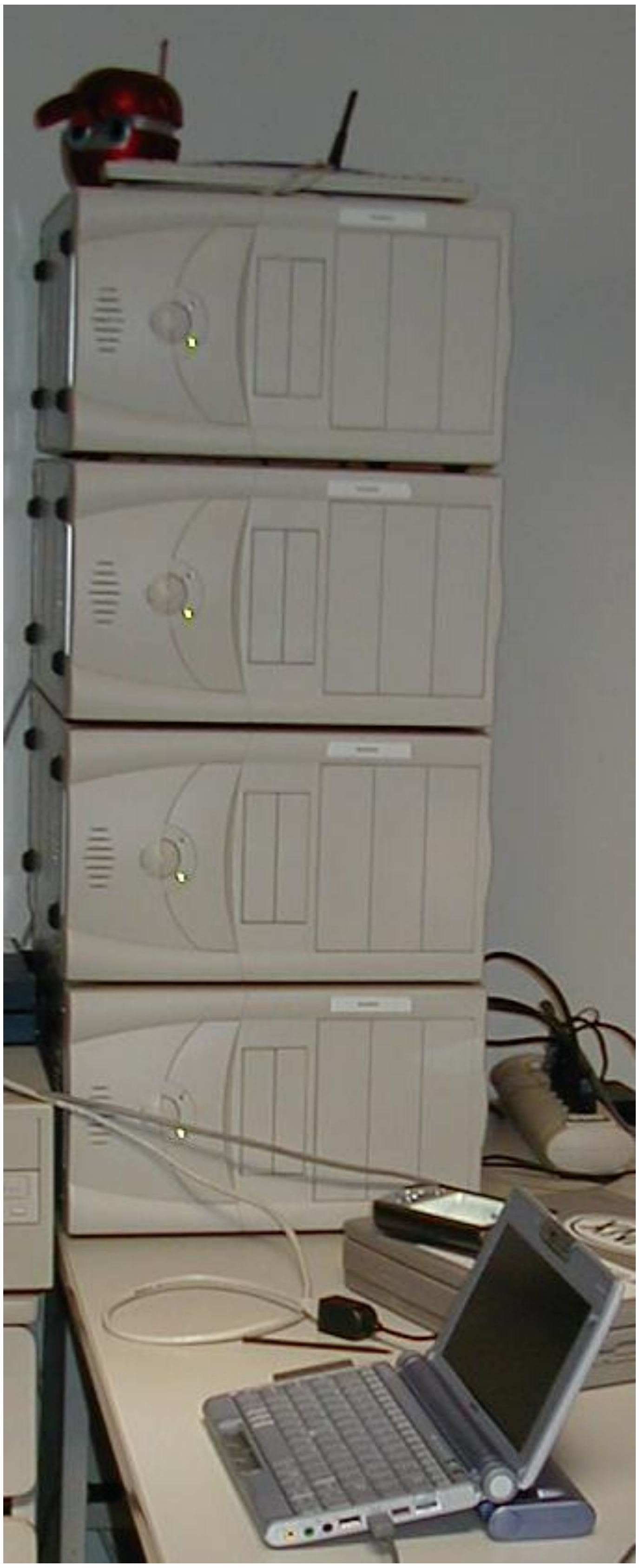


Figure 1. The First ChessBrain Servers

On June 9, 2002, ChessBrain appeared on Slashdot, and the positive exposure resulted in hundreds of new PeerNode operators. Gavin Roy, one of the new members, owns the bteg network (www.bteg.net) and offered to host a SuperNode server free of charge. On June 27, I met Gavin for dinner and handed this near stranger a SuperNode server on a Pentium III machine. ChessBrain gained another server, I gained another friend, and Gavin has become an important supporter of the ChessBrain Project. I transitioned the SuperNode over to Gavin's site, and Walt continued to host the original SuperNode as a secondary backup and experimental server.

During the months that followed, we gained an amazing amount of exposure. Few seemed to mind that ChessBrain couldn't actually play chess. The first eight months of 2002 were spent working on the SuperNode server and porting the PeerNode client to Microsoft Windows and Apple's Mac OS X.

Once the server and clients worked well, the focus was on getting ChessBrain actually to play. The wbec-ridderkerk (www.wbec-ridderkerk.nl) site in the Netherlands lists nearly 200 freely available chess-playing programs. I reviewed a few, looking for one with relatively clean code and the ability to compile under several operating systems. I found an ideal program in Beowulf, written by Colin Frayn, who was then a PhD candidate at Cambridge University in England. We exchanged several e-mails and Colin joined the project. We collaborated entirely on-line using e-mail and instant messaging (IM) and began making necessary modifications. Colin adapted his chess program for distributed computing, and I modified the SuperNode and PeerNode clients to use his engine. The time difference between London and Los Angeles proved ideal. I would IM Colin at my 3AM and again during the day. By my late afternoon, Colin would head for bed, and I would work through his night. Before crashing, I would leave feedback for Colin. This round-the-clock cycle continued for months.

Colin adapted his original Beowulf chess engine to become two chess-playing components, BeoServer and BeoClient. He developed the pair to support distributed chess play within the ChessBrain framework. On December 22, 2002, ChessBrain played its first game of distributed chess. By January 2003, the ChessBrain community had provided 62 machines and was testing regular builds.

Overview

The SuperNode and PeerNode are multithreaded applications written in C++ and compiled using GCC under Red Hat Linux 7.1, 7.2 and 8.0. The primary SuperNode server runs under Slackware 8.0 at bteg network's colocation site in Northern California (Figure 2).

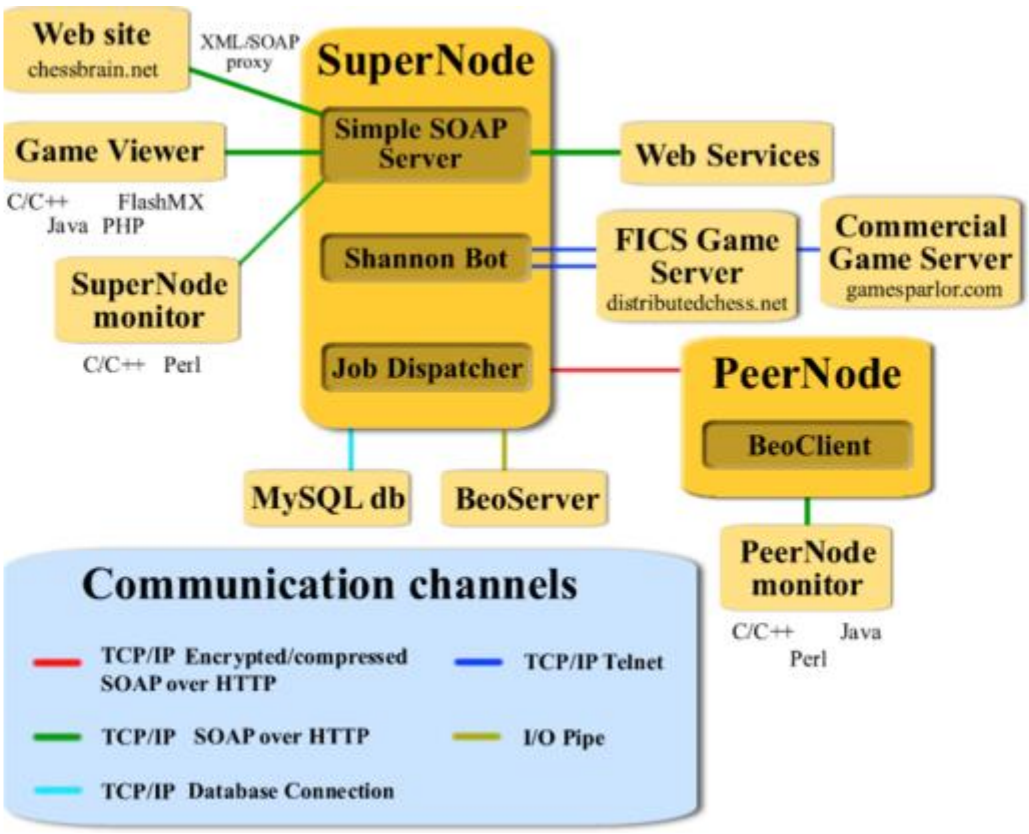


Figure 2. The ChessBrain System Architecture

Because the applications are heavily multithreaded, I spent a fair amount of time resolving threading issues. I used GDB, DDD and custom logs to tackle debugging. Early in the development process, Perl scripts proved especially effective in helping test new functionality and stress test the software. I have 12 machines at home; these, plus an army of Perl scripts pounding on a local server, proved to be formidable testing tools.

XML, SOAP and Web Services

Early in the project I realized the SuperNode server would need to communicate with other servers. During that time XML offered a viable approach, and later XMLRPC (www.xmlrpc.org) brought additional advantages. The Simple Object Access Protocol (SOAP) continued evolving to meet the needs of servers that speak to other servers. Encouraged by promises of improved interoperability, I adopted SOAP as the preferred method of communication for the SuperNode server and PeerNode client.

From the outside, the SuperNode acts like a Web server with SOAP-based interfaces. Although the SuperNode server handles HTTP GET and POST, the POST message is used most often. The SuperNode parses HTTPs and XML-based SOAP requests, processes those requests and returns HTTP packages with embedded SOAP payloads.

The SuperNode and PeerNode parse SOAP requests and route commands to an internal command dispatcher, which ensures that the correct command handlers process the requests. In the SuperNode, the most common requests come from PeerNode clients; a PeerNode must connect to request a job unit. A job unit is an XML block containing a game position and instructions on how to analyze the position. A PeerNode contains a complete chess engine component, compiled and linked as a static library. When the PeerNode receives a job unit, it processes the SOAP response, extracts the job-specific information and passes instructions to its internal chess component for analysis.

The SuperNode server then passes the current game position to the external BeoServer process. Interprocess communication between the SuperNode and BeoServer is accomplished using two pipes. In the near feature, we expect to move BeoServer to its own box and shift to UDP over 1000Base-T Ethernet.

Security

Secure and tamper-free communication is a necessity for ChessBrain. An invalid result created by a malicious user could render the play ineffective and ultimately embarrassing. Sensitive communication is protected using the Advanced Encryption Standard, AES Rijndael (pronounced Rhine-doll). AES is a variable block symmetric encryption algorithm developed by Belgian cryptographers Joan Daemen and Vincent Rijmen as a replacement for the aging DES and Triple DES standards.

Before exploring Rijndael, the Blowfish symmetric cipher was used until the PeerNode client was ported to Mac OS X and problems surfaced involving endian issues with the implementation of Blowfish being used. AES is an endian-neutral algorithm and proved ideal for our situation.

The original design of the PeerNode involved having the client and its chess engine as two separate processes. The PeerNode started the chess engine process and redirected the standard I/O to establish a loose binding. Initially, we avoided directly linking chess code with the PeerNode client so the chess code could be replaced quickly and easily in future iterations of the software. We later moved to a static linking approach to deal with potential security issues. The problem was that it's entirely possible to write a chess engine proxy that sits between the PeerNode and the actual chess engine program. This would offer an easy way to alter results before sending them to the SuperNode server. We decided to link the engine component statically because of two key advantages, tighter security and function-based rather than I/O-based messaging.

The surge of interest from Slashdot soon made it necessary to reduce ChessBrain's bandwidth requirements. To this end, the use of SOAP offered many advantages, but its size left much to be desired. The Zlib data compression library (www.zlib.org) is now used prior to encryption to reduce the size of SOAP-based messaging. Naturally, adding compression and encryption reduces the potential for interoperability; however, the XML encryption specification (www.w3.org/TR/xmlenc-core) offers an alternative approach.

Bots, Presence and Autonomous Play

The SuperNode server has a Bot called Shannon (implemented as a thread) that connects to on-line game servers and maintains a presence. Members of the game server type commands to challenge ChessBrain or to watch the current game being played. It has been fun programming Shannon, which now understands a variety of commands. There is a great deal of potential in on-line bots that can be instructed to perform actions on behalf of their hosts.

During the development of ChessBrain, I downloaded the source code to a Free Internet Chess Server (FICS) and compiled it on an old Pentium 200 MMX Toshiba laptop running Linux. FICS is written in C, and it compiled using GCC without incident. The game server allows users to telnet to port 5000 and sign in with a user name and password. After a few months the traffic increased, and we moved the FICS server to another ChessBrain machine at our secondary domain at distributedchess.net. Users now have several options for watching ChessBrain play on-line. They can telnet directly to the game server where ChessBrain is playing or use one of our viewer programs.

After ChessBrain could play on an on-line game server, I wrote a Java game viewer to allow people to watch live games. As an alternative to the Java viewer, I also wrote viewers based in PHP and Macromedia Flash (www.chessbrain.net/viewers.html). ChessBrain contributor Anthony Bravo wrote a Java-based network viewer to show the active PeerNodes throughout the world. Users can click on nodes to see how many machines are active in a given country. All of the viewers on the ChessBrain site use SOAP to communicate with the SuperNode.

As a security precaution, browser plugins such as Java and Macromedia's Flash ActionScript don't allow the program to connect to a server other than the one from which the applet was downloaded. To work around this issue, I wrote a simple XML proxy script that accepts an HTTP GET request on one server and connects to the SuperNode server on behalf of the client. For example, if you wanted to query the SuperNode server for the current game position, you could enter the following URL into your browser: `http://www.chessbrain.net/xmlproxy.php?command=CBSGetPos`. The server would respond with a SOAP package like the one in Figure 4. On Mozilla you can view the page source to see the actual SOAP document.

Listing 1. A ChessBrain XML Response Package

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env=
  http://www.w3.org/2001/12/soap-envelope
  xmlns:enc=
    "http://www.w3.org/2001/12/soap-encoding">
  <env:Body>
    <cbs:CBSGetPosResponse
        >
      <return>
        rn1qk2r/p2p1ppp/bb2pn2/1p6/1P6/
        P2Q1NP1/1BP1PP1P/RN2KB1R b KQkq -
      </return>
    </cbs:CBSGetPosResponse>
  </env:Body>
</env:Envelope>
```

Monitoring the SuperNode

Monitoring a server's health is an important part of system administration. Fortunately for developers, Linux offers many ways to tackle server monitoring. The Linux /proc virtual filesystem contains a goldmine of valuable system data, offering developers an easy way to profile and monitor system behavior. /proc/net/dev offers device data such as the number of bytes and packets sent and received on a network interface, and /proc/meminfo offers loads of memory statistics. If data mining the /proc isn't your thing, sysinfo() offers a quick and easy way to fill a structure with system statistics, such as system load, freeram and the total number of processes.

The SuperNode server offers a SOAP request that returns system information similar to what is shown in Listing 2. ChessBrain member Greg Davis wrote the first SuperNode monitor in Perl, which issues the SOAP request and displays a screen similar to the **top** command.

Listing 2. An XML Server Status Message

```
<?xml version="1.0" ?>
<env:Envelope

>
  <env:Body>
    <cbs:CBSSysInfoResponse
      >
    <Uptime days="1" hours="14"
            minutes="43" seconds="18" />
    <System proccnt="546" totmem="250.13"
            freemem="4.38"
            memu="98"
            cpustates="3627078,0,2151891,8160852"
            loadavg="0.50,0.30,0.33" />
      <Recv Bytes="2301480887.000000"
            Packets="16652816.000000"
```

```
               Errors="0.000000"
               Drop="0.000000" />
         <Send Bytes="2443488824.000000"
               Packets="12142245.000000"
               Errors="0.000000"
               Drop="0.000000" />
        </cbs:CBSSysInfoResponse>
      </env:Body>
   </env:Envelope>
```

PeerNode Monitoring

Because many members of the ChessBrain community run PeerNode clients on several machines, they wanted a convenient way to monitor the status of a group of machines. The PeerNode client was modified to post SOAP requests to port 3434 so PeerNode monitor applications could listen on that port and display real-time status information. I wrote the first PeerNode monitor application, and other members submitted their own. Greg Davis and Oliver Otte both submitted Perl-based PeerNode monitors. The most popular PeerNode monitor, CBMoc, was written in Java by Kris Drent.

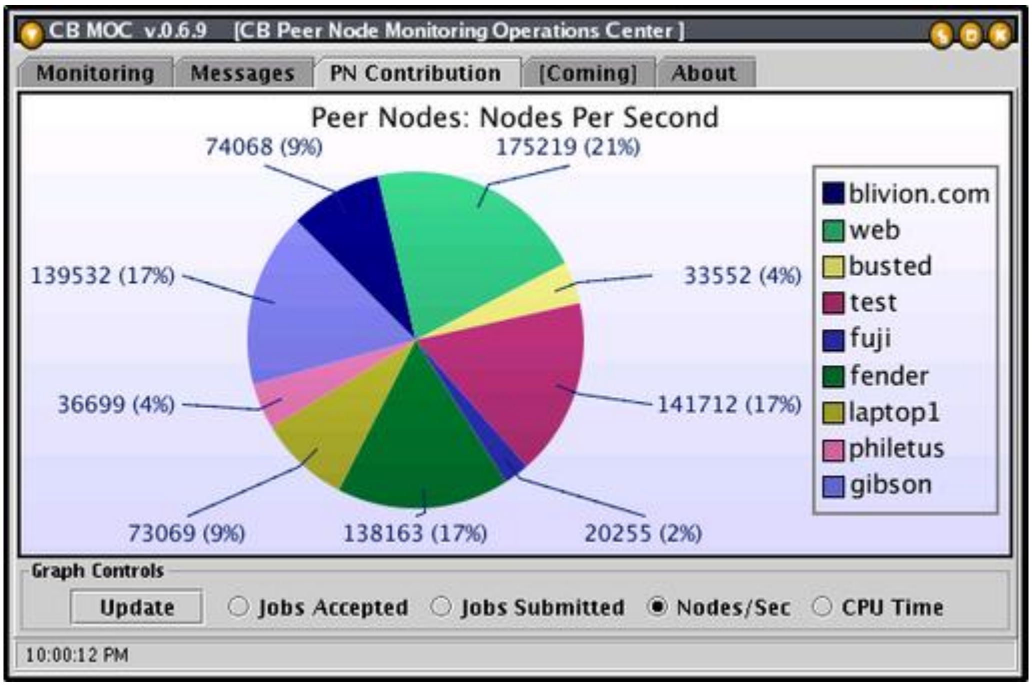


Figure 3. Java-Based CBMoc

Cross-Platform Graphics and Data Visualization

ChessBrain collects a great deal of data, and we're currently exploring data visualization as a way of tracking and investigating system behavior. We're adding 3-D-based game navigation tools to allow us to track network play visually. Sven Herrmann, our resident 3-D expert, has created an OpenGL-based renderer that we use in our SuperNode monitor application. We'll also use the new renderer in our next-generation screensaver program and game viewers.

Figure 4. Real-Time OpenGL Rendering

Conclusion

Today, ChessBrain is a working prototype that plays chess using hundreds of machines running Linux, FreeBSD, Mac OS X and Microsoft Windows. It plays at international master strength, and we see many ways to continue making improvements.

ChessBrain exemplifies the use of open-source tools to solve complex problems. We're preparing ChessBrain itself for release as an open-source project, with the hope that others will join and expand the effort. The wonderful thing about ChessBrain is that there are so many ways to contribute. Anyone with a PC and Internet connection can participate. Download a PeerNode client from the ChessBrain Web site, run the software on one or more computers and help increase the size of ChessBrain.

We're currently working toward an official world record for the largest number of machines used to play a single game. We have an established contact at the World Record office in London and contacts within a number of official chess federations. The ChessBrain Project is supported by a strong core team, including Peter Wilson, the former chairman of the World Chess Federation's (FIDE) Computer Chess and Internet Committee. The ChessBrain team currently is exploring potential venues for a public and Internet-based demonstration involving a new Guinness World Record in distributed computation and chess. Check the ChessBrain site for an official announcement.

Acknowledgements

Many thanks to Janus Daniels, Cedric Griss and other ChessBrain team members for their support in the preparation of this article. For contact information and a list of who's who on ChessBrain visit, www.chessbrain.net/friends.html.

Carlos Justiniano is a 20-year veteran of the software industry. ChessBrain allows him to live several mantras: “Embrace complexity”, “Engage in nontrivial pursuits” and “Do it using open-source software”! Comments on this article can be sent to cjus@chessbrain.net.

Archive Index Issue Table of Contents

Advanced search

Put a Sump Pump on the Web with Embedded Linux

Tad Truex

Issue #113, September 2003

Use a simple circuit to determine if an AC-powered device is on or off, and put the information on the Web.

My wife and I bought our house in the summer of 1996. In early spring of 1997, the basement filled with water. In early spring of 1998, the basement filled with water. In early spring of 2001, the basement filled with water. Fed up with the basement filling with water, we decided it was time to install a sump pump.

Late in 2003, looking up the hill at more than four feet of snow waiting to flood our basement, we realized we never got around to installing the sump pump we decided to get in 2001. This time we vowed we really were going to do something, and we did. After a week of jackhammering the basement, we had ourselves a shiny new sump pump complete with a perimeter drain, battery backup and snazzy cover. Not trusting the ability of this little pump to eject all that melting snow, I become obsessed with the sump. I checked the water level every ten minutes. I woke up at night to check the water level. I called home from work to get a report—it was getting absurd. The very thing that was supposed to make me sleep easier was suddenly consuming me. Then it dawned on me: why not rig up a contraption that would let me monitor the pump remotely? I established some goals for the project: 1) do not in any way burn the house down, 2) do not in any way cause the pump to stop working, and 3) learn something.

In accordance with my primary and secondary goals, I decided I would not place any new circuitry in series with the pump's power. In addition to the obvious dangers of running 10 amps through a circuit I constructed, a fair amount of isolation circuitry would be required to ensure I didn't fry the processor to which I was going to connect. One option was to wrap a coil of wire around the current-carrying conductor in the pump's cord. Presumably, after suitable signal conditioning, the induced voltage would be detectable at the processor. Unfortunately, given that my home electronics lab is pretty meager, this method likely would take too long to develop.

After rejecting several other ideas, I went to Google. Eventually, I was reminded of a phenomenon known as the Hall Effect. The Hall Effect is a manifestation of the Lorentz force acting on electrons flowing in the presence of a magnetic field. The Lorentz force acts normal to both the electric and magnetic fields, causing the electrons to have a non-uniform distribution in the direction of the Lorentz force. The voltage induced on the surface of the conductor in this direction is proportional to the magnetic field strength and therefore can be used to detect its strength. A wide variety of Hall Effect sensors are available commercially, differing in the amount of signal conditioning they provide internally and the magnetic field strengths to which they are sensitive. For this application, I chose the Allegro Microsystems A3240LUA, a fairly sensitive unipolar sensor; a datasheet is available at www.allegromicro.com/sf/3240. A unipolar sensor basically acts as an NPN transistor whose base current is on when the device is in the presence of a south magnetic pole.

I acquired a few of the sensors with which to experiment. My hope was the remote sensor would consist of nothing more than the Hall effect device connected to a general-purpose input/output (GPIO) pin on the processor. To ease the software debugging, I decided to construct a standalone circuit that would indicate whether the pump was on by way of an LED. This would boost my confidence that the sensor was capable of detecting the current. I constructed the circuit shown in Figure 1.

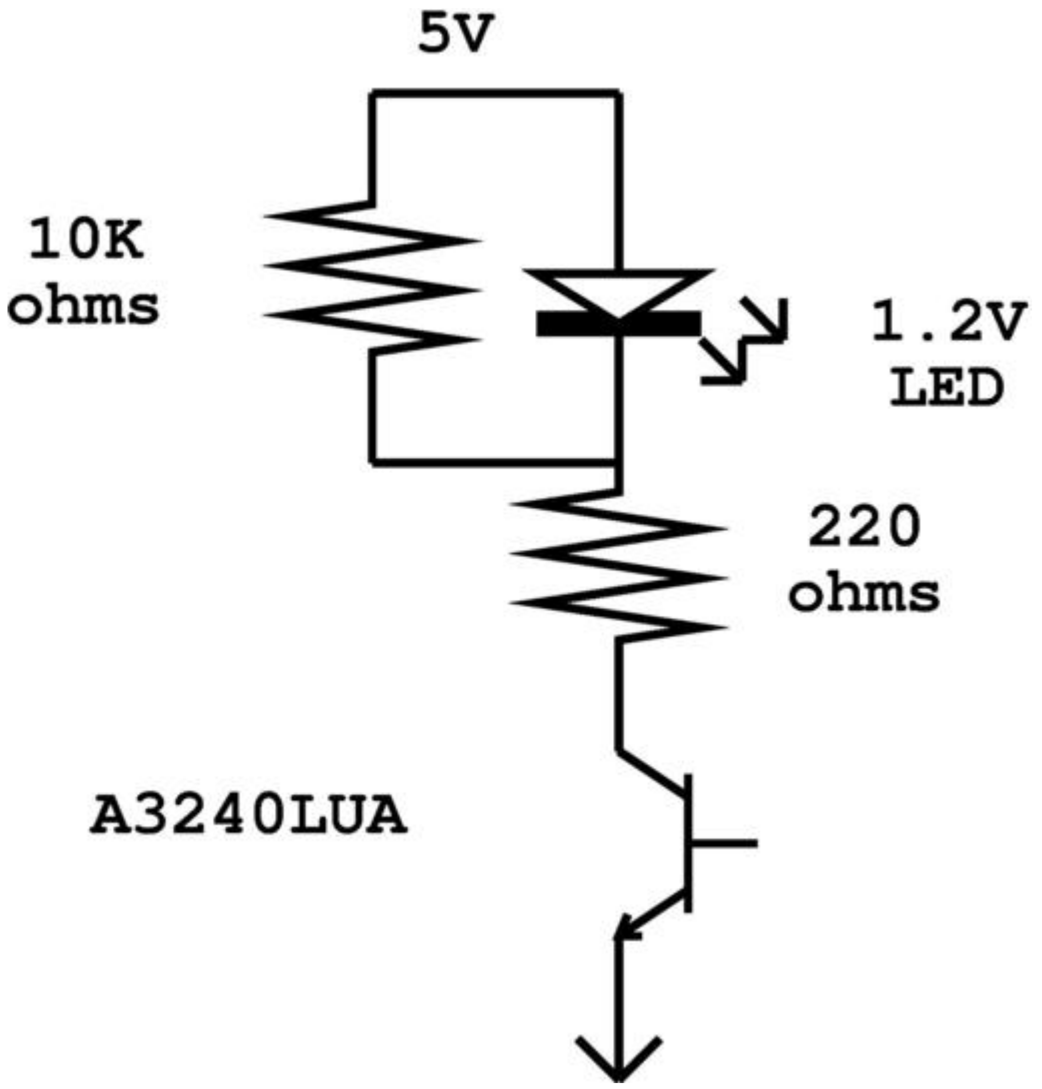


Figure 1. Schematic of the Test Circuit

I plugged the batteries in, waved a magnet in front of the sensor and, as expected, the LED came on. The next step was to wait for the pump to come on and wave the sensor near the power cord to see if it could detect the near field. Wait, wave, nothing; wait, wave, nothing. It appeared that the near-field/far-field boundary was much closer to the conductors than I had originally thought. The magnetic fields of the hot and neutral leads in the power cord were close enough to cancel one another, so I couldn't detect them. No matter where I placed the sensor, I couldn't detect the current. Not to be dissuaded, I came up with a slight modification to the plan. Armed with an old 15-amp extension cord and a piece of soft iron core, I set about strengthening the field by wrapping about ten turns of the neutral lead around the iron core. A couple of wire ties and a bit of epoxy completed the job. With my modified cord, I headed down to the sump. After plugging the pump into the extension cord, I waited for some activity. When the pump came on this time, I was able to get the test sensor close enough to the iron core to detect the field. With a bit of perf board, some solder and a bit more epoxy, the final sensor was completed. Figure 2 shows the finished sensor.

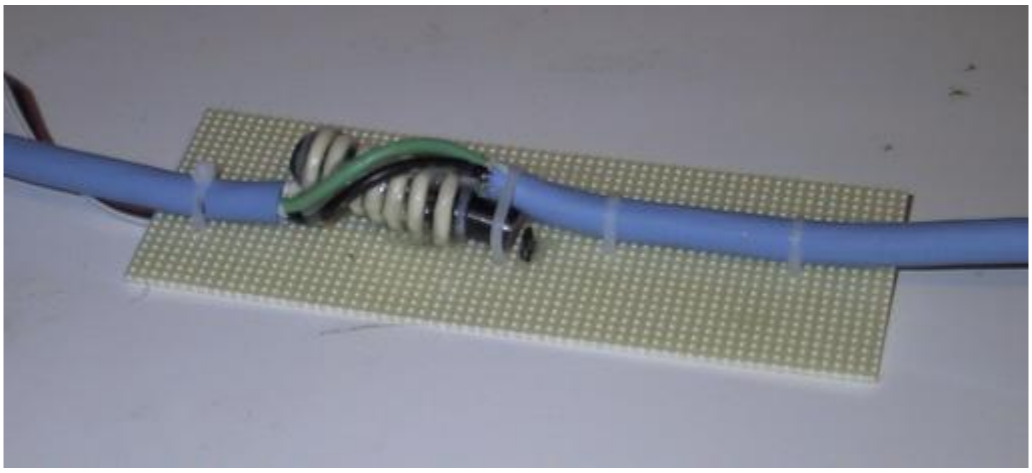


Figure 2. The Finished Sensor

The next decision to make was what sort of a processor to use. The main considerations were price, presence of onboard Ethernet, available GPIOs and the ability to run Linux. After searching for a while, I came to the conclusion that a number of embedded microcontrollers were available that had Ethernet and enough horsepower for this task, but most didn't explicitly mention Linux. At the other end of the spectrum were the PC/104 class embedded PCs, which significantly offered more muscle and cost than was needed for this project. I ended up with a Net4501 from Soekris Engineering, a single-board computer with a CompactFlash socket, 64MB of RAM, an AMD Elan processor and three onboard Ethernet controllers. As an extra bonus, the board has the ability to boot over the network through the Preboot Execution Environment (PXE).

The documentation on the Soekris Web site (www.soekris.com) made it clear that several of the Elan's GPIO pins were available on a readily accessible header, along with a +5V supply. The price was quite reasonable and included a power supply and a nifty metal case to hold the board. Each of Elan's GPIO pins has an internal pull-up or pull-down. Selecting one with an internal pull-up allowed me to connect the bare sensor to the CPU without any other components.

Next, I built a recent (2.4.19) kernel capable of being network-booted and disabled almost everything. I built the kernel with no modules and enabled the Natsemi Ethernet driver, root NFS, serial console and the SC520 watchdog timer. In addition to the normal configuration process, an additional change to the kernel was required for this project. In the 2.4 series of kernels, the default timer interrupt for x86 is set to 100Hz. Because I knew I'd be sampling a signal running at nearly the same frequency (60Hz), I decided to increase the timer frequency. The interrupt timer frequency is controlled by the Hz define in asm/param.h. The upper boundary on this value is 2K, so I set mine to 1,500 to provide me with 1,500 timer interrupts per second. With little else running on the machine, there would be no real likelihood of timer-based routines interfering with one another due to the increased frequency.

I configured my DHCP server and PXELINUX to serve up the previously built kernel. All that was left was to populate the root filesystem for the TFTP sever. To create the initial runtime environment, I built the latest uClibc, BusyBox, TinyLogin and utelnetd packages. I statically linked all three executables against uClibc. BusyBox's version of init starts up a shell on the console port by default. To add other features, I added my own /etc/inittab. It enables the console shell and invokes a simple init script that initially (re)mounts the root filesystem, enables the watchdog and starts a telnetd so I can connect remotely. With a terminal connected to the serial console port, I rebooted the device. Through the console port I was able to see the system load and eventually drop into the BusyBox version of the shell.

Once the system was up and running, it was time to focus on the new drivers. The only kernel-space device driver needed for this project is something to monitor the GPIO pin to which the sensor is connected. Because I am new to kernel programming, I decided to minimize the likelihood of anything going wrong in the kernel by limiting this driver to the smallest possible amount of code. To that end, I decided to write a /proc filesystem driver that simply would report the state of the pump as on or off. Any other work that needed to use user-space programs would poll the low-level driver.

The driver's init function performs three main tasks. First, it registers itself as a /proc filesystem module with the create_proc_entry call. The two important structures the proc_dir_entry returned are the file and inode operations tables, which are set to two static structures with entries filled out appropriately. Because this is such a simple module, the vast majority of the entries in both structures are NULL. Once the proc entry has been created, the init routine pokes a few Elan-specific registers to make sure the desired GPIO is set as an input.

The last thing the initialization routine does is kick off a timer to check the pin at regular intervals. The timer function bears a bit more exploration:

```
static void sample( unsigned long data ){

  static int remaining_samples = 0;
  __u16 dat;
  int ntimeout = 1;
  int mtimeout = Hz*mper - nsamp*ntimeout;

  if ( remaining_samples ){
    /*
     * Take another microsample
     */
    timer.expires = jiffies + ntimeout;
    remaining_samples--;
    dat = readw( pio_dat ) & DMASK;
    if ( dat == 0 ){
      /* low true logic at the pin */
      sample_buf[remaining_samples] = 1;
    }
    add_timer( &timer );
  }
  else {
    /*
     * We have accumulated a full buffer worth of
     * samples.  Decide if the pump is on.
     */
    int i;
    char buf[MAXSAMPLES];

    pump_state = 0;

    for( i = 0; i < nsamp; i++ ){
      /* itegrate the pin signal */
      pump_state += sample_buf[i];

      /* convert to a printable buffer for
         osciloscope mode */
      buf[i] = sample_buf[i] + '0';

      /* clear the buffer for next time */
      sample_buf[i] = 0;
    }
    buf[i] = '\0';

    if((verbose==1 && pump_state) || (verbose>1) ){
      /* print the signal trace */
      printk( KERN_INFO
              "Sample buffer at tick (%ld) %s\n",
              jiffies, buf );
    }

    /* long tick between samples */
    remaining_samples = nsamp;
    timer.expires = jiffies + mtimeout;
    add_timer(&timer);
  }

}
```

Because the magnetic field, and therefore the output of the sensor, oscillates, I couldn't simply sample the state of the pin and report it as the state of the pump. To avoid introducing any noise in the statistics as a result of inopportune sampling, I implemented a crude integrator. When the timer initially fires, the remaining sample count is reset to the number of per-period samples desired. In the default case, we set this value to Hz/60 or 25 samples per period. Remember that Hz is the frequency of the kernel timer interrupt, which I increased to 1,500 when I built the kernel.

At the end of the 25 fast samples, I reset the timer to expire again at the end of the macro-sampling interval. In the default case, I take a macro sample once every five seconds. While resetting the timer, I also integrate (add) the number of active fast samples. Because I am sampling fairly quickly, I can be certain to detect that the pump is on. When I do detect that the pump is on, I set the pump_state variable. When the module is read (in the pump_output function), I simply examine the state of this variable and report it. This mechanism allows me to twiddle around with the sample timing without affecting the response time of the driver.

While debugging the driver, I added a verbose parameter to print various tidbits to the log file. Running the module with verbose=1 passed as a parameter causes it to dump the sample buffer whenever it thinks the pump is on. This provides a simple oscilloscope function, allowing me to check the log file periodically to confirm I am not receiving any spurious results. It also provides a second, somewhat interesting ability. Knowing the trip points of my sensor and the phase angle (time) between the two points, I can calculate the peak magnetic field strength at the sensor. At 1,500Hz, the sensor is on for five samples. Given that the field oscillates at 60Hz, this gives me a phase angle of 0.4PI radians between samples. The equations below can be solved for my sensors' worst-case trip points (on at 50 Gauss, off at 5) for a peak amplitude of 51.4 Gauss. Using the typical values (35 and 25) indicated that the field is likely around 38G peak:

A * sin(theta) = 35

A * sin(theta + 0.4PI) = 25

Another debugging aid I wrote is a similar driver that activates an onboard LED, also connected to a GPIO, when instructed to do so. This driver is similar to the pump driver, except the output function (the read *from* the module) doesn't require any sophisticated sampling, and the driver now has an input (a write *to* the module) that allows a user to set the state of the GPIO pin. The new function (led_input) reads a buffer from user land and decides whether it needs to set, clear or toggle the current state of the pin. This new function is registered using the file_operations struct. The only other structural difference between this driver and the pump driver is that file permissions (specified in the create_proc_entry call) must allow write access. This driver coupled with a simple shell script provides feedback at the pump that the software is working —if the LED state tracks the pump state, everything is up and running.

With the basic drivers in place, it was time to build a useful system out of all the parts. The first piece of user-space code needed was a dæmon to enable remote queries to determine the pump state. As my root filesystem is mounted with NFS from my main server, I could run a simple shell script that would sleep for a while, check /proc/pump and update a real file with the results. But, instead of taking the easy way out, I wrote pumpserv.

pumpserv is a simple dæmon that accepts a connection on port 5678 and copies the entire contents of /proc/pump to the caller. At the other end of the pipe is pumpwatch. pumpwatch is another dæmon that runs on the host computer and checks the pump periodically to record the time of each state change. The transitions are timestamped and dumped to a log file. The log file then can be post-processed with any statistical means desired, or it can be uploaded to a spiffy Web site for the world to view.

The system has been up and running since April 2003. Given that it seems to work fine, I probably should call it a victory and move on, but I can't help pondering pumpserv2. If I ever get around to it, I will do a few things differently. A basic flaw with the current system is it requires an NFS server to serve up the root filesystem and more importantly, it requires a dæmon running on the server to capture the data. A much more robust system would have the root filesystem local to the pump monitor; the Soekris Net4501 has a CF slot, so this should be easy. It also would be desirable to have the dæmon on the pump monitor log the data locally and provide this data when requested. This way, if the main server ever goes down, none of the data is lost.

This basic technique could be modified easily to monitor other devices that draw enough current to trip the sensor. Some possibilities include air conditioners, refrigerators, hottub heaters, well pumps, beer coolers—the possibilities are endless. Perhaps the most useful application of this technique is keeping track of the all-important coffee level in the office coffeepot, because the frequency at which the heater cycles is inversely proportional to the volume of remaining coffee.

For anyone interested in implementing something along these lines, the init script and all of the source code for both drivers, pumpwatch and pumpserv is available at pumps.oldtools.org/src. A compressed tarball of the pump monitor's filesystem also is available. Anyone interested in seeing whether my basement is on its way to a flood can check out pumps.oldtools.org.

The source code is also available from the *Linux Journal* FTP site at ftp.linuxjournal.com/pub/lj/listings/issue113/6827.tgz.

During the day, Tad Truex designs Alpha microprocessors for HP. At night, he tries to juggle his two kids and numerous hobbies.

Archive Index Issue Table of Contents

Advanced search

Kernel Korner

Exploring Dynamic Kernel Module Support (DKMS)

Gary Lerhaupt

Issue #113, September 2003

Manage modules separately from the kernel with a simplified delivery system, and make your package manager more useful.

Source is a wonderful thing. Merged module source in the kernel tree is even better. Most of all, support for that source is what really counts. In today's explosion of Linux in the enterprise, the ability to pick up the phone and find help is critical. More than ever, corporations are driving Linux development and requirements. Often, this meets with skepticism and a bit of anxiety by the community, but if done correctly, the benefits are seen and felt by everyone.

The dynamic kernel module support (DKMS) framework should be viewed as a prime example of this. DKMS, a system designed to help Dell Computer Corporation distribute fixes to its customers in a controlled fashion, also speeds driver development, testing and validation for the entire community.

The DKMS framework is basically a duplicate tree outside of the kernel tree that holds module source and compiled module binaries. This duplication allows for a decoupling of modules from the kernel, which, for Linux solution and deployment providers, is a powerful tool. The power comes from permitting driver drops onto existing kernels in an orderly and supportable manner. In turn, this frees both providers and their customers from being bound by kernel drops to fix their issues. Instead, when a driver fix has been released, DKMS serves as a stopgap to distribute the fix until the code can be merged back into the kernel.

Staying with the customer angle for a bit longer, DKMS offers other advantages. The business of compiling from source, installing or fidgeting with rebuildable source RPMs has never been for the faint-of-heart. The reality is that more Linux users are coming in with less experience, necessitating simpler solutions.

DKMS bridges these issues by creating one executable that can be called to build, install or uninstall modules. Further, using its match feature, configuring modules on new kernels could not be easier, as the modules to install can be based solely on the configuration of some kernel previously running. In production environments, this is an immense step forward as IT managers no longer have to choose between some predefined solution stack or the security enhancements of a newer kernel.

DKMS also has much to offer developers and veteran Linux users. The aforementioned idea of the decoupling of modules from the kernel through duplication (not complete separation) creates a viable test bed for driver development. Rather than having to push fixes into successive kernels, these fixes can be distributed and tested on the spot and on a large scale. This speedup in testing translates to an overall improvement in the speed of general development. By removing kernel releases as a blocking mechanism to widespread module code distribution, the result is better tested code that later can be pushed back into the kernel at a more rapid pace—a win for both developers and users.

DKMS also makes developers' lives easier by simplifying the delivery process associated with kernel-dependent software. In the past, for example, Dell's main method for delivering modules was RPMs containing kernel-specific precompiled modules. As kernel errata emerged, we often were taken through the monotonous and unending process of recompiling binaries for these new kernels—a situation that no developer wants to be in. However, Dell still favored this delivery mechanism because it minimized the amount of work and/or knowledge customers needed to have to install modules. With DKMS, we can meet these usability requirements and significantly decrease our workload from the development standpoint. DKMS requires module source code to be located only on the user's system. The DKMS executable takes care of building and installing the module for any kernel users may have on their systems, eliminating the kernel catch-up game.

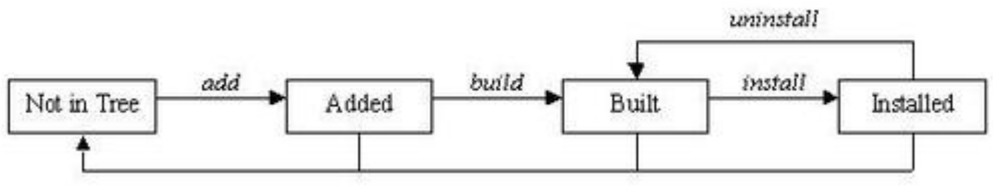


Figure 1. Module States in DKMS

Using DKMS

With all of this up-front hype about DKMS, perhaps it might be best to settle into the particulars of actually how the software is used. First, using DKMS for a module requires that the module source be located on the user's system and that it be located in the directory /usr/src/*(module)*)-(*(module-version)*)/. In addition, a dkms.conf file must exist with the appropriately formatted directives within this configuration file to tell DKMS such things as where to install the module and how to build it. More information on the format of the dkms.conf file can be found later in this article. Once these two requirements have been met and DKMS has been installed on the system, the user can begin using DKMS by adding a module/module-version to the DKMS tree. The example add command:

```
   dkms add -m qla2x00 -v v6.04.00
```

would add qla2x00/v6.04.00 to the extant /var/dkms tree. This command includes creating the directory /var/dkms/qla2x00/v6.04.00/, creating a symlink from /var/dkms/qla2x00/v6.04.00/source to → /usr/src/qla2x00-v6.04.00/ and copying the dkms.conf file from its original location to /var/dkms/qla2x00/v6.04.00/dkms.conf.

Once this add is complete, the module is ready to be built. The dkms build command requires that the proper kernel sources are located on the system from the /lib/module/*kernel-version*/build symlink. The make command used to compile the module is specified in the dkms.conf configuration file. Continuing with the qla2x00/v6.04.00 example:

```
   dkms build -m qla2x00 -v v6.04.00 -k 2.4.20-8smp
```

compiles the module but stops short of installing it. Although build expects a kernel-version parameter, if this kernel name is left out, it assumes the currently running kernel. However, building modules for kernels not currently running also is a viable option. This functionality is assured through the use of a kernel preparation subroutine that runs before any module build is performed. This paranoid kernel preparation involves running a `make mrproper`, copying the proper kernel .config file to the kernel source directory, running a `make oldconfig` and, finally, running a `make dep`. These steps ensure that the module being built is built against the proper kernel symbols. By default, DKMS looks for the kernel .config file in the /lib/modules/*kernel-version*/build/configs/ directory, utilizing Red Hat's naming structure for those config files. If the kernel .config file is not located in this directory, you must specify a --config option with your build command and tell DKMS where the .config file can be found.

Successful completion of a build creates, for this example, the /var/dkms/qla2x00/v6.04.00/2.4.20-8smp/ directory as well as the log and module subdirectories within this directory. The log directory holds a log file of the module make, and the module directory holds copies of the compiled .o binaries.

With the completion of a build, the module now can be installed on the kernel for which it was built. Installation copies the compiled module binary to the correct location in the /lib/modules/ tree, as specified in the dkms.conf file. If a module by that name is already found in that location, DKMS saves it in its tree as an original module, so it can be put back into place at a later time if the newer module is uninstalled. The example install command:

```
dkms install -m qla2x00 -v v6.04.00 -k 2.4.20-8smp
```

creates the symlink /var/dkms/qla2x00/v6.04.00/kernel-2.4.20-8smp → /var/dkms/qla2x00/v6.04.00/2.4.20-8smp. This symlink is how DKMS keeps tabs on which driver version is installed on which kernel. As stated earlier, if a module by the same name is installed already, DKMS saves a copy in its tree in the /var/dkms/*module-name*/original_module/ directory. In this case, it would be saved to /var/dkms/qla2x00/original_module/2.4.20-8smp/.

To complete the DKMS cycle, you also can uninstall or remove your module from the tree. Uninstall removes the module you installed and, if applicable, replaces it with its original module. In scenarios where multiple versions of a module are located within the DKMS tree, when one version is uninstalled, DKMS does not try to understand or assume which of these other versions should be put in its place. Instead, if a true original_module was saved from the original DKMS installation, it is put back into the kernel. All of the other module versions for that module are left in the built state. An example uninstall would be:

```
dkms uninstall -m qla2x00 -v v6.04.00 -k 2.4.20-8smp
```

If the kernel version parameter is unset, the currently running kernel is assumed, but the same behavior does not occur with the remove command. Remove and uninstall are similar in that a remove command completes all of the same steps as does an uninstall. However, if the module-version being removed is the last instance of that module-version for all kernels on your system, after the uninstall portion of the remove completes, remove physically removes all traces of that module from the DKMS tree. In other words, when an uninstall command completes, your modules are left in the “built” state. However, when a remove completes, you have to start over from the add command before you can use this module again with DKMS. Here are two sample remove commands:

```
dkms remove -m qla2x00 -v v6.04.00 -k 2.4.20-8smp
dkms remove -m qla2x00 -v v6.04.00 --all
```

With the first remove command, the module would be uninstalled. If this module/module-version were not installed on any other kernel, all traces of it would be removed from the DKMS tree. If, say, qla2x00/v6.04.00 also was installed on the 2.4.20-8bigmem kernel, the first remove command would leave it alone—it would remain intact in the DKMS tree. That would not be the case in the second example. It would uninstall all versions of the qla2x00/v6.04.00 module from all kernels and then completely expunge all references of qla2x00/v6.04.00 from the DKMS tree. Thus, remove is what cleans your DKMS tree.

Miscellaneous DKMS Commands

DKMS also comes with a fully functional status command that returns information about what is currently located in your tree. If no parameters are set, it returns all information found. Logically, the specificity of information returned depends on which parameters are passed to your status command. Each status entry returned is of the state added, built or installed. If an original module has been saved, this information also is displayed. Some example status commands include:

```
dkms status
dkms status -m qla2x00
dkms status -m qla2x00 -v v6.04.00
dkms status -k 2.4.20-8smp
dkms status -m qla2x00 -v v6.04.00 -k 2.4.20-8smp
```

Another major feature of DKMS is the match command. The match command takes the configuration of DKMS-installed modules for one kernel and applies it to some other kernel. When the match completes, the same module/module-versions installed for one kernel are then installed on the other kernel. This is helpful when you are upgrading from one kernel to the next but want to keep the same DKMS modules in place for the new kernel. In the example:

```
dkms match --templatekernel 2.4.20-8smp
↪-k 2.4.20-9smp
```

--templatekernel is the match-er kernel from which the configuration is based. The -k kernel is the match-ee upon which the configuration is instated.

For systems management purposes, the commands mktarball and ldtarball also have been added to DKMS. These commands allow the user to make and load tarball archives, respectively, into the DKMS tree to facilitate using DKMS in deployments where many similar systems exist. This allows the system administrator to build modules on only one system. Rather than build the same module on every other system, the built binary can be applied directly to the other systems' DKMS tree. Specifically, mktarball creates a tarball of the source for a given module/module-version. It then archives the DKMS tree of every kernel version that has a module built for that module/module-version. Consider the example:

```
dkms mktarball -m qla2x00 -v v6.04.00
↪-k 2.4.20-8smp,2.4.20-8
```

Depending on the -k kernel parameter, mktarball archives only certain binaries compiled for those kernels specified. If no kernel parameter is given, it archives all built module binaries for that module/module-version.

With ldtarball, DKMS simply parses the archive created with mktarball and applies whatever is found to that system's DKMS tree. This leaves all modules in the built state; the dkms install command then can be used to place the module binaries into the /lib/modules tree. Under normal operation, ldtarball does not overwrite any files that already exist in the system's DKMS tree. However, the archive can be forced over what is in the tree with the --force option. An example ldtarball:

```
dkms ldtarball --config
↪qla2x00-v6.04.00-kernel2.4.20-8smp.tar.gz
```

The last miscellaneous DKMS command is mkdriverdisk. As can be inferred from its name, mkdriverdisk takes the proper sources in your DKMS tree and creates a driver disk image that can provide updated drivers to Linux distribution installations. A sample mkdriverdisk might look like:

```
dkms mkdriverdisk -d redhat -m qla2x00
↪-v v6.04.00 -k 2.4.20-8BOOT
```

Currently, the only supported distribution driver disk format is Red Hat, but this easily could expand with some help from the community in understanding driver disk requirements and formats on a per-distribution basis. For more information on the extra necessary files and their formats for DKMS to create Red Hat driver disks, see people.redhat.com/dledford. These files should be placed in your module source directory.

The dkms.conf Configuration File Format

For maintainers of DKMS packages, the dkms.conf configuration file is the only auxiliary piece necessary to make your source tarball DKMS-ready. The format of the conf file is a successive list of shell variables sourced by DKMS when working with your package. For example, an excerpt from the qla2x00/v6.04.00 dkms.conf file:

```
MAKE="make all
↪INCLUDEDIR=/lib/modules/$kernelver/build/include"
```

```
MAKE_smp="make SMP=1 all
↪INCLUDEDIR=/lib/modules/$kernelver/build/include"
LOCATION="/kernel/drivers/addon/qla2200"
REMAKE_INITRD="yes"
MODULE_NAME="qla2200.o:qla2200_6x.o
↪qla2300.o:qla2300_6x.o"
CLEAN="make clean"
MODULES_CONF_ALIAS_TYPE="scsi_hostadapter"
MODULES_CONF0="options scsi_mod
↪scsi_allow_ghost_devices=1"
```

shows that each of the shell variable directives should be coded in all capital letters. One of the current exceptions to this rule is the MAKE_ directive. DKMS uses the generic MAKE= command to build your module. But, if a MAKE_*kernel-regexp-text* command exists and the text after the MAKE_ matches (as a substring) the kernel for which it is being built, then this alternate make command is used. In the above example, you can see how DKMS would use the MAKE_smp directive on any smp kernel for which it was building this module. Similar PATCH_ commands also exist. When the text after the underscore matches the kernel for which a module is being built, that patch first is applied to the module source. This allows developers to distribute one source tarball, with one dkms.conf and multiple patches. Yet, different patches can be applied as necessary to the source to ensure all modules function correctly on all kernels.

Also notice that dkms.conf accepts the $kernelver variable, which, at build time, is replaced with the kernel version for which the module is being built. This is especially important so the correct include directories are referenced when compiling a module for a kernel that is not currently running.

Using DKMS in Conjunction with RPM

DKMS and RPM actually work quite well together. The only twist is that to make it function properly, you have to create an RPM that installs source. Although normal practice is to install source only with source RPMs, a source RPM does not necessarily work with DKMS; it will not let you do much besides install the source. Instead, your source tarball needs to be included with your RPM, so your source can be placed in /usr/src/*module-module-version*/ and the proper DMKS commands can be called. The %post and %preun basically are DKMS commands. Here is a sample .spec file:

```
%define module qla2x00

Summary: Qlogic HBA module
Name: %module_dkms
Version: v6.04.00
Release: 1
Vendor: Qlogic Corporation
Copyright: GPL
Packager: Gary Lerhaupt <gary_lerhaupt@dell.com>
Group: System Environment/Base
BuildArch: noarch
```

```
Requires: dkms gcc bash sed
Source0: qla2x00src-%version.tgz
Source1: dkms.conf
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root/

%description
This package contains Qlogic's qla2x00 HBA module meant
for the DKMS framework.

%prep
rm -rf qla2x00src-%version
mkdir qla2x00src-%version
cd qla2x00src-%version
tar xvzf $RPM_SOURCE_DIR/qla2x00src-%version.tgz

%install
if [ "$RPM_BUILD_ROOT" != "/" ]; then
        rm -rf $RPM_BUILD_ROOT
fi
mkdir -p $RPM_BUILD_ROOT/usr/src/%module-%version/
install -m 644 $RPM_SOURCE_DIR/dkms.conf
$RPM_BUILD_ROOT/usr/src/%module-%version
install -m 644 qla2x00src-%version/*
$RPM_BUILD_ROOT/usr/src/%module-%version

%clean
if [ "$RPM_BUILD_ROOT" != "/" ]; then
        rm -rf $RPM_BUILD_ROOT
fi

%files
%defattr(0644,root,root)
%attr(0755,root,root) /usr/src/%module-%version/

%pre

%post
/sbin/dkms add -m %module -v %version
/sbin/dkms build -m %module -v %version
/sbin/dkms install -m %module -v %version
exit 0

%preun
/sbin/dkms remove -m %module -v %version --all
exit 0
```

Next Steps

Because DKMS is a recently conceived framework, many things can be added, removed or recoded as the community decides. The latest project information for DKMS can be found at www.freshmeat.net/projects/dkms. You can ask questions and provide feedback by joining the DKMS-devel mailing list at lists.us.dell.com/mailman/listinfo.

In the December 2002 issue of *Linux Journal*, Linus Torvalds was quoted as saying, “Basically, all of the commercial people have their own agenda, and that's very healthy because you want to have these often-conflicting agendas to push the system into something that actually works for everybody.” As a commercial reseller of Linux-enabled products, Dell is interested in finding a good solution to the ongoing module/kernel issue, both to support the community and to create a better Linux experience for their customers. DKMS was designed with this in mind.

Gary Lerhaupt (gary_lerhaupt@dell.com) is a software engineer on Dell's Linux Development team. He also has collaborated on the Dell Oracle9i Real Application Clusters (RAC) initiative deployed on Red Hat Linux. Gary has a bachelor's degree in Computer Science and Engineering from The Ohio State University.

Archive Index Issue Table of Contents

Advanced search

At the Forge

Bricolage

Reuven M. Lerner

Issue #113, September 2003

The content management behind *Salon* and other popular sites is friendly to use for your Web site's writers and editors.

Over the past few months, we have looked at a number of different content management systems (CMSes) based on Zope. Of course, Zope is not the only game in town when it comes to open-source content management systems. One increasingly more prominent package is Bricolage, written and maintained largely by David Wheeler and based on mod_perl and PostgreSQL.

Bricolage is designed to be used by nontechnical people. True, it takes a fair amount of experience to modify and maintain Bricolage. But, whereas the people who use Apache or Perl generally are programmers or system administrators, the people who use Bricolage the most are the writers, editors and producers of a Web site.

Bricolage also has managed to acquire a fair amount of real-world experience. Apache and Perl needed to prove themselves for many years before they were accepted as part of the mainstream; Bricolage has been part of *Salon* magazine's CMS for a while, and sites such as *eWeek* and the *Register* are in the process of moving over. Moreover, professional publications not often known for their positive views of open-source software recently have begun to evaluate and review Bricolage. Most have found it to be an excellent package, one that rivals proprietary software costing hundreds of thousands of dollars.

This month, we take an initial look at how to install and begin using Bricolage. Over the next few months, we will look at it from a number of perspectives, examining ways in which we can customize and use Bricolage for different types of sites and publications.

Foundations

The core of every CMS is a database. Commercial CMS software typically uses Oracle or Microsoft's SQL Server as the back-end database server. Many open-source projects, including many of the systems based on PHP, use MySQL. Bricolage, by contrast, uses PostgreSQL for its back-end storage.

PostgreSQL often is known as “the other” open-source database, and it has long supported the notion of a transaction, allowing you to group several database queries or commands into a single all-or-nothing group. PostgreSQL also supports other functionality that serious database operators expect, including views, user-defined functions, subselects, unions, foreign keys and integrity checks. PostgreSQL also supports Unicode, which is increasingly important for handling multilingual sites.

Bricolage uses PostgreSQL for its back-end storage, but the application itself is written in Perl. There are at least two ways to run server-side Perl programs for the Web: CGI, which is slow, safe and portable, and mod_perl, which is fast, potentially unsafe, and works only with Apache. Bricolage works under mod_perl, meaning that its code—which is written as a set of Perl modules—is compiled once, cached in memory as Perl “opcodes” and executed multiple times. As a result, Bricolage executes quickly.

As I mentioned above, mod_perl works only under Apache. Although constant development work is being done on mod_perl for Apache 2.x, you should expect to run mod_perl for only Apache 1.x as of this writing, in early June 2003. Because Apache 1.x runs as a set of processes rather than multiple threads within a single process, no way exists to do true database pooling among the various child HTTP servers. However, keeping an established database connection alive between Apache and PostgreSQL is possible using the Apache::DBI module. Bricolage does exactly this, ensuring that database connections do not need to be re-established each time a user makes a request.

Finally, Bricolage presents its data to the end user with a set of Perl/HTML templates. Many such templates are available for Perl in general and for mod_perl in particular. I have been a longtime fan of HTML::Mason.

As you can tell, one of the reasons I am so enthusiastic about Bricolage is it combines some of my favorite technologies—PostgreSQL, mod_perl, Apache and HTML::Mason—into a single application that is good for end users.

Installation

Installing Bricolage is not a simple process. This situation is not the fault of the Bricolage authors or maintainers but, rather, a result of Bricolage using so many different modules from the Comprehensive Perl Archive Network (CPAN). Currently, installation still is not as smooth or easy as it could be, but things have improved over time, with easier installation accompanying each version.

The easiest way to install Bricolage, after you have already installed Perl, Apache and mod_perl—as a static module, not as an Apache dynamic shared object (DSO)—is to use the pseudo-module Bundle::Bricolage defined in CPAN. Normally, you can install a Perl module with the interactive CPAN tool by first starting the CPAN shell with `perl -MCPAN -e 'shell'` and then typing `install Bundle::Bricolage` at the prompt. If you are running a relatively recent version of Perl, and if you have defined the environment variables PGINCLUDE and PGLIB, all of the modules should download, compile and install perfectly.

This is a long and involved process, however, and something is bound to go wrong, if you're like me, with CPAN and double-checking that you have installed everything you need by trying one last time to install Bundle::Bricolage. For example, I installed LWP and Bundle::CPAN using the interactive CPAN shell. I then tried to install Bundle::Bricolage; the installation (on a virtual colocation system running Red Hat Linux 7.3) failed for Cache::Cache the first time around but succeeded the second time. CPAN dependencies sometimes can be tricky, and not all modules clearly define and indicate theirs. It also failed on DB_File (because the RPM for db3-devel was not installed), causing problems with the installation of Apache::Session, which in turn caused problems with HTML::Mason, on which Bricolage depends. And, there were problems installing libapreq (because Apache was already running on the same port number) and XML::Parser (because the expat-devel RPM wasn't installed). Luckily, trying to install a CPAN bundle indicates (at the end) which packages didn't install cleanly. You always can try to re-install the bundle, in which case the CPAN shell tells you which modules already are installed and which still need to be installed.

Bundle::Bricolage does not install the Bricolage modules but the modules on which Bricolage depends. So after you have double-checked that Bundle::Bricolage is working correctly, download the latest Bricolage tarball from the Bricolage home page (www.bricolage.cc), open it up and type `make`. The Makefile checks to make sure all required and some optional Perl modules have been installed, and then asks if you want to install any that are missing. It also checks that mod_perl was compiled statically (and not as a DSO) and that PostgreSQL is installed. Finally, it asks for a number of user names and passwords that Bricolage needs in order to set up its databases and install its HTML::Mason components on the system.

Once you answered all of the questions, you can start the installation with:

```
make cpan && make test && make install
```

This command downloads and installs necessary modules from CPAN, tests the Bricolage installation to ensure that it is working right and, if all went well, installs Bricolage itself.

It's easy to run into some problems when installing Bricolage, but overall the installation was impressively easy, given the complexity of the system. The Makefile offered intelligent defaults, forced me to provide a password to the “bric” PostgreSQL user and generally ensured that things were running correctly and smoothly.

Once everything was installed, I invoked `bric_apachectl start` to start Apache, pointed my Web browser at the root directory and was greeted with a screen asking me to log in. The installation had succeeded. As suggested by the Bricolage documentation, I immediately changed the administrative password, which is appropriately set to “change me now!” by default.

At this point I should mention that Bricolage uses a fair number of pop-up windows. I personally happen to dislike pop-ups, even when they are not used in the context of advertising. Although I understand the nature of user interfaces in HTML often requires pop-ups for the natural flow of information, I still wish Bricolage used them a bit less. Moreover, a number of the internal links depend on JavaScript, which means I cannot use my beloved middle mouse button to open a new tab instead of a new window. But, this and other minor blemishes do little to change the fact that Bricolage is truly an amazing package.

Behind the Scenes

Bricolage consists of several parts: a data model in PostgreSQL, a number of Perl modules and a number of front-end templates that display information retrieved by the modules. In this way, it is similar to other sophisticated database-backed systems. Indeed, although Bricolage attempts to solve a more limited set of problems than OpenACS does, and obviously uses different technologies to accomplish its goals, the separation of data, libraries and templates is quite similar to that system and to many other systems using a three-tier architecture.

If you look in the inst directory of the Bricolage package, you see the PostgreSQL database definitions used by the system. If you are new to PostgreSQL, you might be a bit surprised by some of the things you see there, such as sequences and constraints.

Sequences are a special kind of numeric object in PostgreSQL whose values can never be reused. They are most often used to ensure that IDs in the system are unique, especially when used as primary keys. If you define a PostgreSQL column to be SERIAL rather than INTEGER, you actually are creating a sequence behind the scenes. Older versions of PostgreSQL made it inconvenient to drop a table with a SERIAL column; you had to drop the table and then drop the sequence associated with the SERIAL column as well. As of PostgreSQL 7.3, however, removing a table automatically removes any sequences that were associated with its SERIAL columns.

Constraints allow the database to reject INSERT and UPDATE statements that set values outside of a particular range. For example, Bricolage defines a media table in which every element has a priority between 1 and 5 but a default of 3. The column definition thus looks like:

```
  priority   NUMERIC(1,0)  NOT NULL
                           DEFAULT 3
                           CONSTRAINT ck_media__priority
                           CHECK (priority BETWEEN 1 AND 5)
```

The constraint can be given an optional name, in this case ck_media_priority. This makes it easier to find and fix errors; if you try to insert an invalid value into this column, PostgreSQL indicates the name of the violated constraint. This helps quite a bit when debugging problems with the database definition as well as with the applications using the database.

Also a bit surprising is that a small number of functions are defined. PostgreSQL makes it possible to define functions in a number of different languages, including standard SQL and the procedural Pl/PGSQL, as well as database-enhanced versions of Perl, Python and Tcl.

Of course, the core of the Bricolage data model is the tables. A person table is used to describe system users, an org table describes organizations and then a person_org table handles the intersection between these two tables.

Indeed, it's possible to achieve a good understanding of what's happening inside Bricolage without too much difficulty by simply looking at the database definitions. For example, I added a new story—a glowing review of *Core Perl*—to Bricolage using the default administrative user account with all of the defaults. This inserted a new record into the story table, assigning it a priority (as we saw above), a date of publication and expiration, a version number (as

Bricolage also handles versioning of articles) and an indication of whether the story has been published.

Several keys foreign to other tables demonstrate how this particular article fits into the system. We can see it was created by the administrative user, because the usr column points to the usr table; it is part of the story workflow, distinct from other workflows that have been defined, and referenced in a foreign key from the workflow__id column; it came from the edit desk, distinct from other desks, such as copy, legal and publishing, and referenced with a foreign key to the desk table; and it is considered to be a book review element, because its element__id column points to the element table. Each of these other tables is connected to still other tables that provide additional auxiliary information, from burners to group IDs to formatting information.

In short, the story table sits close to the center of the Bricolage data model—just as it should be given that a CMS is centered around content, of which stories are the primary example. Indeed, if you are new to the world of relational databases or want to see an open-source project that uses them in a sophisticated way, you would do well to look at Bricolage.

Conclusion

This month, we took our first look at Bricolage, an open-source content management system based on mod_perl and PostgreSQL. We learned how to install and begin using it and then dove a bit into the data model that Bricolage uses to keep track of the various story elements.

Next month, we'll look at how to define elements, categories, media types and burners, which will let us not only poke around with the system but actually publish documents to our own private Web site. Following that, we will dive a bit more deeply into the system, examining the Mason templates that allow us to move away from the default Bricolage look and feel, toward something closer in spirit to the design we want for our own personal Web sites.

Resources

The main source of information about Bricolage is the project's Web site, located at www.bricolage.cc). This site has pointers to downloadable source code (hosted at SourceForge), documentation and an instance of Bugzilla (bugzilla.bricolage.cc for bug reports and feature requests.

Several Bricolage mailing lists are hosted by SourceForge, and the developers participate actively. If you have questions, or want to learn about new releases, you can subscribe at the SourceForge page (sourceforge.net/projects/bricolage).

The Bricolage documentation is generally quite good, if technical. A more user-level introduction to the system was published by O'Reilly and Associates as an appendix to their recently published book, *Embedding Perl in HTML with Mason* by Dave Rolsky and Ken Williams. You can read that appendix on-line at www.masonbook.com/book/appendix-d.mhtml.

For more information about PostgreSQL, see the project's main site at www.postgresql.org. For more information about Apache, see httpd.apache.org. To learn more about mod_perl, see perl.apache.org. Remember that Apache 2.x and mod_perl 2.x are both unsuitable for Bricolage, although that may change by the time you read this. Finally, you can learn more about Mason both from the Mason book site (www.masonbook.com) and from the Mason home page (www.masonhq.com).

Finally, you can learn more about David Wheeler, the primary author and maintainer of Bricolage, at david.wheeler.net, and about his company Kineticode at www.kineticode.com.

Reuven M. Lerner (reuven@lerner.co.il) is a consultant specializing in open-source Web/database technologies. He and his wife, Shira, recently celebrated the birth of their second daughter, Shikma Bruria. Reuven's book *Core Perl* was published by Prentice Hall in early 2002, and a second book about open-source Web technologies will be published by Apress in 2003.

Archive Index Issue Table of Contents

Advanced search

Cooking with Linux

Watching the Community Network

Marcel Gagné

Issue #113, September 2003

Use your Linux system to watch TV, record home movies and listen to FM radio.

Yes, François, that is me—20 years ago, but still me. What are you smiling about? I don't look quite that strange. Yes, I've been told that I look like I am doing a Carl Sagan impression, but that was simply the way I spoke, *mon ami*. Stop smirking. Our guests will be here any moment and with this issue's feature on community networks, I am going to need you to take good care of them.

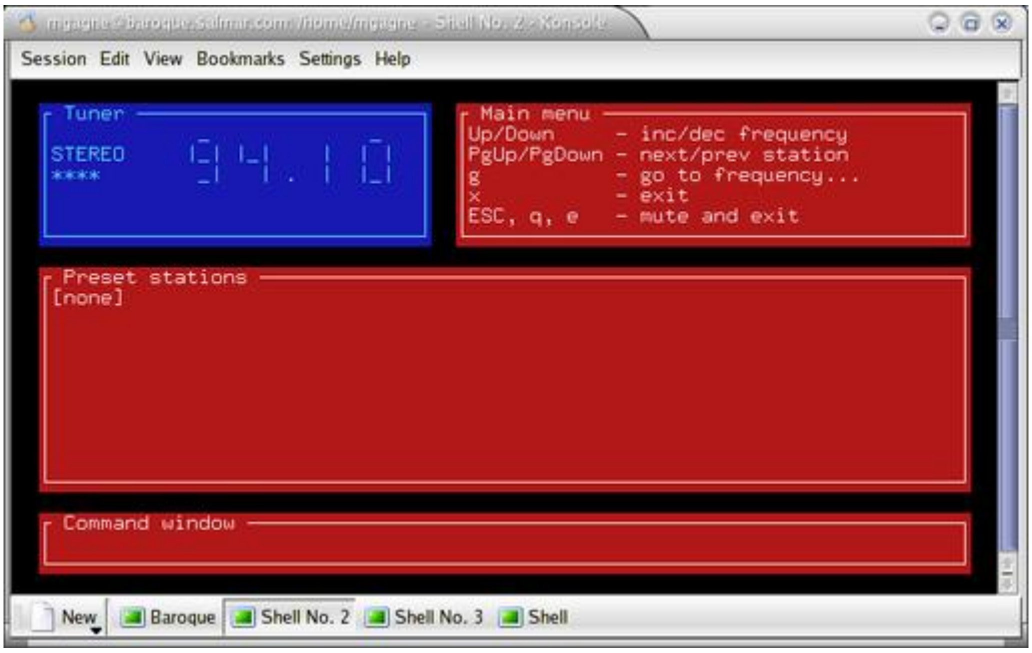
Ah, but they are already here. Welcome, *mes amis*! François—to the wine cellar. The 2000 Cabernet Sauvignon Stellenbosch from South Africa is drinking quite nicely right now. Please fetch it, *tout de suite*. Please, everyone, sit and make yourselves comfortable.

I was showing François a tape from a documentary I did a number of years ago with some friends. I used to volunteer at the community cable station where I operated the cameras and worked the video board from time to time. It was a great deal of fun. One summer, with the equipment from the station, my friends and I produced a documentary that I wrote and narrated. For years, I've had the tape floating around, and now, using this issue's feature as a springboard, I thought it might be nice to convert it to a more permanent, digital format.

Playing video from an analog source can be done with the use of a TV tuner card. The card I used is a Hauppauge WinTV card, based on the btv878 chipset. You can ignore the card's name safely, *mes amis*. It works quite well with any recent Linux kernel. The card I bought is well supported, and a driver for it was loaded automatically on my Mandrake 9.1 test system (it also worked beautifully on another Debian system). The Linux bttv kernel module, or driver, supports a large number of TV tuner cards. A quick look at the CARDLIST file in the kernel documentation (/usr/src/linux/Documentation/video4linux/bttv/) will give you an idea.

Welcome back, François. Please pour for our guests...and quit smirking, it isn't that funny. You see, *mes amis*, my faithful waiter is amused by the sight of myself all those years ago. Ignore him and enjoy the wine while I show you how this works.

Watching television is obviously the point, but these TV tuner cards sometimes have FM radio receivers as well. Watching television while you are trying to work might qualify as distracting. Listening to the radio isn't as problematic. An FM radio program that caught my eye is Gerd Knorr's frightfully simple **radio** program (bytesex.org/xawtv). This is an ncurses-based program that comes as part of the xawtv source package. I make a point of specifying source here, because the package, called radio, is separate from xawtv when you work with RPMs.


mgagne@baroque.salmar.com: /home/mgagne - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
New
Baroque
Shell No. 2
Shell No. 3
Shell

```
Tuner                          Main menu
STEREO      94.10              Up/Down       - inc/dec frequency
****                           PgUp/PgDown   - next/prev station
                               g             - go to frequency...
                               x             - exit
                               ESC, q, e     - mute and exit

Preset stations
[none]

Command window
```

Figure 1. radio: Simple and ncurses-Based

From the command line, type `radio -s`. With the -s option, the program displays the frequencies it visits while looking for stations, then starts unmuted. If you use the -i option instead, the program writes a .radio file in your home directory with the stations and frequencies it identified. Then, you can go back and edit this text file to give more interesting names to the stations. You also can start the radio on a favorite station by passing the -f option (`radio -f 99.10`).

There you are, a local community network broadcast on that most venerable of medium, radio. I should point out that I did run into some interesting problems here. My Hauppauge card did not play sound directly. It was possible to plug my speakers right into the card's audio out port, but I wanted the control in the system. To do this, I had to use the accompanying cable to connect the tuner card to my system's sound card. I then used alsamixer to raise the levels of my line-in and all was well. Depending on your TV tuner card, you may have to do the same. This will be particularly important when we start talking about recording.

Another radio project you should have a look at is **GQradio**, created by John Ellis (gqmpeg.sourceforge.net/radio.html). GQradio features a slick graphical interface, autotune, station presets and more. The site offers Red Hat RPMs as well as source. Compiling the program requires the GTK+ and gtk-pixbuf libraries. Beyond that, it's the classic extract and build five-step:

```
tar -xzvf gqradio-0.99.0.tar.gz
cd gqradio-0.99.0
./configure
make
su -c "make install"
```

If you run the gqradio program from the command line, you'll notice something interesting the first time:

```
$ gqradio
Creating dir:/home/mgagne/.gqradio
Creating dir:/home/mgagne/.gqradio/skins
```

The program creates an initial directory for its configuration as well as for skins. That's right, the program is skinnable. Not only can you download skins, but GQradio comes with a built-in skin editor so you can unleash your creative spirit. Right-click on the GQradio GUI for a drop-down menu that lets you modify the program at will.

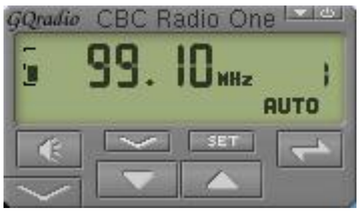


Figure 2. GQradio with the Presets Display Open

Next, we turn our attention to the world of video broadcasts and my own early community network brush with television reporting. François, perhaps you had better pour me a glass as well. After all your laughter, it occurs to me that I may need a little courage.

For the impatient among you, most Linux distributions come with a television viewing application called **xawtv** by Gerd Knorr (of ncurses **radio** fame). If you simply start up the xawtv program, you'll likely find that you get nothing but a blank screen, particularly if you are in North America. The program defaults to the PAL format among other things. To lock xawtv into some saner settings for your particular location, you need to edit the $HOME/.xawtv configuration file. Here's what mine looks like:

```
[global]
freqtab = us-bcast

[defaults]
input = Television
norm = NTSC

[vcr]
channel = 3
key = 3
```

The file is broken up into sections, denoted by a title inside square brackets (for all the possible settings, do a `man xawtvrc`). The global and defaults section are the most important ones, because they allow us to set our local transmission standards as well as the input device. The [vcr] section is one I added. Quite simply, it represents channel 3, the output from my VCR.

Figure 3. Future Linux Chef?

Many programs are available for watching TV. MPlayer (www.MPlayerHQ.hu) is a popular, do-it-all kind of player. Once again, check your distribution—you may already have a copy. Using codecs, it can play all those AVI or MPG files you have floating around. You may know this already, but MPlayer also can handle a TV tuner card. Here's how:

```
mplayer -tv \
on:driver=v4l:channel=3:input=0:norm=NTSC\
:width=640:height=480
```

The backslashes in the above command are there because I couldn't fit the entire command on one line. You can type it as one, unbroken line, if you prefer. To use the MPlayer GUI, use `gmplayer` instead. The driver parameter selects the video4linux driver (v4l). The channel setting is obvious, and the norm setting lets me choose the NTSC broadcast standard for North America. The final settings are for height and width.

These programs are great ways to watch TV on your Linux system, but if you are like me, you've looked with some amount of envy at the entertainment systems your friends with digital cable have a nasty habit of showing off. With their TiVos and their PVR (personal video recorder) units, they have access to on-screen displays of upcoming shows, which they can schedule and watch whenever the mood takes them. Well, envy no more, *mes amis*. To have an entire video entertainment system under the control of your Linux system, look no further than MythTV, a full-featured personal video recorder and television master control.

Imagine if you will, a digital video recorder that lets you view live TV, with instant replay so you can pause, rewind and fast-forward through the action. MythTV lets you program shows on a timer and view on-line television listings on which you can do string searches—looking for a particular episode of a *Buffy* rerun? Combine this with support for multiple TV tuner cards, multiple simultaneous recordings and a distributed system so you can set up different MythTV boxes on your network. Wrap it all up in a slick, themeable package and you are starting to get an idea of what I am talking about.

Does this sound like something you can't wait to get your hands on? Then, head on over to www.mythtv.org and pick up the packages. RPM packages are available for different distributions, as are Debian packages and Gentoo ebuild and digest files. Check out the “Software” section of www.mythtv.org/docs/mythtv-HOWTO-3.html for links to prebuilt packages. This fine package is also available in a source archive bundle.

MythTV is a very cool package, but making it run requires some work and that you install a few prerequisites. Most of these are development libraries comprising of freetype2-devel, XFree86-devel, qt-devel, lame and libexpat. MythTV recommends that you get the latest libexpat from sourceforge.net/projects/expat. Finally, to use MythTV's XMLTV channel information grabber and scheduling tool, you definitely need Perl and a handful of modules. It also is possible to use an infrared remote with MythTV (using lirc), but that is one nuance I did not sample.

You may find, as I did, that you need a few additional modules to get XMLTV actually working. These modules are XML::Twig, Date::Manip, LWP and XML::Writer. The easiest way to install them is with the `perl -MCPAN -e shell` command. Start the CPAN shell as root, and you'll see a cpan> prompt.

If this is the first time you use CPAN, you'll encounter a question-and-answer session to help the software identify your local CPAN mirrors. Then, enter the following at the cpan> prompt:

```
cpan> install XML::Twig
cpan> install Date::Manip
cpan> install LWP
cpan> install XML::Writer
cpan> exit
```

This is all you need. There are also a handful of recommended modules, which you will be told about when you install the xmltv package. The installation is typical for Perl packages and not unlike the standard extract and build five-step:

```
xmltv-0.5.10.tar.bz2
cd xmltv-0.5.10
perl Makefile.PL
make
su -c "make install"
```

As part of the XMLTV installation, you'll be asked about listings for your area, so pay attention to the messages at this stage. You're almost there. Because MythTV uses MySQL to store its information, you need to set up a database for it. You also should have the qt-MySQL package loaded (look for qt-mysql or libqt3-mysql). Even if you have MySQL set up properly, you may get an error similar to `QMYSQL3 driver not loaded` if you run without this. Now, make sure MySQL is installed and running, then use the provided schema file:

```
cd database
mysql -u root < mc.sql
```

Depending on your distribution, you may not need the `-u root` above. Now, we still have a couple of database things to attend to here. At this point, only the localhost address is allowed to access the MythTV databases. In my case, I wanted to provide access to all users in my 192.168.22.0 private subnet:

```
$ mysql -u root mythconverg
mysql> grant all on mythconverg.* to \
mythtv@"192.168.1.%" identified by "mythtv";
```

The percent sign above is used as a wild card. Consequently, if you want to allow all domains in (probably not what you want), remove the subnet portion and leave only the percent sign.

Aside from taking several steps, the installation was pretty easy. That said, I did run into a couple of minor problems. For instance, with qmake, part of the Qt development package, the configure script didn't seem to be able to locate the program, so I created a symbolic link for it in /usr/bin. Then, I ran a `make` from the MythTV distribution directory, and all was well.

Now that you've compiled and installed the software, in the MythTV distribution directory, you'll see a directory called setup, and in that directory, an executable by the same name. Run the command as follows: `setup/setup`.

You'll see a screen with four options labeled General, Capture Cards, Video Sources and, finally, Input Connections. Go through each of these to configure MythTV for your local system setup. Your input connections (number 4) should basically be set up already as part of selecting your video sources. Pay attention to the on-screen help messages and guides as you go through this process. When you are done with all the setup steps, press Esc.

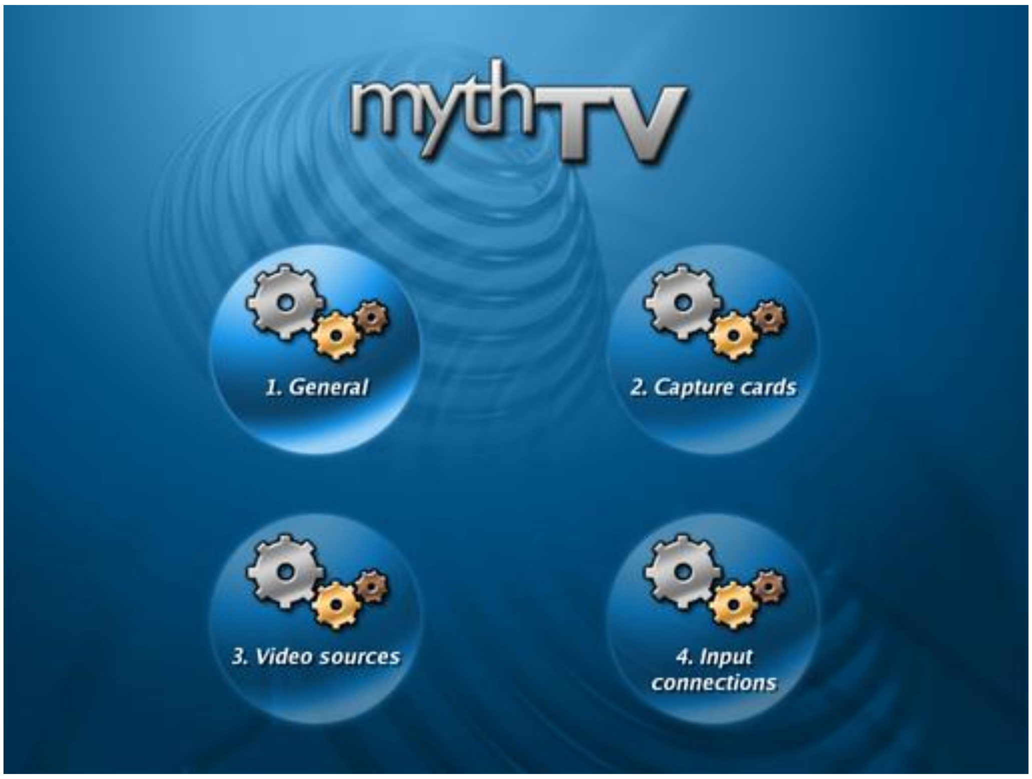


Figure 4. Setting Up the MythTV Front End

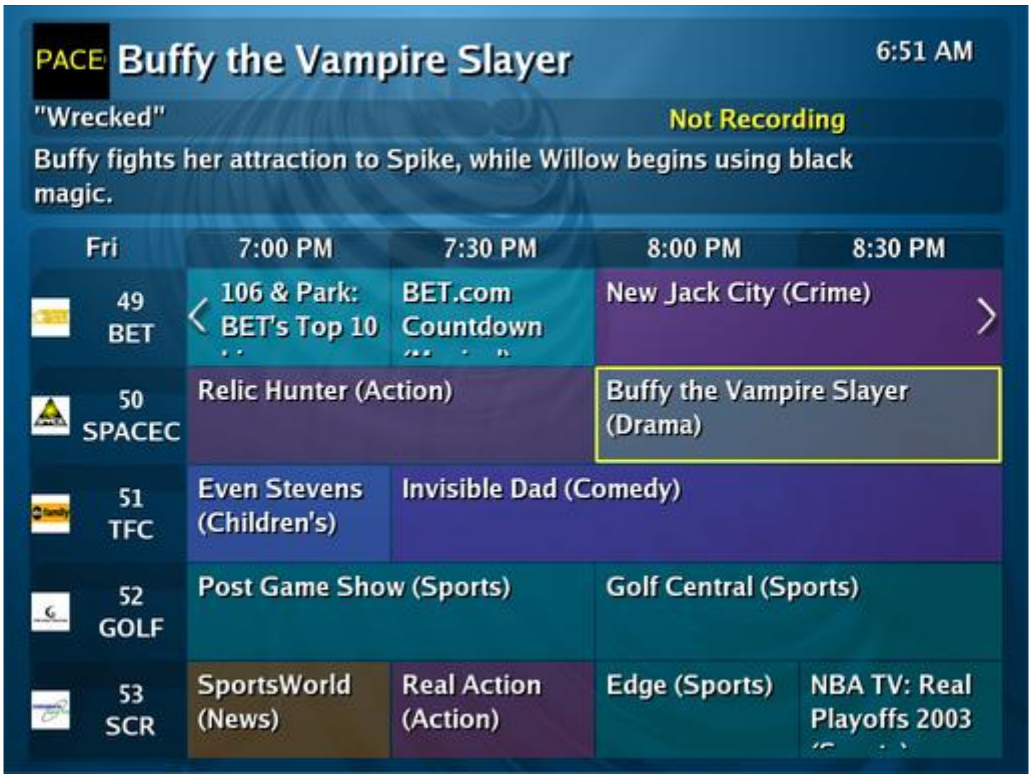
Now, run `mythfilldatabase` to pick up your television listings. If this is the first time, you may encounter an error because your local xmltv configuration file may not exist:

```
/usr/bin/tv_grab_na --configure
```

You'll be asked for your ZIP or postal code to help identify a local provider for the TV guide listings. From that list, you also will be able to decide for which channels xmltv will gather listings. Being one for spice in my life, I opted for maximum variety and accepted every channel. Filling the database with program information is certainly something you will want to do with a cron job.

The next step is to start the mythbackend program. I chose to run it as a dæmon by using the -d option (you can add this to your rc.local start-up scripts if you like).

Finally, as a regular user, run `mythfrontend` to start up the MythTV interface. At this stage, you have two options on the graphical screen, setup and TV. Use your cursor keys to navigate the screen. Choose TV, and you'll be able to select from watching TV now, scheduling a recording or watching a previously scheduled recording. You also can browse the Program Guide for current or upcoming programs (Figure 4).



PACE Buffy the Vampire Slayer 6:51 AM

"Wrecked" Not Recording

Buffy fights her attraction to Spike, while Willow begins using black magic.

Fri	7:00 PM	7:30 PM	8:00 PM	8:30 PM
49 BET	106 & Park: BET's Top 10	BET.com Countdown	New Jack City (Crime)	
50 SPACEC	Relic Hunter (Action)		Buffy the Vampire Slayer (Drama)	
51 TFC	Even Stevens (Children's)	Invisible Dad (Comedy)		
52 GOLF	Post Game Show (Sports)		Golf Central (Sports)	
53 SCR	SportsWorld (News)	Real Action (Action)	Edge (Sports)	NBA TV: Real Playoffs 2003

Figure 5. Searching through the MythTV Program Guide

Another option is to record embarrassing programs from your youth, such as my early documentary of the 1984 Tall Ships Festival, “Romancing the Sail”. Select the appropriate channel for your VCR, pop in the tape and record those memories.

MythTV is an excellent package with a great deal of energy and promise. In addition to the obvious television playing and recording aspect of the package, additional modules are available, which I don't have time to cover here. They include MythWeb, which allows control of MythTV through a Web page; MythGallery, for photos and slideshows; MythMusic, for ripping, storing and playing music files; and MythWeather, for weather information.

Well, *mes amis*, the hour is growing late and we must soon close the doors. As late as it is, after tonight, you will have no worries about missing your favorite programs, *non*? In the meantime, François happily will pour you another glass of wine—perhaps we should make it two glasses, and you can pretend you never saw that old Marcel video. There is surely something to be said for growing up. Until next time, *mes amis*, let us all drink to one another's health. A vôtre santé! Bon appétit!

Resources

bttv, radio and xawtv: bytesex.org/xawtv

GQradio: gqmpeg.sourceforge.net/radio.html

MPlayer: www.MPlayerHQ.hu

MythTV: www.mythtv.org

Marcel's Wine Page: www.marcelgagne.com/wine.html

Marcel Gagné lives in Mississauga, Ontario. He is the author of *Linux System Administration: A User's Guide* (ISBN 0-201-71934-7), published by Addison-Wesley, and is currently at work on his next book. He can be reached via e-mail at mggagne@salmar.com.

Paranoid Penguin

Authenticate with LDAP, Part III

Mick Bauer

Issue #113, September 2003

In the conclusion of his series on LDAP, Mick takes the secure LDAP server project to the point where it can authenticate real users for real applications.

For the past couple of months in the Paranoid Penguin column, we've been building an LDAP server. We've installed OpenLDAP; configured slapd, the server dæmon; made TLS encryption work; and created our first LDAP record, a root organization entry. Now, it's time to add some users and start using our server for authenticating IMAP sessions.

Database Structure

The first step in creating an LDAP user database is to decide on a directory structure, including whether to group users and other entities or use a completely flat structure. If your LDAP database is strictly an on-line address book or authentication server, a flat database may suffice. In that case, your users' Distinguished Names (DNs) should look like this: `dn=Mick Bauer,dc=wiremonkeys,dc=org`.

If, however, your database contains information not only about individual users but also records for organizational subgroups or departments, for computers on your network and so on, you'll probably want to use a more sophisticated directory tree structure. There are a variety of ways to do this. One is by using domainComponent (dc) fields to create subdomains of your domain name, regardless of whether these actually exist in DNS. The method looks like `dn=Bick Mauer,dc=engineering,dc=wiremonkeys,dc=org`. Another is to use organizationalUnit objects in the same way, for example, `dn=Dick Lauer,ou=engineering,dc=wiremonkeys,dc=org`.

In order to keep this discussion simple, I use a flat database for the rest of the article; I leave it to you to determine whether and how to structure an LDAP database that best meets your particular LDAP needs. The documentation found at www.openldap.org and included with OpenLDAP software provides ample examples.

Schema and User Records

Another decision you need to make is which LDAP attributes you want to include for each record. Last month, I described how these are grouped and interrelated in schemas. You may recall that the schemas you specify, or include, in /etc/openldap/slapd.conf determine which attributes are available for you to use in records.

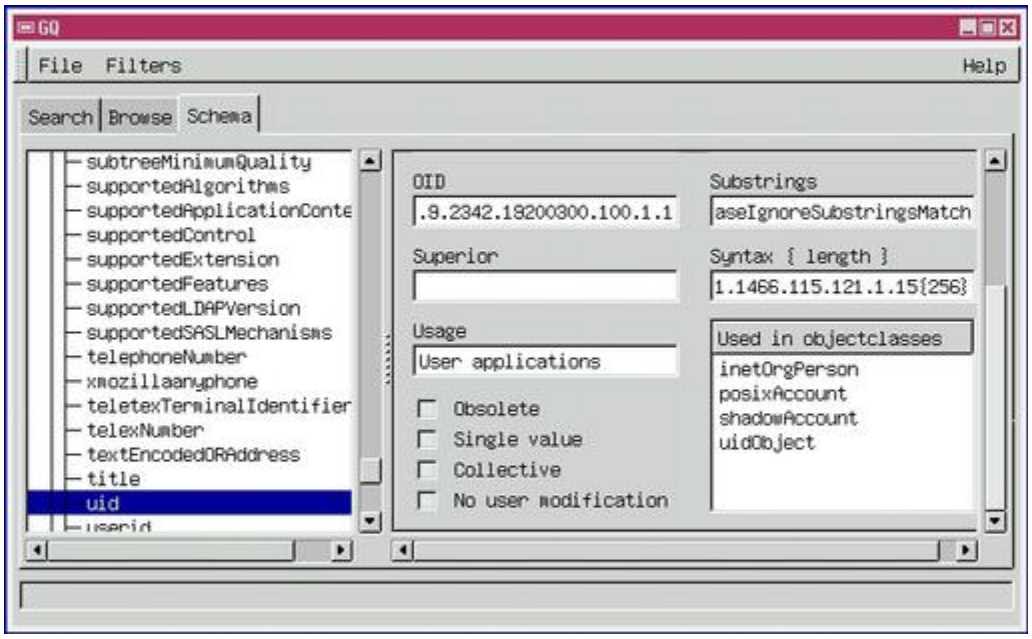
In addition to including schema in /etc/openldap/slapd.conf, in each record you create you need to use objectClass statements to associate the appropriate schemas with each user. Again, as discussed last time, the schema files in /etc/openldap/schema determine which schema support which attributes, and within a given schema, which object classes to which those attributes apply.

Suppose you intend to use your LDAP server to authenticate IMAP connections. The essential LDAP attributes for this purpose are uid and userPassword. This also holds true for any other application that authenticates to LDAP using the Bind method, in which the authenticating service simply attempts to bind to the LDAP server using the user name and password supplied by the user. If the bind succeeds, authentication is judged successful, and the LDAP connection is closed.

One way to determine which schema and object classes provide uid and userPassword is to `grep` the contents of /etc/openldap/schema for the strings uid and userPassword, note which files contain them and then manually parse those files to find the object classes that contain those attributes in MUST() or MAY() statements. If I do this for uid on a Red Hat 7.3 system running OpenLDAP 2.0, I find that the files core.schema, cosine.schema, inetorgperson.schema, nis.schema and openldap.schema contain references to the uid attribute.

Quick scans of these files (using `less`) tell me the following: core.schema's object uidObject requires uid; cosine.schema's only reference to the attribute uid is commented out and can be disregarded; inetorgperson.schema contains an object class, inetOrgPerson, that supports uid as an optional attribute; nis.schema contains two object classes, posixAccount and shadowAccount, both of which require uid; and openldap.schema's object class OpenLDAPperson also requires uid.

Luckily, there's a much faster way to determine the same information. The gq LDAP tool allows you to browse all supported attributes in all supported schema on your LDAP server. Figure 1 is a screenshot illustrating my LDAP server's support for uid, according to gq.



```
subtreeMinimumQuality
supportedAlgorithms
supportedApplicationConte
supportedControl
supportedExtension
supportedFeatures
supportedLDAPVersion
supportedSASLMechanisms
telephoneNumber
xmozillaanyphone
teletexTerminalIdentifier
telexNumber
textEncodedORAddress
title
uid
userid
```

OID: .9.2342.19200300.100.1.1

Substrings: aseIgnoreSubstringsMatch

Superior:

Syntax { length }: 1.1466.115.121.1.15{256}

Usage: User applications

Used in objectclasses:
inetOrgPerson
posixAccount
shadowAccount
uidObject

☐ Obsolete
☐ Single value
☐ Collective
☐ No user modification

Figure 1. Schema Browsing with gq

The Used in objectclasses box in Figure 1 tells us that the selected attribute, uid, appears in the object classes uidObject, posixAccount, shadowAccount and inetOrgPerson, all of which we identified earlier using `grep`. The object class OpenLDAPperson does *not* appear in the gq screen, because the LDAP server in question doesn't have an include statement in its /etc/openldap/slapd.conf file for the file openldap.schema. When in doubt, you should include schema you're not sure you need. After you settle on an LDAP record format, you can always un-include schema that don't contain object classes you need.

All this probably sounds like a lot of trouble and indeed it can be, but it's extremely important to be able to create records that contain the kinds of information pertinent to your LDAP needs. Because LDAP is so flexible, figuring out precisely how to assemble that information in the form of attributes can take some tinkering.

Building and Adding Records

Just as schema browsing can be done either manually or with a GUI, so can adding LDAP records. We used the manual method last month to create our root organization entry, and we'll do so again to add our first user record. This method has two steps: first, create a special text file in LDIF format, then use the ldapadd command to import it into the LDAP database. Consider the LDIF file in Listing 1.

Listing 1. LDIF File for a User Record

```
dn: cn=Wong Fei Hung,dc=wiremonkeys,dc=org
cn: Wong Fei Hung
sn: Wong
givenname: Fei Hung
objectclass: person
objectclass: top
objectclass: inetOrgPerson
mail: wongfh@wiremonkeys.org
telephonenumber: 651-344-1043
o: Wiremonkeys
uid: wongfh
```

Because they determine everything else, we'll begin by examining Listing 1's objectclass statements: this user has been associated with the object classes top (mandatory for all records), person and inetorgperson. I chose person because it supports the attributes userPassword (which is *not* set in Listing 1; we'll set Mr. Wong's password shortly) and telephonenumber, which I don't need now but may in the future. The object class inetOrgPerson supports the uid attribute, plus a whole slew of others that also may come in handy later.

One way around having to know and comply with the MUST and MAY restrictions in schema is to add the statement `schemacheck off` to /etc/openldap/slapd.conf. This allows you to use any attribute defined in any schema file included in slapd.conf without needing to pay any attention to object classes. However, it also adversely affects your LDAP server's interoperability with other LDAP servers and even with other applications (besides flouting LDAP RFCs), so many LDAP experts consider it poor form to disable schema checking in this manner.

It isn't necessary to discuss each and every line in Listing 1; many of the attributes are self-explanatory. In short, know that you don't need to set every attribute you intend to use, but some are mandatory; they are contained in MUST() statements in their respective object class definitions. Each attribute you do define must be specified in the MUST() or MAY() statement of at least one of the object classes defined in the record, and some attributes, such as cn, may be defined multiple times in the same record.

To add the record specified in Listing 1, use the ldapadd command:

```
$ ldapadd -x \
-D "cn=ldapguy,dc=wiremonkeys,dc=org" \
-W -f ./wong.ldif
```

This is similar to how we used ldapadd in last month's column. For a complete explanation of this command's syntax, see the ldapadd(1) man page.

If you specified the attributes required by all object classes set in the LDIF file, if all attributes you specified are supported by those object classes and if you provide the correct LDAP bind password when prompted, the record is added to the database. If any of those conditions is false, however, the action fails and ldapadd tells you what went wrong. Thus, you can use trial and error to craft a workable record format. After you've figured this out the first time, you can use the same format for subsequent records, without going through all this schema-induced zaniness.

I offer one caveat: say your LDIF file contains multiple records, which is permitted, if your LDAP server detects an error, it quits parsing the file and does not attempt to add any records below the one that failed. Therefore, you should stick to single-record LDIF files for the first couple of user adds, until you've finalized your record format.

The manual record creation method is a little clunky, but it accommodates a certain amount of tinkering. This is especially useful in the early stages of LDAP database construction.

Once you have a user record or two in place, you can use a GUI tool such as LDAP Browser/Editor (www.iit.edu/~gawojar/ldap) or gq (included in most Linux distributions) to create additional records. In gq, for example, left-clicking on a record pops up a menu containing the option New→Use current entry, which copies the selected record into a new record. This is much faster and simpler than typing everything into an LDIF file manually.

Creating Passwords

I mentioned in the description of Listing 1 that we generally don't specify user passwords in LDIF files. A separate mechanism is used for that, in the form of the command ldappasswd. By design, its syntax is similar to that of ldapadd:

```
bind-$ ldappasswd -S -x -D
"cn=hostmaster,dc=upstreamsolutions,dc=com" \
-W "cn=Phil Lesh,dc=upstreamsolutions,dc=com"
```

You don't need to be logged in to a shell session on the LDAP server to use the ldappasswd command. You instead can use the -H option to specify the URL of a remote LDAP server, like this:

```
$ ldappasswd -S -x \
-H ldaps://ldap.upstreamsolutions.com \
-D "cn=hostmaster,dc=upstreamsolutions,dc=com" \
-W "cn=Phil Lesh,dc=upstreamsolutions,dc=com"
```

This option also may be used with ldapadd.

The ldaps:// URL is required in the above example. I've specified the -x option for simple clear-text authentication, so I definitely need to connect to the server with TLS encryption rather than in the clear. Last month, I showed how to set up an LDAP server to accept TLS connections.

Having said all that, however, I must point out that password management for end users is one of LDAP's problem areas. On the one hand, if your users all have access to the ldappasswd command, you can use a combination of local /etc/ldap.conf files and scripts/front ends for ldappasswd to make it reasonably simple for users to change their own passwords.

But for users who run some other OS, you must manage passwords centrally and have all users contact the e-mail administrator every time they need to change their password, or you must install LDAP client software for their OS. For client systems running Microsoft Windows, you can configure Samba to let users change their LDAP password with the Windows password tool. See the article “OpenLDAP Everywhere” in *LJ*, December 2002.

Access Controls

Technically, we've covered or touched on all the tasks needed to build an LDAP server using OpenLDAP (excluding, necessarily, the sometimes lengthy step of actually getting your various server applications to authenticate users against it successfully). In the interests of robust security, a concept not alien to readers of this column, we need to discuss one more thing: OpenLDAP access control lists (ACLs).

As with most other things affecting the slapd dæmon, ACLs are set in /etc/openldap/slapd.conf. And, like most other things involving LDAP, ACLs can be confusing to say the least and usually require some tinkering to get right. Listing 2 shows a sample set of ACLs.

Listing 2. ACLs in /etc/slapd.conf

```
access to attrs=userPassword
    by dn="cn=ldapguy,dc=wiremonkeys,dc=org" write
    by self write
    by * compare

access to *
    by dn="cn=ldapguy,dc=wiremonkeys,dc=org" write
    by users read
    by * auth
```

ACLs are described in detail in the slapd.conf(5) man page, but in Listing 2 you can see generally how they work. For each LDAP element to which you wish to control access, you specify who may access it and with what level of access. Technically, an entire ACL can be listed on one line, but by convention we list each “by...” statement on its own line. slapd is smart enough to know that the string “access to” marks the beginning of the next ACL.

Space doesn't permit my describing ACL syntax in detail, but remember a few important points. First, ACLs are parsed from top to bottom, and first match wins; they act like a stack of filters. Therefore, it's crucial that you put specific ACLs and by statements above more general ones. For example, in Listing 2 we see an ACL restricting access to the userPassword attribute, followed by one applicable to *, that is, the entire LDAP database. Putting the userPassword ACL first means the rule that allows users to change their own passwords (access to attrs=userPassword by self write) is an exception to the more general rule stating users may read anything (access to * by users read).

Another important point is access levels are hierarchical. Possible levels are none, auth, compare, search, read and write, where none is the lowest level of access and write is the highest, and where each level includes the rights of all levels lower than it. These two points, the first match wins rule and the inclusive nature of access levels, are crucial to understanding how ACLs are parsed. They also are important for making sure your ACLs don't lead to either greater or lesser levels of access than you intend in a given situation.

Conclusion

LDAP is one of the most complicated technologies I personally have worked with lately. To make it work the way you need, you have to spend a lot of time testing while watching logs and fine-tuning the configurations of both the LDAP server and the applications you wish to authenticate against it. But, having such a flexible, powerful and widely supported authentication and directory mechanism is well worth the trouble. I hope this series of articles has helped you get there or at least pointed you in the right direction.

Resources

OpenLDAP software and documentation, including the important “OpenLDAP Administrator's Guide”: www.openldap.org.

List of error codes used in LDAP error messages. This is essential in interpreting LDAP log messages: www-user.tu-chemnitz.de/~fri/web500gw/errors.html.

The Exchange Replacement HOWTO, which describes how to use LDAP as the authentication mechanism for Cyrus-IMAPD: www.arrayservices.com/projects/Exchange-HOWTO/html/book1.html.

Carter, Gerald. *LDAP System Administration*. Sebastopol, California: O'Reilly & Associates, 2003. An excellent new book with detailed coverage of OpenLDAP.

Mick Bauer, CISSP, is *Linux Journal*'s security editor and an IS security consultant for Upstream Solutions LLC in Minneapolis, Minnesota. Mick spends his copious free time chasing little kids (strictly his own) and playing music, sometimes simultaneously. Mick is author of *Building Secure Servers With Linux* (O'Reilly & Associates, 2002).

Archive Index Issue Table of Contents

Advanced search

EOF

The Open Source Development Lab

Stuart Cohen

Issue #113, September 2003

What does OSDL do, besides hire Linus Torvalds?

You can tell a technology finally has made it into the general consciousness when it's the cover story in a major business magazine. With all the recent media attention on Linux, it would be easy to assume that the awareness battle has been won and now, finally, things are going to get easier. But those of us on the front lines of business computing know the real heavy lifting has only started. The penguin has been a symbol of promise and fundamental change in software development for some time, but now is the time for it to really prove itself.

The fact that the broader business community is starting to take Linux seriously is a good thing: more consideration will be given to Linux, and more development energy within organizations will be devoted to Linux applications. But with all of this new attention—sometimes by people and organizations having little experience with the Open Source community and its workings—comes a certain degree of uncertainty. And if there's anything corporate IT departments don't like, it's uncertainty—especially when it comes to business-critical applications. As Linux makes its move from edge-of-the-network to the data center, Linux applications will be under heavier stress and flaws will be highly visible.

Despite its pervasiveness as a Web server platform and its maturity, thanks to a vibrant and committed development community, Linux still has to prove itself as a true enterprise platform. IT management needs to know that security, scalability and availability all are on par with proprietary systems. Application developers, both independents and those who work in corporate IT departments, need access to test equipment that replicates a corporate data center to validate their code and make the proper modifications. Enter the Open Source Development Laboratory (OSDL).

Created as a nonprofit corporation by a consortium of technology companies in August 2000, OSDL is dedicated to accelerating the adoption of Linux in corporate computing. The Lab is a place where kernel and middleware code can be stress-tested and hardened to support enterprise Linux applications. In addition to facilitating application projects, we also are actively involved in Linux kernel and middleware development. Recently, OSDL has contributed to the development of a new device module for Linux 2.5, as well as accomplished significant work on stability enhancements.

OSDL was created to give open-source developers access to data center-like equipment to test their applications and receive technical as well as moral support. Although OSDL's charter has expanded over the past year, the core of its original mission remains: provide open-source developers with resources and guidance to build data center and telco class enhancements into Linux and its open-source software stack.

OSDL offers three unique programs focused on developers and designed to accelerate the growth and adoption of Linux in the enterprise:

1. A fully configured data center environment for Linux development and testing, available for qualified projects around the world.
2. Creation of enterprise-class development tools and performance test suites for corporations, ISVs and other Linux developers.
3. Hosting and coordination of global initiatives that define requirements and harden Linux to meet reliability, availability and performance requirements for telecommunication and data center environments.

Since its creation, OSDL has supported more than 200 Linux projects ranging from Apache to virtual memory improvements. The Lab's test suite includes x86 and Itanium systems up to 32-way setups. Lab resources are available to new or existing Linux projects.

Our development tools include an automated scalable test platform (STP) for Linux that provides a repeatable set of tests to verify how well patches and enhancements perform in enterprise computing environments. We also provide a patch life-cycle manager (PLM) to verify that patches compile on the Linux kernel prior to STP testing. We want to help corporate users better understand what open-source software can do for their businesses, as well as how they can become active participants in the community. As you can see, we have our work cut out for us. But, so do you. I encourage you to visit the Lab at www.osdl.org and see what we're up to. Perhaps you can help us.

Stuart Cohen is CEO of the Open Source Development Lab.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

C++ Minuses

Just read your column about C++ [*LJ*, June 2003]. It is deliberately provocative, and as such, it will get you a large number of well-deserved replies. One point I wanted to make: C++ is getting old and still is not there. It is way away from being the main programming language, and I can see two main reasons for that.

The first is lack of standardization. C++ got ANSI less than ten years ago and still has no standardized ABI. Net result: installing C++-dependent software and libraries is a nightmare. If you ever tried to compile some KDE apps yourself you know what I mean. It does not stop there, however; the fact that you cannot link together two objects compiled with different compilers prevents vendors from easily distributing binary libraries. Not only a technical issue, it also becomes a commercial one.

The second reason is complexity. C++ has accumulated so many features over the years that it has become incredibly complex. I believe I can safely say that nobody can master all of it. Result: put together a team of, say, ten people, each of them mastering 90% of the language (that is, really good C++ programmers). Chances are that the 10% they don't know will not be the same for all programmers. You will end up with pieces of code that can be understood only by the people who wrote them. This is a disaster waiting to happen. Most project managers will refuse to sign for a language that will bring more trouble than it may solve. You just started a language war; you expected such things to happen, didn't you?

—
Nicolas

Perl-Based Shell

In the April 2003 issue, David Bandel mentions psh as an alternative shell to bash. Although psh is the most feature-rich Perl shell I've seen, it doesn't seem to be lighter as David claims. I'm running psh 1.8 on my Gentoo box, with Perl 5.8.0. **top** shows psh resident in 3964k of RAM, while showing bash at only

1440k. It would be great to see future optimization of this shell, because Perl is already competing with traditional shells as the de facto scripting language on many systems.

—
Christopher J. Pilkington

Free, Yes. Cheap, No.

I manage several research networks where I have been using Linux (Red Hat mostly) for quite a few years and have been very pleased with not only the functionality but the great response from the Open Source community. This week, I received a somewhat unsettling call from a Red Hat sales rep when I asked for info on their Red Hat Enterprise Linux AS operating system. I was told that on December 31, 2003, Red Hat will discontinue support for their OS, with the exception of their Enterprise line, priced at $1,499–$2,499 per year, per server. I asked what the Enterprise version offered that I did not get with the versions I was currently using, and was told I would get support and an extended life on the kernel version (2.4.9). I am a firm believer in trying to support development in the Open Source community, and paying for a good OS is just part of doing business, but $1,499–$2,499 per server actually seems more expensive than the Microsoft offering, Sun Microsystems or Apple OS X Server. How do the hardworking programmers who develop the open-source code feel about someone selling it for what appears to be a substantial profit?

—
Robert Christner
MIS/MIT MIND Institute

More on the Zaurus, Please

I recently purchased a Sharp Zaurus 5600 and am very, very glad I saved a copy of your January 2003 magazine containing a good article about the Sharp Zaurus. I had a few questions about things I had read and sent a message to Guylhem Aznar, who promptly responded and quickly helped me. Keep up the good work! Thank you for producing such a great magazine! Keep the Zaurus articles coming!

—
Jonathan M. Rose
President, Farious Net Solutions

Does “Public Domain” Software Exist?

The *Linux Journal* October 2002 article, “Why the Public Domain Isn't a License” (/article/6225) seems to disagree with prevailing wisdom as well as this rather official-looking CMU page: www.cmu.edu/innovationtransfer/Home/documents/ipg6.html. More specifically, they seem to disagree about whether something can be placed in the public domain by a deliberate act. Is there any way of clearing up this confusion?

—
Dan

Lawrence Rosen replies: I wrote the article to clear up confusion but apparently that hasn't helped. I don't recommend dedicating software to the public domain and am not convinced such a dedication is fully effective. Instead, use a license (like the BSD license) that accomplishes everything you want without disclaiming ownership.

Thumbs-up on Nagios

I was gratified to see Richard Harlan's article “Network Management with Nagios” in the July 2003 issue. We converted to Nagios last fall after using another popular and widely available management tool for several years. The old tool was free as in “free beer”, but its license severely restricted our use of it in some emerging business opportunities, so we went looking for an open-source replacement. Nagios fit the bill nicely. The flexibility is awesome. Using the plugins from the APAN Project (apan.sourceforge.net), we were able to use Nagios to accommodate our Multi-Router Traffic Grapher (people.ee.ethz.ch/~oetiker/webtools/mrtg) monitoring of traffic through network devices. It was then easy to extend Nagios to use Round Robin Database (people.ee.ethz.ch/~oetiker/webtools/rrdtool) to record and graph such diverse indicators as ping times, disk utilization, CPU and memory usage, RAS dial-up sessions, IPSec VPN sessions and database license usage. The visibility it gives us into our network has proven its worth multiple times from identifying software with a memory leak on one server to finding a piece of spyware saturating a network segment on another machine. This is a tool that should be in every network admin's bag of tricks.

—
Rob Embry

Screenshots, Please

I've been reading Reuven Lerner's articles for several months now. Although they're very good articles, he never includes any graphic examples. One article mentioned how great and customizable the user interface was, with different color options. Again, no examples. Please Reuven, show us something.

PS: Marcel's Cooking With Linux is the best! Don't ever change.

—
Callum Benepe

Archive Index Issue Table of Contents

Advanced search

UpFront

diff -u: What's New in Kernel Development

Zack Brown

Issue #113, September 2003

Kernel development seems to be inching toward a **2.6** release. The October 31, 2002 feature-freeze has not held firm, and many new features have been going into the kernel ever since. However, **Linus Torvalds** and others have been more strict about how invasive any new features can be; in early May 2003, Linus said he would begin rejecting patches that were not completely obvious, unless they went through a lot of other folks in the chain of command. My guess is the chance of seeing a 2.6 kernel before the end of 2003 is slim, but we may see a **code freeze** by September or October 2003.

Linus Torvalds has made this very clear: he will accept patches supporting **Digital Rights Management** (DRM). DRM is intended to deprive users of control over how data on a system may be used. For instance, with DRM, a digitized song might be played only a certain number of times before requiring the user to pay money to a vendor. DRM is a generic term referring to any attempt to exert this kind of control over how data is used. It may be implemented in software or hardware. In the free software and open-source world, DRM is regarded as a terrible threat, possibly to the very existence of free and open software. Linus feels the fundamental kernel features involved in DRM are not bad in themselves, but represent generic security measures that could be used for DRM or for something else. Some patches already have been proposed, and by adopting them or others into the kernel, Linus may be creating a foothold to prevent the eventual criminalization of free software.

The **HFS+** filesystem, the default on Mac OS X systems, has been given a fresh infusion of life by **Roman Zippel**. On behalf of Ardis Technologies, he took the existing HFS+ driver by **Brad Boyer** and made a big batch of improvements, including full read and write support, better performance and support for hard links. This is a real benefit for iPod and other Mac users, but unfortunately, as is so often the case, Roman forgot to tell Brad what he was doing, so there was a bit of a skirmish over maintainership when Roman announced the new code. As of this writing, it remains unclear as to whether Roman or Brad will lead the project or whether something else will happen.

The **National Security Agency** (NSA) has made some API changes to their **SE Linux** distribution as part of a push to get SE Linux patches into the main 2.5 tree in time for 2.6. Their changes include support for extended attributes on the ext3 filesystem. This would allow virtually any kind of additional file metadata to be stored with a file. **Extended Attributes** (EAs) can be used to implement security features such as **Access Control Lists** (ACLs) and filesystem capabilities. The **Linux Security Module** (LSM), at the back end of SE Linux, also received some attention from the NSA. They modified some of the module's hooks in order to prevent security exploits during the fraction of a second between when a file security label is changed and the corresponding inode security field is updated to reflect that fact.

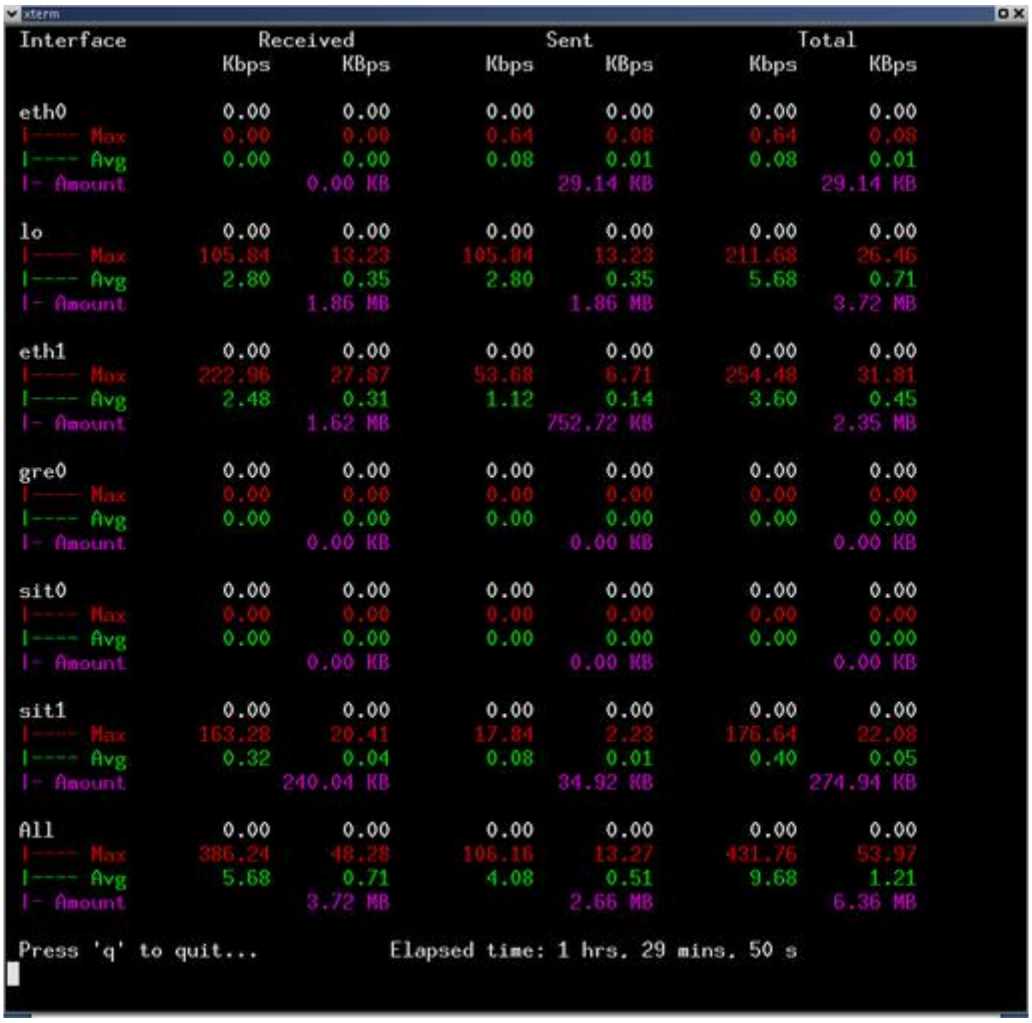
It seems that some **Wireless LAN** chips (Broadcom's BCM4306 and BCM2050, to name two) are capable of receiving military communications and transmitting at those same frequencies. This is slowing down the creation of free drivers for these chips, because the hardware manufacturers can release the chip specifications only at the risk of angering various government agencies around the world. As one might expect, once the “features” of these chips became known, many hackers dropped whatever they were working on and started reverse engineering the chips with all their might, in some cases using the existing **Microsoft Windows** drivers to guide them.

ibmonitor: ibmonitor.sourceforge.net

David A. Bandel

Issue #113, September 2003

For those who like to watch traffic while it's passing, ibmonitor allows you to do this. It shows you maximum traffic, data throughput and average traffic as well as current traffic. Display is designed to fit on the standard VT, although that depends on what you have turned on and how many interfaces you have. Requires: Perl, Perl module Term::ReadKey, Getopt::Long, Term::ANSIColor and Data::Dumper.



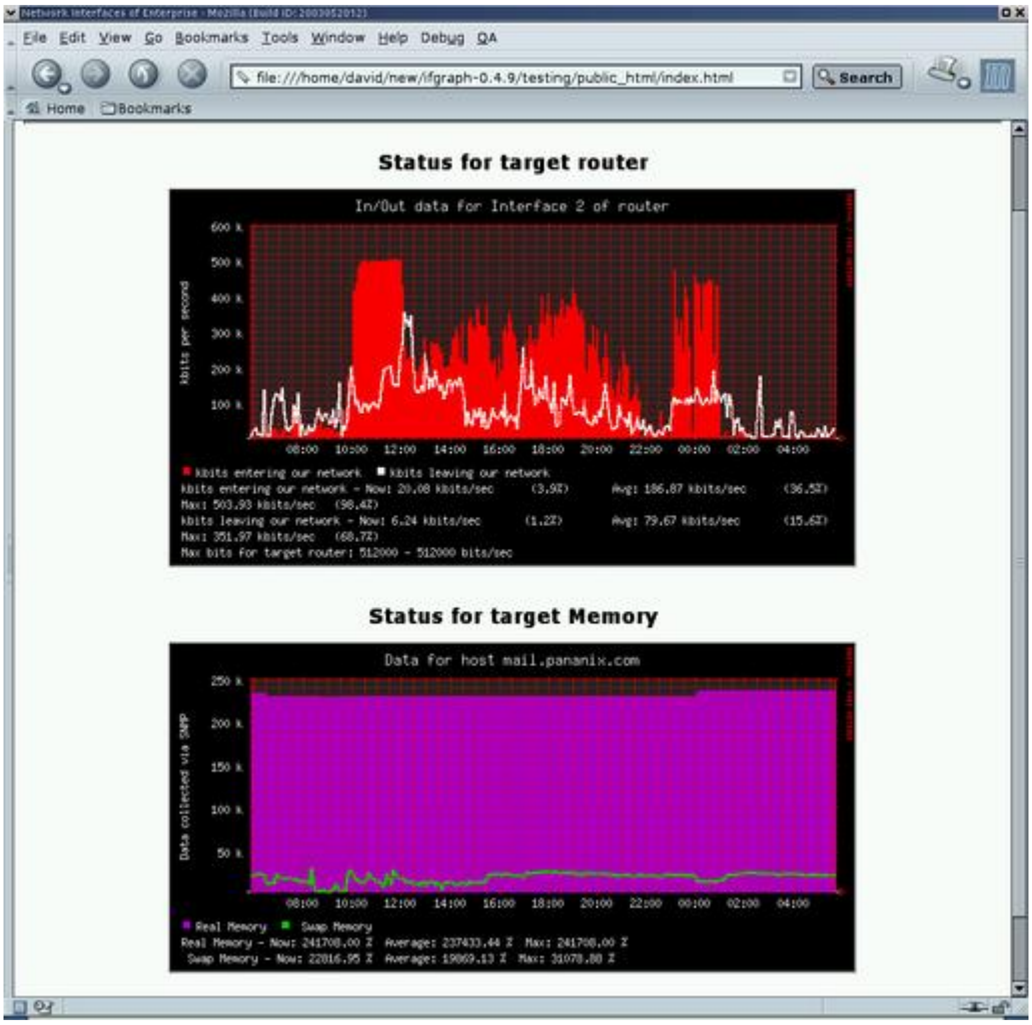
```
Interface            Received                 Sent                   Total
                   Kbps       KBps        Kbps       KBps        Kbps       KBps

eth0               0.00       0.00        0.00       0.00        0.00       0.00
|----- Max         0.00       0.00        0.64       0.08        0.64       0.08
|----- Avg         0.00       0.00        0.08       0.01        0.08       0.01
|- Amount                  0.00 KB               29.14 KB               29.14 KB

lo                 0.00       0.00        0.00       0.00        0.00       0.00
|----- Max       105.84      13.23      105.84      13.23      211.68      26.46
|----- Avg         2.80       0.35        2.80       0.35        5.68       0.71
|- Amount                  1.86 MB                1.86 MB                3.72 MB

eth1               0.00       0.00        0.00       0.00        0.00       0.00
|----- Max       222.96      27.87       53.68       6.71      254.48      31.81
|----- Avg         2.48       0.31        1.12       0.14        3.60       0.45
|- Amount                  1.62 MB              752.72 KB                2.35 MB

gre0               0.00       0.00        0.00       0.00        0.00       0.00
|----- Max         0.00       0.00        0.00       0.00        0.00       0.00
|----- Avg         0.00       0.00        0.00       0.00        0.00       0.00
|- Amount                  0.00 KB                0.00 KB                0.00 KB

sit0               0.00       0.00        0.00       0.00        0.00       0.00
|----- Max         0.00       0.00        0.00       0.00        0.00       0.00
|----- Avg         0.00       0.00        0.00       0.00        0.00       0.00
|- Amount                  0.00 KB                0.00 KB                0.00 KB

sit1               0.00       0.00        0.00       0.00        0.00       0.00
|----- Max       163.28      20.41       17.84       2.23      176.64      22.08
|----- Avg         0.32       0.04        0.08       0.01        0.40       0.05
|- Amount                240.04 KB               34.92 KB              274.94 KB

All                0.00       0.00        0.00       0.00        0.00       0.00
|----- Max       386.24      48.28      106.16      13.27      431.76      53.97
|----- Avg         5.68       0.71        4.08       0.51        9.68       1.21
|- Amount                  3.72 MB                2.66 MB                6.36 MB

Press 'q' to quit...            Elapsed time: 1 hrs. 29 mins. 50 s
```

ifGraph: www.ifgraph.org

David A. Bandel

Issue #113, September 2003

If you find, for whatever reason, that MRTG isn't the tool for you, but need something to show you the I/O on a router interface (or even a host interface, like your mail or Web host), check out ifGraph. By default, it creates graphs for interface statistics and memory usage but should be adaptable to most anything you can collect with SNMP. Requires: Perl, Perl modules FindBin, Getopt::Std, Net::SNMP, File::Copy and rrdtool.


Network Interfaces of Enterprise - Mozilla (Build ID: 2003052012)
File Edit View Go Bookmarks Tools Window Help Debug QA
file:///home/david/new/ifgraph-0.4.9/testing/public_html/index.html
Search
Home
Bookmarks

Status for target router

In/Out data for Interface 2 of router
kbits per second
600 k
500 k
400 k
300 k
200 k
100 k
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
00:00
02:00
04:00
🟥 kbits entering our network
⬜ kbits leaving our network

```
kbits entering our network - Now: 20.08 kbits/sec      (3.9%)      Avg: 186.87 kbits/sec      (36.5%)
Max: 503.93 kbits/sec   (98.4%)
kbits leaving our network - Now: 6.24 kbits/sec       (1.2%)      Avg: 79.67 kbits/sec      (15.6%)
Max: 351.97 kbits/sec   (68.7%)
Max bits for target router: 512000 - 512000 bits/sec
```

Status for target Memory

Data for host mail.pananix.com
Data collected via SNMP
250 k
200 k
150 k
100 k
50 k
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
00:00
02:00
04:00
🟪 Real Memory
🟩 Swap Memory

```
Real Memory - Now: 241708.00 %  Average: 237433.44 %  Max: 241708.00 %
 Swap Memory - Now: 22816.95 %  Average: 19869.13 %  Max: 31078.80 %
```

LJ Index—August 2003

- 1. Number of Windows desktops in London's Newham district expected to be replaced by Linux and open-source applications: 5,000
- 2. Numbers of Windows desktops expected to do the same if Nottingham follows Newham's lead: 6,500
- 3. Expected cost cuts in both cases: 1/3
- 4. Number of computers migrating to Linux in Munich: 14,000
- 5. Number of police computers in Lower Saxony expected to switch to Linux: 11,000
- 6. Percentage of computers sold in Western Europe on which Linux can be found: 15
- 7. Percentage of broadband usage growth in Europe in 13 months ending April 2003: 136
- 8. Percentage of the same growth in the UK: 235
- 9. Percentage of European users now connected by broadband: 28
- 10. Percentage of US users now connected by broadband: 35
- 11. Percentage of Hong Kong users connected by broadband: 85

- 12. Millions of Europeans who will be using broadband by March 2004: 50
- 13. Percentage of Australian McDonald's restaurants equipped with Wi-Fi: 100
- 14. Number of rural communities in the Loni-Shirdi area of western Maharashtra that have raised money to form a technology cooperative: 200
- 15. Number of Wi-Fi hot spots established by the cooperative as of May 2003: 50
- 16. Idle percentage range for one provider's 350,000km of fiber-optic cable in India: 60–70
- 17. Billions in Wi-Fi device sales expected by 2006: 2.3
- 18. Millions of Wi-Fi users Cometa expects to see by 2008: 50
- 19. Millions of Wi-Fi devices Cometa expects to see by 2008: 100
- 20. Number of hot spots Cometa plans to install by February 2004: 5,000
- 21. Number of hot spots Cometa plans to install by 2005: 20,000

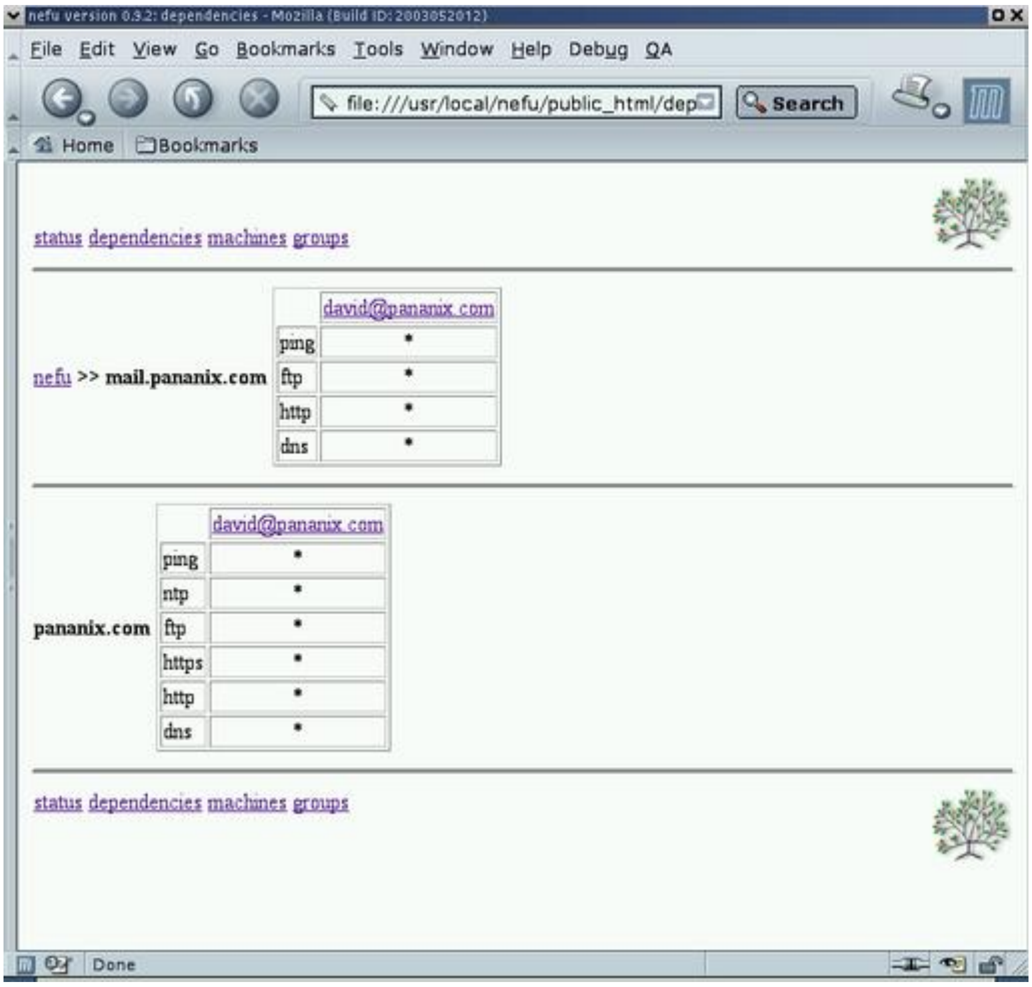
- 1–6: ZDNet UK
- 7–12: Nielsen/NetRatings
- 13: *New York Times*
- 14–16: *The Hindu*
- 17: Jupiter Research
- 18–21: Reuters

nefu: rsug.itd.umich.edu/software/nefu

David A. Bandel

Issue #113, September 2003

nefu, another monitor for services on your network servers, can test for a number of well-known services, or you can write your own scripts. One thing nefu understands that some others don't is dependencies. That is, if you tell nefu about a router between it and your mail server, and the router goes down, the router is noted as down in an e-mail to you, not the mail server, which may be down. Requires: libresolv, libnsl, libssl, libcrypto, glibc and libdl.


nefu version 0.3.2: dependencies - Mozilla (Build ID: 2003052012)
File Edit View Go Bookmarks Tools Window Help Debug QA
file:///usr/local/nefu/public_html/dep
Search
Home
Bookmarks
Done

status dependencies machines groups

nefu >> **mail.pananix.com**

	david@panarix.com
ping	*
ftp	*
http	*
dns	*

pananix.com

	david@pananix.com
ping	*
ntp	*
ftp	*
https	*
http	*
dns	*

status dependencies machines groups

The Linux Option for School Computers

Walt Pennington

Issue #113, September 2003

Mandrake Linux is now available on free computers for schools and nonprofits in California from the Technology Training Foundation of America (TTFA). Since 1998, TTFA has accepted donated computers from corporations in California and provided computer access to more than 40,000 students. First, TTFA finds corporations wishing to recycle computer hardware. Many corporations replace between 20–30% of their desktop computers annually, and TTFA eagerly anticipates those 2- to 4-year-old computers.

TFTA uses the services of refurbishing partners throughout the state of California. TTFA coordinates the transportation of the computers to one of its 16 refurbishing partner sites, nine of which are run by the California Department of Corrections (CDC), that meet the applicant's technology standards. The refurbishing partners remove all data from the hard drives, repair or replace components as necessary and configure the computers to meet the needs of the recipient.

CDC and TTFA offer three choices for operating systems and software, including Mandrake Linux 9.0 with OpenOffice.org, The GIMP and other standard software packages. TTFA and CDC have the ability to install additional software but require proof of licensing before installing proprietary software.

TTFA accepts working monitors and computers with a speed of 300MHz or faster. TTFA is a statewide 501(c)(3) nonprofit organization located in San Diego, California. More information about TTFA and its computer donation program can be obtained from their Web site at www.computers2learnby.org.

They Said It

People object that there is no economic model for it, but there is: the economic model of flower-boxes...I put out flower-boxes to raise my self-esteem and make my house look more attractive. If almost everyone does it then the whole town becomes more beautiful. The same thing can happen with communications. The challenge for telecom is to combine wireless with very intelligent devices....The combination will allow us to use spectrum in a very different way, in more efficient and organic ways than before.

—Nicolas Negroponte (Nordic Wireless Watch, www.nordicwirelesswatch.com/wireless/story.html?story_id=3152)

We were discussing the future challenges in information technology, including the issues related to software security...I made a point that we look for open-source codes so that we can easily introduce the users-built security algorithms. Our discussions became difficult, since our views were different....The most unfortunate thing is that India still seems to believe in proprietary solutions....Further spread of IT, which is influencing the daily life of individuals, would have a devastating effect on the lives of society due to any small shift in the business practice involving these proprietary solutions. It is precisely for these reasons open-source software needs to be built, which would be cost-effective for the entire society. In India, open-source code software will have to come and stay in a big way for the benefit of our billion people.

—A.P.J. Abdul Kalam, President of India (ZDNet News/India, news.zdnet.co.uk/story/0,,t272-s2135401,00.html)

We know that Linux is not for everything....But there are not many applications that require more than Linux can give us.

—Mark Snodgrass, VP Global Technology & Services, Merrill Lynch (CNET, news.com/2100-1016_3-1014287.html)

There's no fancy equipment, so a lot of what we use as test equipment turns out to be hacked development boards, blinking LEDs, and occasionally a loud buzzer which scares the hell out of everyone.

—Andrew Greenberg of the Portland State Aerospace Society, which uses Linux and open-source code to control suborbital satellite launches

Be a guest on Jay Leno—not exactly, but your TiVo can claim you are. With the simple use of some scripts to run on your TiVo, you can change the descriptions of the television programs to guest star you.

—Rafi Kirkorian on TiVo hacks (Wasted-Bits, www.bitwaste.com/wasted-bits/index.cgi/media/books/tivo-hacks)

w3m: w3m.sourceforge.net

David A. Bandel

Issue #113, September 2003

Three years ago I reviewed several good programs, but the hands down winner for me was the text-based browser w3m. With the large majority of systems today using frames and images, using a text browser on the Internet can be a frustrating experience. Although great for slow links or systems not running X, this is not ideal. However, w3m works well with frames, allowing you to see multiple frames on the screen. It also, depending on your build options, allows you to see some graphics. The older Lynx has long since been replaced on my system with w3m. Requires: libgc, libpthread, libm, libnsl, libgpm, libncurses, libssl, libcrypt o, glibc, libstdc++, libdl and libgcc_s.

watchfolder: dstunrea.sdf-eu.org/files/watchfolder-02.2-p3.tar.gz

David A. Bandel

Issue #113, September 2003

Forgiving the misleading name, watchfolder watches directories for additions, then takes action on those additions. It runs as a dæmon and can act on new files or changed files as you configure it. You do need to be careful of permissions, but it's perfect for a repetitive task. I print files in Mozilla as PostScript into a watchme directory, and watchfolder handles converting them to PDFs and moving them into a another directory. Requires: glibc.

Watching the World

Doc Searls

Issue #113, September 2003

Linux Journal has added a new publication to its on-lineup: *WorldWatch* (WorldWatch.LinuxGazette.com). “Linux is a truly global phenomenon with a diversity of developments occurring in different societies. Therefore we're eager to transcend national and cultural borders in our articles, viewpoints and analyses”, says Willy Smith, editor in chief of the new publication. *WorldWatch* offers a daily digest of articles from publications around the world about topics surrounding Linux and open-source software. These topics are generally nontechnical and offer big-picture perspectives on the growth of Linux and its impact on societies. A sophisticated feedback system allows readers to comment, interact and elaborate on each issue.

Archive Index Issue Table of Contents

Advanced search

From the Editor

Wireless Networking

Don Marti

Issue #113, September 2003

Remember how fun it was to get your first Net connection working? It's like that.

How about a project that combines hardware construction, community building, network hacking and, of course, Linux and other free software? Best of all, the stuff you need to get started is cheap and standardized, and there's a great balance of helpful resources and unanswered questions. We're talking about wireless networks. You'll impress even people who aren't Linux users by offering convenient Net access at your business or organization and the public spaces nearby.

Here on the West Coast of the USA, people sometimes get complacent about being first to understand and deploy new technology. If you're in San Francisco or Seattle, you're bound to be living a couple years ahead of everyone else, gadget-wise, right? Wrong.

Even though the West Coast has some wireless users groups that have done some impressive work, wireless is really making an impact in New York City, where “business improvement districts” are using access points to improve business in their neighborhoods, and even the phone company is offering access points. NYCwireless is a force to be reckoned with and might give you some ideas for your town. Find out exactly what that beat box on the cover has to do with Linux, and learn about the wireless frenzy sweeping New York City, on page 42.

When Doc's article has you typing “LGA” into your favorite travel site, be sure to pack your favorite Linux PDA or laptop, and bring along a copy of Kismet. Tony Steidler-Dennison shows you how to discover all the wireless resources available to you, on page 58.

Meanwhile, the author of Kismet, Mike Kershaw, explains how to set up your very own access point, with NoCatAuth and a friendly login screen for security, on page 52. If you merely leave your access point open, people might hesitate to use it because they're polite or don't know what your intentions are. Change its name to something with “public” or “open” in it, and put up NoCatAuth so that people can sign in and understand the terms under which they're allowed to use it.

As always, networks are most useful and fun when you can connect them to real-world devices. Tad Truex has a sump pump on the Web, and you can learn to hook up your own electrical appliances on page 38. Remember, safety first, and read the part about not burning your house down.

Often, you need a big directory that handles everyone's information for your business, and the last thing you want is to get locked in to some vendor's idea of how to do it, so naturally you've been reading Mick Bauer's series on OpenLDAP. This month, on page 32, Mick finishes the series. If you're a new subscriber, check out interactive.linuxjournal.com for the previous two articles in the series.

Finally, don't miss the chance to get your Web site onto a content management system (CMS) that helps everyone do his or her job better and release pages to the public at the right time. Reuven Lerner offers Bricolage for your consideration on page 16.

Don Marti is editor in chief of *Linux Journal*.

On the Web

FUD vs. Freedom

LJ Staff

Issue #113, September 2003

Watch our Web site for facts on the SCO case.

If you typed “Linux” into Google News or any other news aggregator site recently, you probably got a bunch of links to the SCO vs. IBM lawsuit, the information technology industry's most bizarre legal battle. Because the case is moving way too fast to keep up with in a monthly publication, we're running our articles about the facts of the case on our Web site.

We're not writing down merely the latest “Fear Uncertainty and Doubt” (FUD) that SCO execs Darl McBride and Chris Sontag are giving the mainstream media. You know us better than that. We've got articles that bring a little more insight to the case.

In August 2002, our correspondent Jeff Gerhardt published the first hint of SCO's search for money in the dusty file cabinets of UNIX licenses, and he quoted McBride as saying, “obviously Linux owes its heritage to UNIX, but not its code. We would not, nor will not, make such a claim” (www.linuxjournal.com/article/6293).

A lot has changed since then, and Doc Searls covered the initial complaint in March 2003, when Sontag said, “I have to say that this is not an issue regarding the Linux community. This is an issue between SCO and IBM.”

Later, SCO pulled its own Linux distribution, and Sontag claimed that there is “significant copyrighted and trade secret code within Linux”.

Our May 15 story was the first public mention of SCO's offer to let independent experts review the allegedly copied code under a nondisclosure agreement (NDA), available at www.linuxjournal.com/article/6877.

Later, we published the full text of the NDA and decided it was too legally risky to accept it ourselves (www.linuxjournal.com/article/6923).

But, Ian Lance Taylor, the mastermind behind Taylor UUCP, came to the rescue, signed it and walked into SCO's office. His essay on the journey revealed tantalizing facts about an 80-line function that is in fact the same in SCO UnixWare and Linux—but also found elsewhere (www.linuxjournal.com/article/6956). He wrote:

> The similar portions of the code were some 80 lines or so. Looking around the Net, I found close variants of the code, with the same comments and variable names, in sources other than Linux distributions. The code is not in a central part of the Linux kernel. The code does not appear to have been contributed to Linux by SCO or Caldera. The code exists in current versions of the Linux kernel.

What's next? By the time this issue goes to press, anything could happen, so watch our Web site for the facts. After all, you can get FUD anywhere.

Best of Technical Support

Red Hat without a Network Card?

Is it possible to run my Red Hat 8.0 system without a network card?

—
Prasanna

softpras@rediffmail.com

Sure, it's possible, and you don't have to do anything. Any normal Linux system configures lo (the loopback interface) automatically. It's even possible to run without that (as embedded Linux systems sometimes do). However, you shouldn't have to do anything special. Simply install your distribution and refrain from entering any network settings.

—
Jim Dennis

jimd@starshine.org

Incompatible Web Site?

I recently used Konqueror 2.2.2 to visit a credit-card Web site (URL withheld to protect the guilty) to check my account. The site said I needed to upgrade to at least Netscape 4.0 or Microsoft Internet Explorer 4.0 in order to use the 128-bit-strong encryption their site used. I know that Konqueror 2.2.2 is a much later version than 4.0 of either of the browsers they mentioned. I checked the various browser settings, and sure enough, SSL2 and 3 both were enabled, with encryptions going as high as 168 bits. Almost every encryption standard was enabled in my settings; the exceptions being FZA-FZA-CBC-SHA, FZA-NULL-SHA, NULL-MD5 and NULL-SHA—they all say “0 of 0 bits”. Finally, I changed the user agent setting to broadcast that it was Netscape or MSIE, and magically, I stopped getting the error messages. Now when I fill in the forms, such as the login screen, my input is ignored and the form simply refreshes. I have enabled JavaScript, Java and cookies. Nothing works. Am I doing something wrong or is the site nonstandard? If I look in my settings under certificates, it says I have a certificate from the Web site. I can verify that certificate, but it does not say what type of encryption they use or give me any other useful information.

—
John Handis

mrintensity@worldnet.att.net

I'll bet the Web site uses the user agent string to help identify what it thinks is a valid browser version. Try Mozilla. If that doesn't work complain to the Web site.

—
Christopher Wingert

cwingert@qualcomm.com

Too many Web developers code to a particular implementation (target platform) rather than to the standard protocols and APIs (application programming interfaces). This is exacerbated considerably by JavaScript. Writing any piece of nontrivial JavaScript so it can run correctly on several different Web browsers is daunting. I encourage all Web site developers to start with the simplest implementation of the core requirements. Add bells and whistles in JavaScript, but always allow the user to get at core functions without it. My suggestions: complain to your bank and try using Mozilla 1.x or Netscape 6.

—
Jim Dennis

jimd@starshine.org

There is a silver lining, however. You may not be aware of this, but the default Apple browser, Safari, is based on the KHTML rendering engine from the KDE Project. This, obviously, is the same engine that Konqueror uses. Also, now that Microsoft discontinued development on the Macintosh version of Internet Explorer, Web developers who code for Windows and Mac only will be coding for you too.

—

Ben Ford

ben@kalifornia.com

It's Hot in Here

I am using an ASUS motherboard. Under Microsoft Windows 98, a program called asusprobe reports that the CPU temperature is about 47°C/116°F and the motherboard temperature is about 31°C/87°F. These may change slightly but are pretty steady. How do I find these temperatures under Linux?

—
Michael Mather

mmather@eol.ca

Most commodity motherboards that provide this information are built around the LM78 series of chips that communicate over the i2c 2-wire bus. The SMBus is a particular implementation of i2c. The drivers and utilities for accessing this information under Linux are in the lm-sensors package. You can learn more about that project from secure.netroedge.com/~lm78. The i2c and lm-sensors drivers are included with mainstream kernel sources and are compiled into all mainstream Linux distributions like Red Hat, Debian and SuSE. Perusing the FAQ reveals that different motherboards report differing numbers for temperature and voltage; you can adjust those settings by editing the /etc/sensors.conf file. Debian installed a sample sensors.conf file that's about 20 pages long. Also, the FAQ specifically mentions ASUS P2B motherboards in relation to odd temperature readings—if that's your motherboard, read the FAQ at the above URL. Incidentally, a number of packages use this lm-sensors interface, dæmons that store histories of readings for statistical analysis and graphing, GUI widgets that run in KDE, GNOME or Window Maker panels and so on. At the very least you probably should use the sensors command that will read the settings from your /etc/sensors.conf to adjust the raw readings it gets from the drivers.

—
Jim Dennis

jimd@starshine.org

Quick Download of Many Images

Suppose there's a site called www.foo.org/technical/pics. How can I download only the pictures—let's say the only extension is .jpg—from a Web site using **wget**?

—
Kunthar

kunthar@gmx.net

Here's an example:

```
wget -r -l1 --no-parent -A "*.jpg" \
http://www.server.com/dir/
```

This recursively (-r) downloads all the *.jpg files from the dir directory on the www.server.com Web server up to one level depth. Do a `man wget` for more of this great utility's options.

—
Felipe Barousse Boué

fbarousse@piensa.com

New Products

ATCA-710 SBC

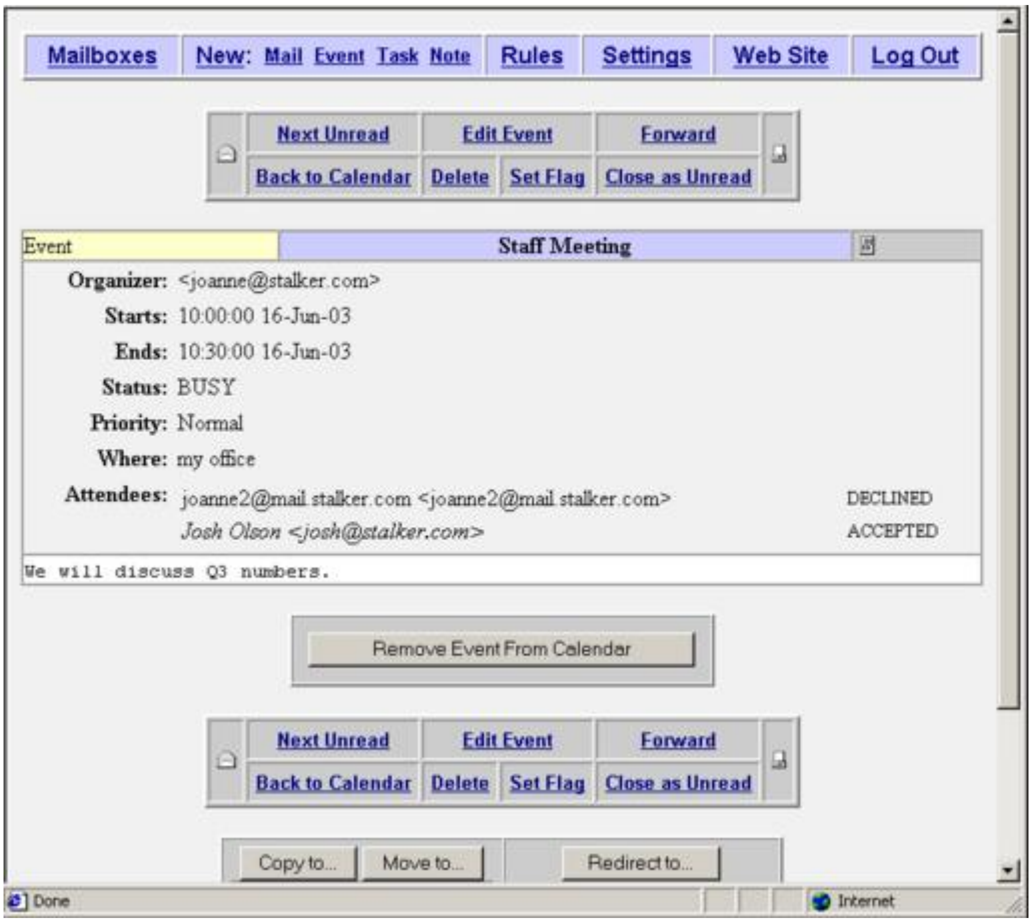
Force Computers announced a new SBC based on the AdvancedATA series of open specifications. The ATCA-710 has a 1.8GHz Pentium 4 processor with support for carrier-grade Linux applications, and it supports 2.6G/3G wireless and broadband wireline applications. Options include a daughtercard with quad PMC slots for I/O expansion using PCI-X and a 12-port GB Ethernet switch. The chipset also supports 1.6GB/sec memory bandwidth, up to 266MHz ECC-protected DDR SDRAM memory and up to 8MB of user and 4MB of boot Flash memory. Other options include redundant connections to the backplane fabric interface and base interface.

Force Computers, 4211 Starboard Drive, Fremont, California 94538, 510-624-5300, www.forcecomputers.com.

CommuniGate Pro 4.1

CommuniGate Pro 4.1 with Groupware is a messaging and collaboration application that provides support for e-mail, address books, shared folders, calendaring and group scheduling tasks company-wide, including Outlook users. CommuniGate Pro also supports standards-based mail and calendaring clients. New to version 4.1 is the ability to complete tasks using a Web interface. Protocols supported include SMTP, POP, IMAP, iCalendar, iTIP and iMIP. CommuniGate Pro server software runs on many OS/hardware combinations that may include UNIX, Linux, Windows, Mac OS X and AS/400. A customizable wireless WML interface is included for mobile devices.

Stalker Software, Inc., 655 Redwood Highway, Suite 275, Mill Valley, California 94941, 800-262-4722, www.stalker.com.



Event | Staff Meeting

Organizer: <joanne@stalker.com>
Starts: 10:00:00 16-Jun-03
Ends: 10:30:00 16-Jun-03
Status: BUSY
Priority: Normal
Where: my office
Attendees: joanne2@mail.stalker.com <joanne2@mail.stalker.com> DECLINED
Josh Olson <josh@stalker.com> ACCEPTED

```
We will discuss Q3 numbers.
```

SciTech MGL 5.0

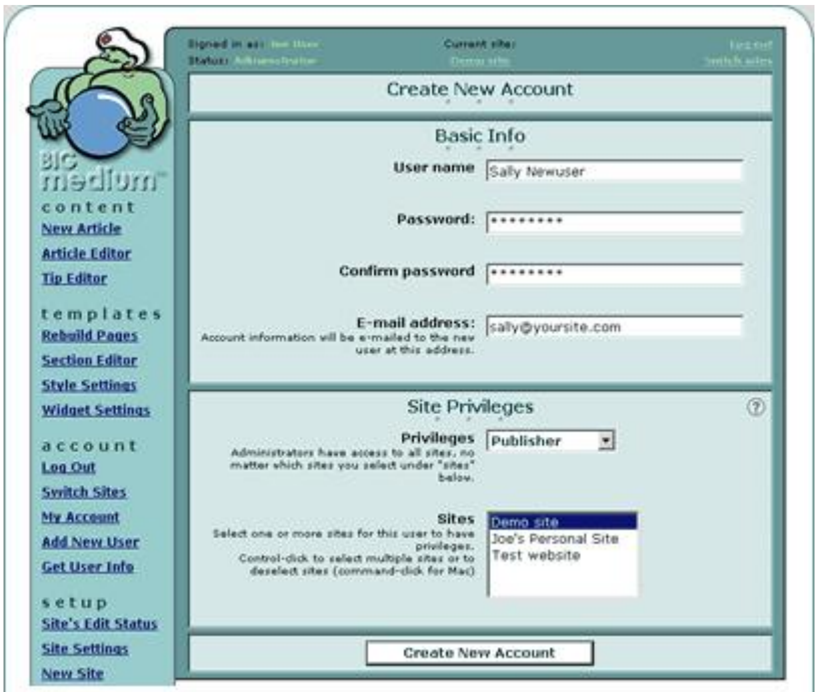
SciTech MGL version 5.0 is a low-level graphics library for Linux, UNIX, QNX, OS/2 and embedded systems that is an alternative to the X Window System. SciTech MGL contains C and C++ graphics code that can be used to produce desktop and embedded applications that operate on a variety of OSes. SciTech MGL provides rapid, low-level rasterization of 2-D and 3-D primitives that can be used to create computer games, user interface software and other real-time graphics applications. Applications are created in SciTech MGL using the OpenGL API in either window or full-screen mode, rendering to VGA, VESA VBE or DirectDraw surfaces. LGPL and proprietary licenses are available.

SciTech Software, Inc., 180 East 4th Street, Suite 300, Chico, California 95928, 530-894-8400, www.scitechsoft.com.

Big Medium 1.1

Big Medium 1.1 is Web-based content management software that allows nontechnical staff to add, edit and publish Web content to multiple sites without needing to know HTML. Designed for Web servers using UNIX and its variants, Big Medium allows rapid setup of new sites based on preconfigured specifications for design, layout and content. Current skins are available for news and product tours, with more available soon. Big Medium's features include duplication and backup of Big Medium sites; a review sites option to browse, edit and delete any content; and automated application on a skin to an existing site. Written in Perl code and using standards-compliant HTML/XHTML, Big Medium is available for a free trial download from demo.globalmoxie.com.

Global Moxie, www.globalmoxie.com.



Create New Account

Basic Info

User name: Sally Newuser

Password: ********

Confirm password: ********

E-mail address: sally@yoursite.com
Account information will be e-mailed to the new user at this address.

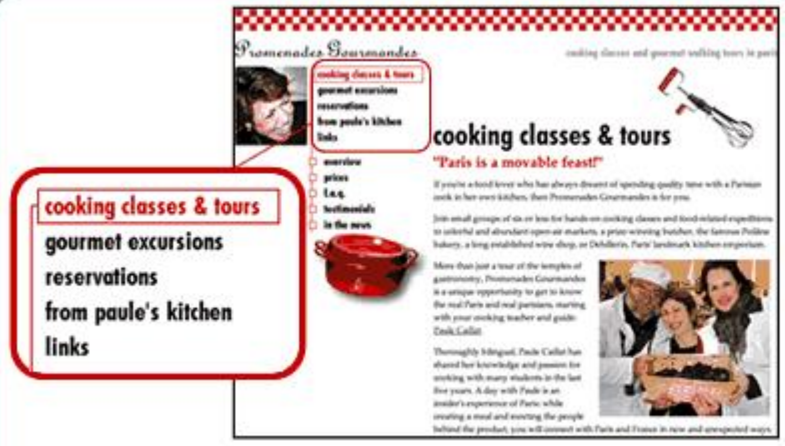
Site Privileges

Privileges: Publisher
Administrators have access to all sites, no matter which sites you select under "sites" below.

Sites: Demo site / Joe's Personal Site / Test website
Select one or more sites for this user to have privileges.
Control-click to select multiple sites or to deselect sites (command-click for Mac)

Create New Account

Add accounts to allow a team to collaborate on site content.



Big Mediumcan manage your site's navigation menus, including rollover effects.

ProStore SATA

The ProStore SATA (serial ATA) storage system offers multiple terabyte storage capacity for data-intensive applications, including clustering, digital content creation and scientific visualization. ProStore contains 36 serial ATA drives in a 4U chassis and can be scaled to 9TB using current SATA drive capacities. 200GB serial ATA disk drives are used in the ProStore, as are the Intel E7501 chipset, dual Xeon 3.06GHz processors and up to four 3ware Escalade 8500 serial ATA

RAID controllers. Additional features include hot-swappable hard drives, power supplies, dual 10/100/1000 Ethernet interfaces and RAID support for reliability and redundancy.

ProMicro, 13880 Stowe Drive, Poway, California 92064, 866-776-6427, www.promicro.com.

In-Reach LX-4048S

MRV Communications announced the In-Reach LX-4048S, a secure console server designed for remote management duties. The LX-4048S is driven by the In-Reach Operating System (IROS), a Linux-based OS tuned for performance, security and reliability. With up to 48 ports available, In-Reach is designed to manage and control large data centers and clustered computing environments. In-Reach offers a secure remote site presence, with serial connectivity, console, power and alarm management capabilities contained in a single box. Internal modem options also are available in all port densities. All models use 32-bit RISC embedded processors. Security features include per-port password protection, RADIUS, PPP dial-back and various other options.

MRV Communications, Inc., 20415 Nordhoff Street, Chatsworth, California 91311, 818-773-0900, www.mrv.com.

AlterPath ACS1

The AlterPath ACS1 is a compact single-port console server used to connect serial devices to a TCP/IP network. Typical applications include branch office management, retail automation and IP enabling of legacy serial devices. Dual PCMCIA slots allow for enhanced functionality with support for many interface cards, such as Ethernet, modem (V.90, GSM, CDMA and ISDN) and wireless LAN. The ACS1 uses PowerPC dual CPUs to transfer data from a 10/100Base-T Ethernet interface to the RS-232/RS-485 serial interface and back, enabling communication between the serial device and the network. The ACS1 supports SSHv2 (Secure Shell) for security in data connections.

Cyclades Corporation, 41829 Albrae Street, Fremont, California 94538, 888-292-5233, www.cyclades.com.